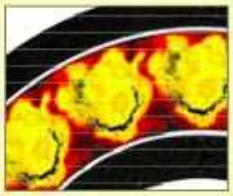
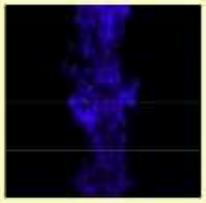
Center for Integrated Turbulence Simulations

2004 Annual Technical Report

Center for Integrated Turbulence Simulations
in collaboration with the
Computer Systems Laboratory

Stanford University

Preface

The aim of the Stanford University Center for Integrated Turbulence Simulations (CITS) is to develop computer tools and physical models to perform detailed simulations of a jet engine. This problem requires modeling the major engine components, including the compressor, combustor, and turbine. The major challenges involve not only developing accurate prediction methods for the individual components, but integrating them into a unified tool in order to study their interactions. The resulting integrated models examine many aspects of jet engine operations, including turbine blade vibrations, rotating stall in the compressor, acoustic instabilities in the combustion process, and heat transfer from the hot combustion products to the turbine blades. CITS has built strong connections with the major jet engine manufacturers; in particular, the long-lasting relationship with Pratt & Whitney has achieved maturity as we transfered to them one of CITS flagship computational tools during last year.

CITS' application efforts are focused on the development of two computer codes: TFLO2000 and CDP. These are multiphysics simulation systems, capable of running in an efficient and scalable way on the largest DOE parallel computers (with thousands of CPUs). Although our overarching problem is the simulation of a realistic jet engine, the CITS codes are general-purpose computational tools that can be used in a variety of applications including fire and plume dynamics, high-speed compressible flow, enviromental and security applications, bioengineerng, etc., of direct interest to DOE National Labs. An example of the general applicability of CITS computational tools is the recent adaptation of both CDP and TFLO2000 in the simulation of advanced rotorcraft in a program sponsored by DARPA.

CITS is organized in four research groups: the turbomachinery group, lead by Prof. Juan J. Alonso; the combustor group, lead by Prof. Parviz Moin; the integration group, lead by Prof. Heinz Pitsch and the Merrimac group lead by Prof. William Dally.

The Turbomachinery group develops the code TFLO2000 and the supporting science for the simulation of the flow in the turbine and the compressor. TFLO2000 is a multiblock structured, compressible flow solver; it has been completely rewritten during the last year and a half and numerous new features have been added including: time-spectral discretization, low-speed and Block-Jacobi preconditioning, real gas effects, conservative interpolation, and adaptive wall functions among others. Refinements of the turbulence models within the Reynolds-averaged Navier-Stokes framework have been introduced in the code and a thorough validation has been carried out in a variety of flow regimes. Work on several novel numerical algorithms is under way in an effort to make TFLO faster than similar codes used routinely in the turbomachinery industry.

The Combustor group research and development effort is directly linked to CDP, our Large Eddy Simulation code. CDP is based on an in-house non-dissipative numerical algorithm for unstructured grids and includes multiphase flow capabilitites for reacting spray and liquid/gas interaction modeling, sophisticated combustion models, and advanced subgrid scale closures. CDP's capabilities were expanded considerably in the last year, and it now includes a a multiple-constraint load balancing algorithm for achieving scalable Eulerian/Lagrangian simulations, an unstructured particle level set solver, and integration with UDM, LANL's

parallel I/O library. CDP is presently being used to simulate reacting flows in realistic Pratt & Whitney combustor geometries at multiple operating conditions and in multiple sectors.

The Integration group is continuing its effort in increasing the size and scope of the TFLO2000/CDP coupled simulations. Major accomplishments during last year included the development of more accurate boundary conditions for the interface between LES-URANS-LES, and the computation of integrated solutions that now include the last stage of the compressor, the pre-diffuser, and the cold flow through the combustor of a P&W engine. An important addition in this area has been the development of a new Python-based integraton interface for both codes. This approach allows us to integrate complex software in a more flexible fashion than the multiple-executable MPI procedure that we have been using so far. In addition, it allows to more quickly reconfigure the software for applications that we had not previously thought of.

CITS' computer science goal, unique among the ASC centers, is the architectural design of a new class of supercomputers for scientific calculations: Merrimac. This effort, which started in 2002 in collaboration with Stanford's Computer Systems Laboratory, CSL, has seen substantial new developments this year with the official release of a detailed (130 pages) architectural specification document. This has spurred the interest of three potential partners interested in the Merrimac chip. Additional collaborations with Lawrence Livermore National Lab and Lawrence Berkeley National Lab have been initiated this year. The area of the development of software tools for Merrimac has also seen substantial improvements with the introduction of a C-like language for easier programming in a stream environment. From an application point of view, focus has been on porting fundamental components of the CITS codes on Merrimac with particular emphasis this year on the PDE solvers in CDP. In addition, porting of scientific codes on Graphics Processor Units (GPU) has been initiated as a viable avenue to explore the possibility of programming on currently available streaming platforms. Initial performance test are very encouraging.

In this seventh operational year, I appointed Gianluca Iaccarino as CITS' new Deputy Director. It is clear that at this stage in its course CITS needs a dedicated executive director to manage the day-to-day operation of the Center, monitor the progress of each group towards their milestones, enhance communication between the groups, and ensure responsiveness to DOE's inputs and concerns. Gianluca brings to this position a great deal of experience in computational engineering and excellent rapport with students, research associates, faculty and our industrial partners.

In an effort to provide better access to CITS research, we have completely redesigned our Web site. The site offers a fresh look at the Center's overall research effort; our computational resources, scientific contributions and computer tools will now be presented in detail. We are currently developing additional components of the web site which will enable us to build (and share with the scientific community) a database of validation cases related to our ASC work.

Finally, one of the most anticipated byproducts of the DOE/ASC investment at Stanford has become a reality: the Institute for Computational and Mathematical Engineering, ICME, began its operations during the fall quarter by enrolling students and offering courses. In line with the multidisciplinary nature of CITS effort, ICME is building bridges to all nine engineering departments at Stanford in addition to the Departments of Mathematics and Statistics. ICME also welcomed two new faculty members this year: Prof. Charbel Fahrat and Prof. Amin Saberi. Prof. Fahrat's field of study is computer aided structural design and numerical analysis and Prof. Saberi is interested in game theory and optimization.

The Center's roster for 2004 is provided in the Appendix. This report is also available online at the Center's Web site (http://cits.stanford.edu).

Parviz Moin
Director

Contents

Preface **i**

1 Combustor Group **1**
1.1 Summary . . . 1
1.2 CDP: Code Status and Development . . . 2
1.2.1 Particle/Mesh Load Balancing . . . 2
1.2.2 Unstructured Particle Level Set Solver . . . 4
1.2.3 Integration with UDM . . . 7
1.2.4 Subgrid Scale Modeling in CDP . . . 7
Dynamic Localization Model – Constrained . . . 7
Variational Multiscale Method . . . 8
1.3 Multiphysics Combustor Simulations . . . 11
1.4 Combustion Models for Large-Eddy Simulations . . . 14
1.4.1 Validation of the Flamelet/Progress Variable Approach . . . 14
Simulation of the Sandia-D Flame . . . 15
Simulation of the UTRC Swirl Combustor . . . 16
Extensions to the FPV model - Heat Transfer Effects . . . 23
1.4.2 Consistent Hybrid LES-FDF scheme for Turbulent Reactive Flows . . . 24
Hybrid LES-FDF Scheme . . . 25
Consistency Requirements . . . 27
Numerical Test - Application to a Bluff-body Stabilized Flame . . . 28
1.4.3 Modeling Soot Formation In Gas Turbine Engines . . . 33
Soot model . . . 33
Validation of soot chemistry and model . . . 35
1.5 Advanced Spray Models in CDP . . . 39
1.5.1 Modeling Droplet Dispersion in LES of Two-Phase Flows . . . 40
1.5.2 Accounting for Volumetric Effects of Dispersed Phase . . . 43
1.5.3 Modeling Liquid Films . . . 48
1.6 Compressible Formulation . . . 53
1.6.1 Motivation for proposed approach . . . 53
1.6.2 Governing Equations . . . 54
1.6.3 Discretization . . . 56
1.6.4 Implementation . . . 58

1.6.5 Results I: Shock tube problem . . . 59
1.6.6 Results II: Isotropic Turbulence . . . 60
1.7 Verification and Validation . . . 62
1.7.1 Staggered Jets in Crossflow . . . 64
Discussion . . . 66
1.7.2 Calculation of flow in a jet engine fuel nozzle rig . . . 68
Results . . . 70
1.7.3 Flame visualization . . . 70
Governing equations . . . 73

2 Turbomachinery Simulations **81**
2.1 Algorithmic Improvements . . . 81
2.1.1 Time-Spectral Approach . . . 82
General formulation of the Time-Spectral Method . . . 82
Stability . . . 84
Sector periodicity . . . 84
NACA 64A010 pitching airfoil . . . 87
Aachen Axial Turbine . . . 87
Conclusions and Future Work . . . 89
2.1.2 Squared Preconditioning . . . 90
2.2 Modeling and Fundamental Studies on Turbulence and Transition . . . 92
2.2.1 TFLO2000 v2-f Model Implementation . . . 94
Improvement of Numerical Scheme for v^2-f . . . 94
2.2.2 Improvements in Modeling Rotating Wall-Bounded Flows . . . 95
Rotating Channel Flow . . . 96
Numerical results . . . 97
2.2.3 Modeling Rotation and Buoyancy Effects on Turbulence . . . 99
2.2.4 Near-Wall Behavior of RANS Turbulence Models and Implications for Wall Functions 100
Numerical Implementation . . . 103
Flow over flat plate . . . 103
Improving numerical results on coarse grids . . . 105
Recirculating flow . . . 106
2.2.5 Transition simulation with compressible LES. . . . 111
Simulation Conditions . . . 111
Averaged Results . . . 112
Effect of the leading-edge . . . 112
2.2.6 DNS of an Infinite Stationary and Rotating Serpentine Passage . . . 113
Methodology . . . 113
Some Results . . . 120
2.2.7 Towards Large Eddy Simulation of Film-Cooling . . . 120
2.3 Simulations in Support of Validation and Verification Efforts . . . 125
2.3.1 Selected Validation Test Cases . . . 125
Fin-plate junction . . . 125
Wing tip vortex . . . 129

2.3.2 NASA Stage 35 and Rotating Stall . . . 130
Simplified Nonlinear Model . . . 130
TFLO2000 and Stage 35 . . . 131
2.3.3 Python Software Integration and Aeroelastic Simulations . . . 137
Python Integration . . . 137
Why Python? . . . 137
Python Tools . . . 138
Numeric-py . . . 138
pyMPI . . . 139
Automatic Fortran Wrapper . . . 139
2.3.4 Aeroelasticity . . . 139
Aerodynamics . . . 140
Structures . . . 141
Fluid Structure Interface . . . 142
Displacement Transfer . . . 142
Load Transfer . . . 142
Multi-block Mesh Perturbation . . . 143
Aeroelastic Iteration . . . 146
Conclusions And Future Work . . . 147

3 Integrated Simulations **155**
3.1 Major Accomplishments . . . 155
3.2 Motivation and Challenges of Integrated Simulations . . . 156
3.3 Code Implementation . . . 157
3.3.1 LES Outflow Boundary Conditions for CDP . . . 157
3.3.2 Anti-Aliasing Filters and Local Phase Correction . . . 160
Anti-aliasing on turbulent flows . . . 164
Anti-aliasing filters: Implementation into CDP . . . 164
Local phase correction: Implementation . . . 166
3.3.3 TFLO 2000 Interface . . . 166
3.4 Basic Research . . . 167
3.4.1 RANS Boundary Conditions for Integrated Simulations . . . 167
Interface boundary conditions . . . 167
3.4.2 Improving Unsteady Compressible Boundary Conditions . . . 177
Improved boundary conditions . . . 177
Improved boundary conditions: test-case and flow solver . . . 178
Improved boundary conditions: Results . . . 179
3.4.3 Separated Diffuser . . . 180
3.5 Application of Integrated RANS-LES to Complex Geometries . . . 180
3.5.1 NASA Stage 35/Diffuser . . . 182
3.5.2 Geometry . . . 182
3.5.3 Results . . . 183
3.5.4 Pratt & Whitney Engine Geometry . . . 183
3.6 Summary . . . 188

3.7 Future Work Plan . . . 189

4 The Merrimac Streaming Supercomputer 195
4.1 Project Summary . . . 195
4.2 Overview of the Merrimac Architecture . . . 197
4.2.1 Node Architecture . . . 198
4.2.2 System Architecture . . . 199
4.2.3 Fault Detection and Reliability . . . 202
4.2.4 Overall Operation . . . 202
4.3 Merrimac Specification . . . 203
4.3.1 Kernel ISA . . . 203
LRF Related Changes . . . 203
Iterative Operations Support . . . 204
SRF Access Modes . . . 204
Merrimac Floating-Point Support . . . 205
4.3.2 Micro-controller ISA . . . 207
Communication with Other Units . . . 208
Instruction Sequencing . . . 209
Looping Behavior . . . 209
Double Buffering Long Streams . . . 211
4.3.3 Stream ISA . . . 212
Stream Controller Control Instructions . . . 213
Stream Memory Operations . . . 213
Kernel Control Operations . . . 214
Scalar Stream Instruction Encoding . . . 214
4.4 Architectural Enhancements . . . 215
4.4.1 Iterative Unit . . . 215
4.4.2 LRF Organization . . . 216
4.4.3 Enhancements to the Stream Register File . . . 217
Unified control and address generation for indexed and sequential stream accesses . 217
Support for read/write (scratchpad) data . . . 218
Efficient conditional stream accesses . . . 218
Support for high-reliability mode . . . 218
4.4.4 Cache Coherency and Consistency . . . 219
Stream Cache Coherency and Consistency . . . 219
Scalar Cache Coherency and Consistency . . . 220
4.4.5 Atomic Memory Operations . . . 221
Atomic Memory Operation Unit . . . 221
Atomic Memory Operation in Multi-node . . . 223
4.5 Simulation Environment . . . 223
4.5.1 Scalar Processor Integration . . . 223
4.5.2 Multi-Node Support . . . 224
Simulator Extension Multi-Node Configurations . . . 224
Atomic Memory Operation Scalability Study . . . 224

4.6 Programming Support . . . 226
4.6.1 High Level Compiler (HLC) . . . 226
Current Status of HLC . . . 228
4.6.2 LLC . . . 229
Kernels . . . 229
4.6.3 Scalar . . . 230
Resource Allocation . . . 232
Status . . . 232
4.6.4 Legacy Program Support . . . 232
4.7 Merrimac applications . . . 235
4.7.1 StreamMD . . . 236
4.7.2 StreamCDP . . . 243
4.7.3 Brook for GPUs . . . 245
4.7.4 Gromacs on GPU . . . 247
Shake Algorithm in Brook . . . 251
4.7.5 Finite Element method for solid mechanics . . . 253

CITS Roster **260**

Chapter 1

Combustor Group

1.1 Summary

In a jet engine combustor, pressurized air from the upstream compressor is reacted with atomized liquid fuel to produce the combustion products that drive the downstream turbine. The combustor group is using the Large Eddy Simulation (LES) approach to simulate the reacting multiphase flow in the combustor because of its demonstrated superiority in predicting turbulent mixing when compared to the Reynolds Averaged Navier-Stokes (RANS) technique.

With regard to large scale multi-physics simulations, the group's major accomplishments this year include:

- The capabilities of the unstructured LES code CDP have been significantly expanded to include particle/mesh load balancing, a particle level set solver, integration with LANL's UDM parallel I/O library, and implementation of additional subgrid scale closures including the variational multiscale method (§1.2),
- Several multiphase reacting flow simulations of realistic Pratt & Whitney combustors are being made, including two single sector calculations at cruise and high power conditions, and a three sector simulation to investigate the assumption of sector periodicity and pattern factor at the turbine inlet (§1.3),

A large portion of the combustor group works on more fundamental problems in combustion science, spray physics, and algorithm development. The methods and models developed in this research will eventually be implemented in CDP, however most of this reported research makes use of fast and accurate structured codes for verification and validation. With regard to this supporting science, the group's major accomplishments this year include:

- In combustion science, the flamelet/progress variable model has been extended to handle extinction and non-adiabatic conditions; the LES-Filtered-density function (LES-FDF) approach has been pursued to understand the evolution of sub-filter FDF in reacting flows; and a flamelet-based model has been implemented for the prediciton of NO_x (§1.4),
- in spray physics, a model has been developed to account for the effect of subgrid fluctuations on particle lift and drag forces; the point-particle formulation has been extended to account for the finite-size

effects in regions of dense spray; the dense spray model along with a film model based on lubrication theory has been developed and evaluated (§1.5),

- in algorithm development, a non-dissipative, robust, implicit algorithm has been proposed for direct numerical and large-eddy simulation of compressible turbulent flows (§1.6).

Finally, at the end of this chapter a section on verification and validation specific to the Combustor group is included.

1.2 CDP: Code Status and Development

CDP[1] is the flagship unstructured LES code being developed by the combustor group for reacting multi-phase flow simulation in complex geometry. The code is written in fortran 90, is fully parallel (including preprocessing) and is based on the following discretization and models:

1. Second-order energy-conserving finite volume discretization of low Mach number variable density equations,
2. Lagrangian particle/parcel tracking for liquid spray phase with evaporation and stochastic breakup,
3. Dynamic Subgrid Scale closures for momentum and all scalars,
4. Steady flamelet/progress variable combustion modeling.

In the push towards larger and more complex simulations, a number of new capabilities have been implemented in CDP in the last year and are presently in various states of verification and validation, including:

1. Particle/Mesh load balancing capability,
2. Implementation of an unstructured Particle Level Set solver,
3. Integration with UDM, LANL's parallel I/O libraries for large unstructured data sets,
4. Implementation of alternative subgrid-scale closures, including the variational multiscale method.

These additional capabilities are discussed in more detail in the following subsections.

1.2.1 Particle/Mesh Load Balancing

The Lagrangian particle tracking and associated spray modules of CDP are developed based on the particle-in-cell model. That is, the droplets are located on the same processor as their containing control volume. While a separate partitioning of the Lagrangian and Eulerian spaces is indeed possible, the particle-in-cell approach was chosen because it avoids any additional message exchanges required to get gas phase data at the particle locations, or vice versa. It also simplifies the development of new breakup or coalescence models.

[1] CDP is named after Charles David Pierce (1969-2002)

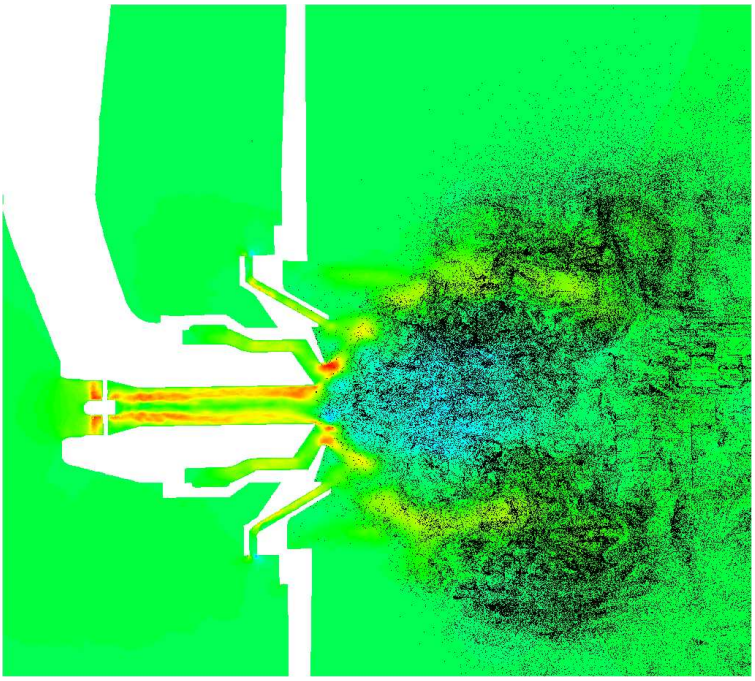


Figure 1.1: Spray evolution from a P&W injector, 3.5 million droplet parcels representing over 14 million droplets.

Based on this decision, however, the geometric localization of particles around the fuel injectors can create significant load imbalance when a single-constraint gas-phase partitioning is used. Synchronization points in the time advancement can lead to the many of processes waiting for those with the majority of particles to complete their ODE integration. By using a dual constraint partition, which attempts to balance both the number of control volumes and the number of particles on each processor, this idle time can be almost completely eliminated at the cost of a slight reduction in parallel efficiency.

To illustrate, consider the case shown in figure 1.1, a realistic P&W injector. This case has been calculated in the past to validate the stochastic breakup model against experimental measurements [7]. In this case we use it to illustrate the codes new particle/mesh load balancing capability.

Figure 1.2 shows the distribution of both control volumes and particles across 96 processors using a single constraint partition. In this case the ParMETIS multilevel partitioning library [42] was used in a single constraint mode to generate the partition. Figure 1.3 shows the distribution of both control volumes and particles across the processors using a dual constraint partition, where the ParMETIS routine was this time given the dual constraint of balancing both the number of control volumes and the number of particles per cell.

Figures 1.4 and 1.5 illustrate how the single and dual constraint partitions differ in appearance and complexity. Table 1.1 summarizes the net effect on wall clock time per time step. Overall, the dual constraint partition led to a reduction in time per time step of about 40%. For more complex breakup models or significantly higher particle densities, this speedup is expected to be even more significant.

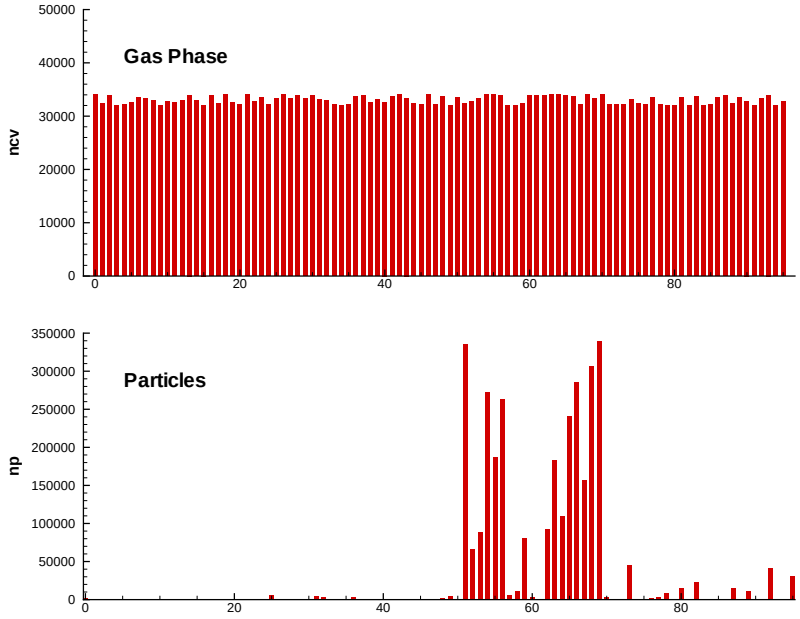


Figure 1.2: Distribution per processor of control volumes (top) and particles (bottom) for single constraint partition.

1.2.2 Unstructured Particle Level Set Solver

The level set method is a surface capturing method required in CDP as a component of both advanced combustion modeling (G-equation models for premixed or partially-premixed combustion) and for the large-surface structure model for primary atomization. An unstructured particle level set solver has been implemented in CDP based on the method developed for structured grids by Enright *et al.* (2002). The method uses CDP's Lagrangian particle capabilites to correct the Eulerian level set field, and significantly improve the conservation properties of the method. Figure 1.6 compares the performance of the method with adn without particle corrections for the Zalesak disk problem, where a notched disk is rotated about the origin once using a solid body rotation. Any chance in the shape of the disk can thus be attributed to the numerics.

Table 1.1: Load Balancing: Comparison of wall clock time per time step.

	single constraint	dual constraint
gas phase	8.0	8.5
particle phase	7.8	1.1
total	15.8 s	9.6 s

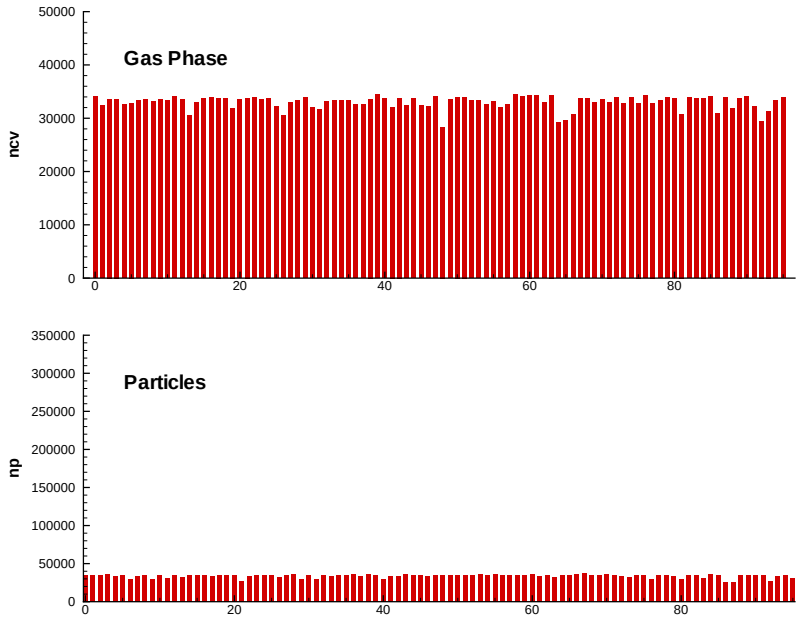


Figure 1.3: Distribution per processor of control volumes (top) and particles (bottom) for dual constraint partition.

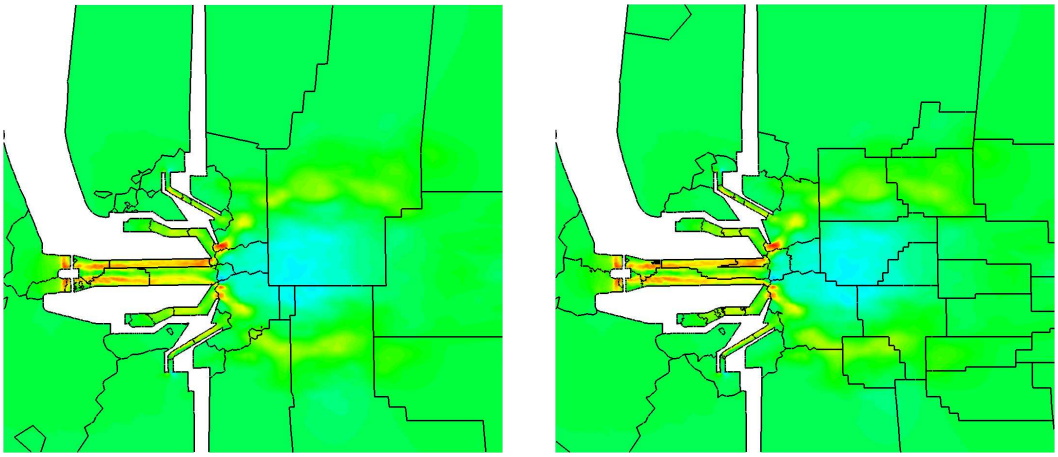


Figure 1.4: Comparison of partitions on plane cut through vertical centerline. Left is the single constraint partition (edge cut 124170), right is dual constraint partition (edge cut 176686, +42%).

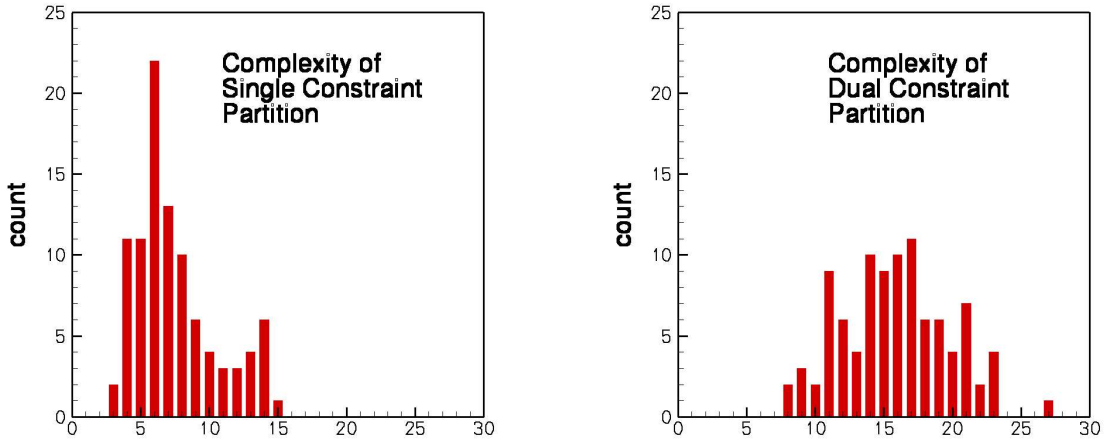


Figure 1.5: Comparison of complexity for single and dual constraint partitions. Here complexity refers to the number of immediate partition neighbors that each partition must communicate with.

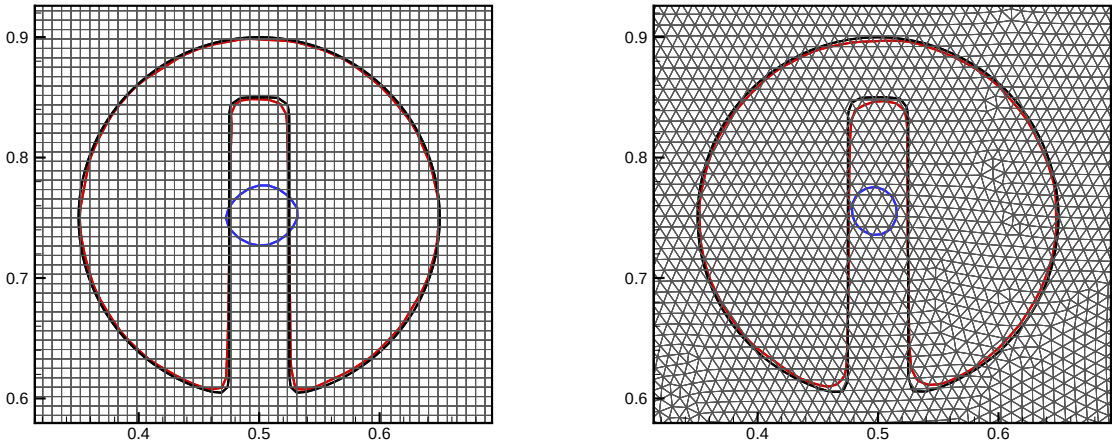


Figure 1.6: Zalesak's disk problem after one full revolution on structured and unstructured grids. Black lines are initial condition, red is particle levelset method, blue is levelset without particle corrections.

1.2.3 Integration with UDM

Recently, CDP was integrated with LANL's Unified Data Model (UDM) libraries for parallel I/O. UDM reads and writes self-descriptive portable binary files, and supports the fully unstructured meshes used by CDP. The package also provides tools to visualize the resulting data sets in the commercial visualization package EnSight.

Table 1.2: Parallel I/O rates on Frost for UDM/CDP integration.

nproc	file size, B	write time, s	rate, MB/s
16	88846420	0.44	201.9
16	88846420	0.48	185.1
16	88846420	0.34	261.3
16	88846420	0.48	185.1
16	88846420	0.57	155.9
16	88846420	0.36	246.8
16	88846420	0.48	185.1
128	839061868	1.39	603.7
512	847424936	17.9	47.3
512	847424936	1.875	452.0
512	847424936	19.7	43.0

During a recent dedicated run on LLNL's Frost, a number of UDM files were written and some timings taken. These are reported in table 1.2.

1.2.4 Subgrid Scale Modeling in CDP

Unless otherwise stated, all CDP simulations reported in this document have used the dynamic Smagorinsky model for subgrid scale stress and subgrid scalar flux closures. The averaging required to avoid numerical instabilities associated with negative subgrid scale viscosities/diffusivities is performed by either averaging along Lagrangian flow lines [27], or by simply applying the mean-preserving test filter a few times (essentially local averaging) and then clipping negative coefficient values. For flows where there is one or more homogeneous directions, or an ensemble averaging is possible among various instances of statistically identical regions of the flow (the 18 repeated sectors or the full P&W combustor, for example), case-specific filters can be implemented as well. In any case there remains, in a sense, a free parameter that must be specified (e.g. the Lagrangian integration time scale, or the extent of the local averaging required).

In an attempt to further reduce the user's parametric input into subgrid scale modeling (either implicit or explicit), two alternate closures were implemented in CDP and are described in the following subsections.

Dynamic Localization Model – Constrained

In the Dynamic Localization Model of Ghosal *et al.* (1995), an integral equation is solved in each time step to compute a local value of the Smagorinsky constant that is consistent with the Germano identity. The iterative solution of the integral equation along with the suggested preconditioning were implemented in

CDP for the case of the constrained model, and very good results were achieved for turbulent channel flows. The resulting system could be converged adequately in about 5 iterations, equivalent to less than 2% of the overall cost of the CDP incompressible flow simulation.

When the same implementation and preconditioning were applied to more complex problems on unstructured grids with anisotropic tetrahedral and prism elements, however, the integral system was found to become significantly stiffer. In several cases more than 25 iterations per time step were required, thus increasing the cost of the overall simulation significantly. Recognizing that the stiffness problem could be overcome by alternate preconditioning and/or better solvers, more complex simulations were performed for the case of the staggered jets in crossflow, and the frontend spray rig. In both cases, the model admitted substantial local fluctuations in the computed coefficient value that, while always positive, resulted in increased rms turbulence velocities, particularly in areas where the simulation was under-resolved. The constrained version of the model makes no assumption on the smoothness of the coefficient variation, and this appears to have detrimental effects on the simulation when under-resolved, as is often the case.

Variational Multiscale Method

In the past year, a version of CDP has been developed that enables the variational multiscale large eddy simulation of turbulent flows. Two important aspects characterize variational multiscale LES. Firstly, a variational projection separates scale ranges rather than a spatial filter as in traditional LES. Secondly, the (direct) influence of the subgrid-scale model is confined to the small resolved scales. Thus, the large resolved scales are solved as a direct numerical simulation (DNS), i.e. without any (direct) influence of the modeling term. Of course, the large resolved scales are still influenced indirectly by the subgrid-scale model due to the inherent coupling of all scales. In [16], a general class of scale-separating operators based on combined multigrid operators and suited for variational multiscale LES both with dynamic and constant-coefficient based subgrid-scale modeling is proposed. These operators may also be used for the dynamic modeling procedure in a traditional LES. For this purpose, a two-grid structure has been implemented in CDP. Starting from a coarser 'parent' grid, which defines the resolution level for the large scales of the problem, a 'child' grid is obtained by an isotropic hierarchical subdivision procedure.

Dynamic multiscale (DMS) and non-multiscale (DS) as well as constant-coefficient based multiscale (CMS) and non-multiscale (CS) Smagorinsky models based on various scale-separating operators have been tested for the case of a turbulent channel flow in [16]. Figs. 1.7 and 1.8 depict the mean streamwise velocity profile and the turbulent kinetic energy, respectively, for various methods based on a projective multigrid (PM) scale separation at $Re_\tau = 590$ with 64 control volumes in each coordinate direction. A substantial numerical error introduced by a second-order accurate method in combination with a relatively coarse discretization comes into play for all methods applied. However, the lower accuracy of the basic method affects CMS-PM to a far lesser extent than the other methods towards the channel center. It is considerably closer to the DNS profile of [25] in this part of the channel than a calculation with no model (NM), DS-PM, and DMS-PM. DMS-PM shows no improvement in comparison to NM, and DS-PM performs even slightly worse. Despite the higher accuracy in the inertial layer (usually expected to start at $x_2^+ = 30$), CMS-PM slightly underpredicts the velocity profile in the buffer layer (usually expected to range from $x_2^+ = 5$ to $x_2^+ = 30$). The profile for the turbulent kinetic energy shows a much better agreement for the height of the peak in comparison to the DNS profile than the other methods. Of course, CS is far worse than all other methods with respect to both aspects of the flow. Besides the remarkable results, CMS-PM represents a very efficient methodical combination with regard to the important aspect of computational cost.

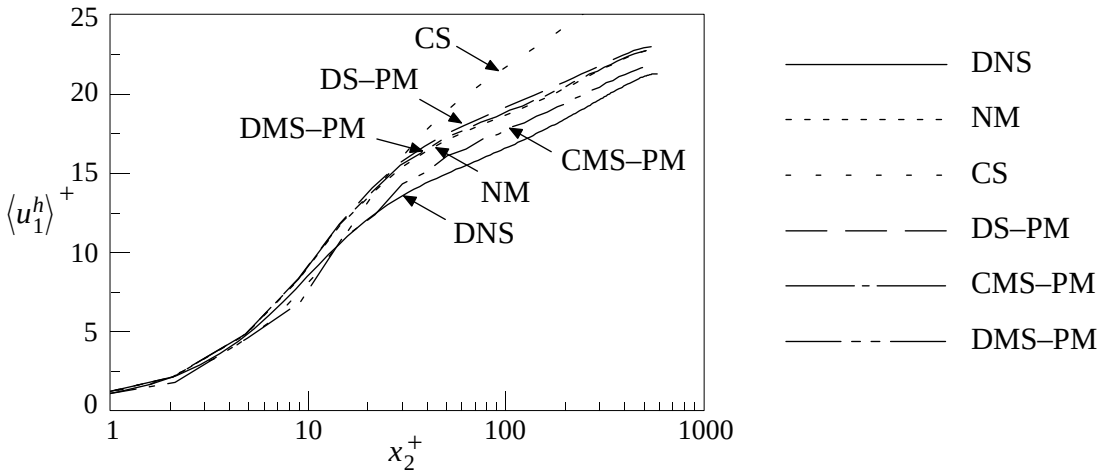


Figure 1.7: Comparing methods (pm-separation): mean streamwise velocity for turbulent channel flow at $Re_\tau = 590$.

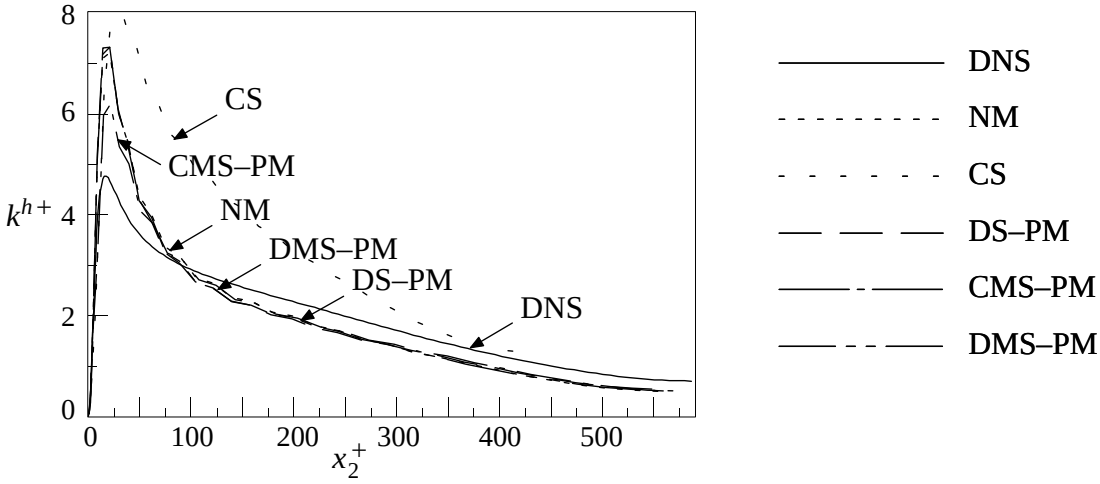


Figure 1.8: Comparing methods (pm-separation): turbulent kinetic energy for turbulent channel flow at $Re_\tau = 590$.

The variational multiscale LES using the aforementioned scale-separating operators is presently being applied to turbulent flow in a planar asymmetric diffuser. Furthermore, more advanced dynamic modeling procedures within the multiscale environment for completely inhomogeneous flow situations are currently in progression.

1.3 Multiphysics Combustor Simulations

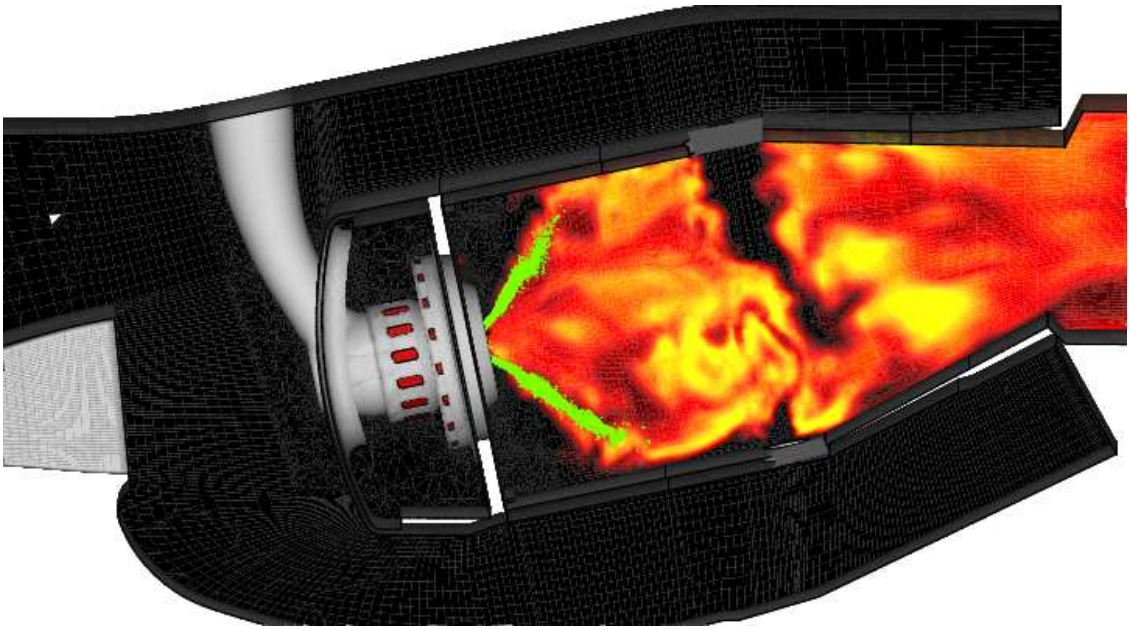


Figure 1.9: Instantaneous snapshot of temperature contours in PW combustor superimposed with droplet scatter plot in the symmetry plane.

A multi-scale, multi-physics simulation of turbulent reacting flow in a realistic Pratt & Whitney combustor was performed in FY04. This simulation includes all the complex models for spray breakup, evaporation, and turbulent combustion described in §1.2. This simulation will serve as the first validation study of the reacting multiphase flow solver (CDP) in complex combustor geometry. The simulation is performed for a single injector which represents one sector of the full combustor containing 18 injectors.

Liquid fuel (Jet-A) at enters the combustion chamber through an annular ring at the injector exit. This liquid film is approximated by large drops of the size of the annulus radius. These drops are convected by the surrounding hot air, they break, evaporate, and the fuel vapor thus formed mixes with the surrounding air giving a non-premixed spray flame. The computational grid consists of 1.9M hybrid elements (hexes, pyramids, and tets) with fine resolution close to the injector. The Reynolds number based on inlet conditions and a reference length scale of 1 inch is around 500,000 and becomes 150,000 in the main (core) swirler channel.

Figure (1.9) shows an instantaneous snapshot of temperature on the $z = 0$ symmetry plane. A scatter plot of droplets forming a conical spray is also shown. A highly complex oscillatory flame is observed in this simulation. The high temperatures near the nozzle are reduced by large mass-influx through the dilution holes. Figure (1.10) shows instantaneous snapshots of temperature in various cutting planes through the combustor.

Another reacting flow simulation in PW single-sector geometry at high-power conditions has been initiated to investigate the sensitivity of inflow liquid fuel rates on the predictions.

In FY04, we also initiated a multi-sector (three-sectors) calculation of the PW combustor to study any

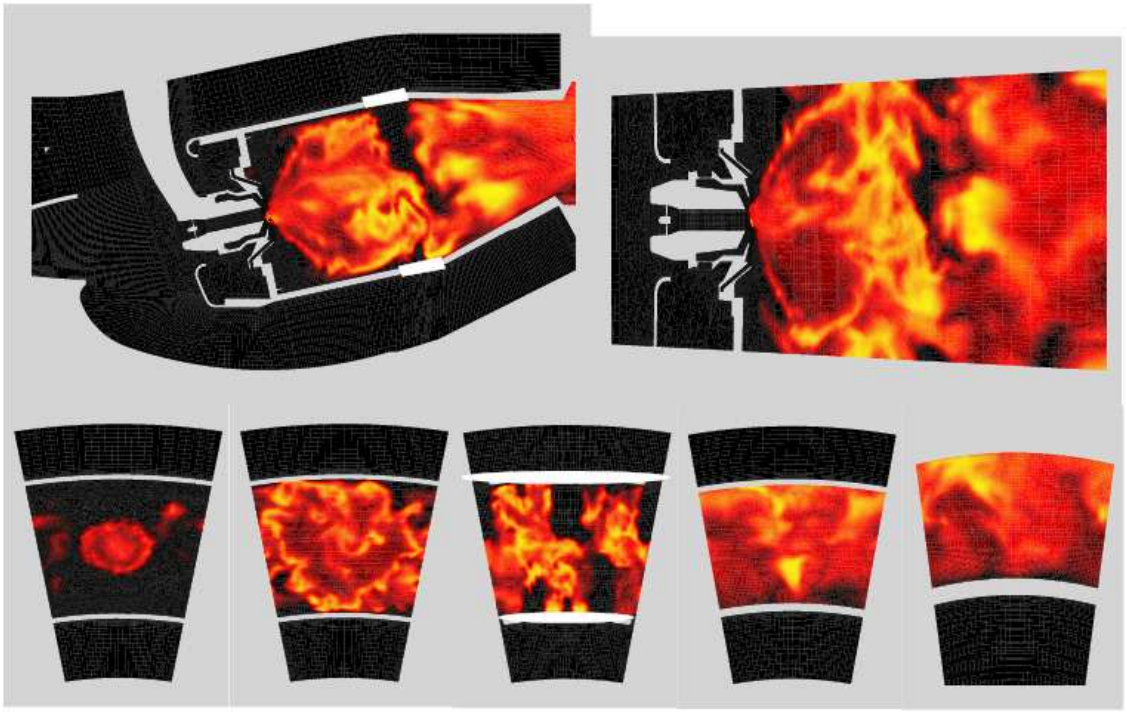


Figure 1.10: Instantaneous snapshot of temperature contours in PW combustor in different cutting planes.

interactions between adjacent sectors that might lead to the observed pattern factor of hot streaks at the turbine inlet plane. Figure (1.11) shows a snapshot of temperature in the multi-sector simulation.

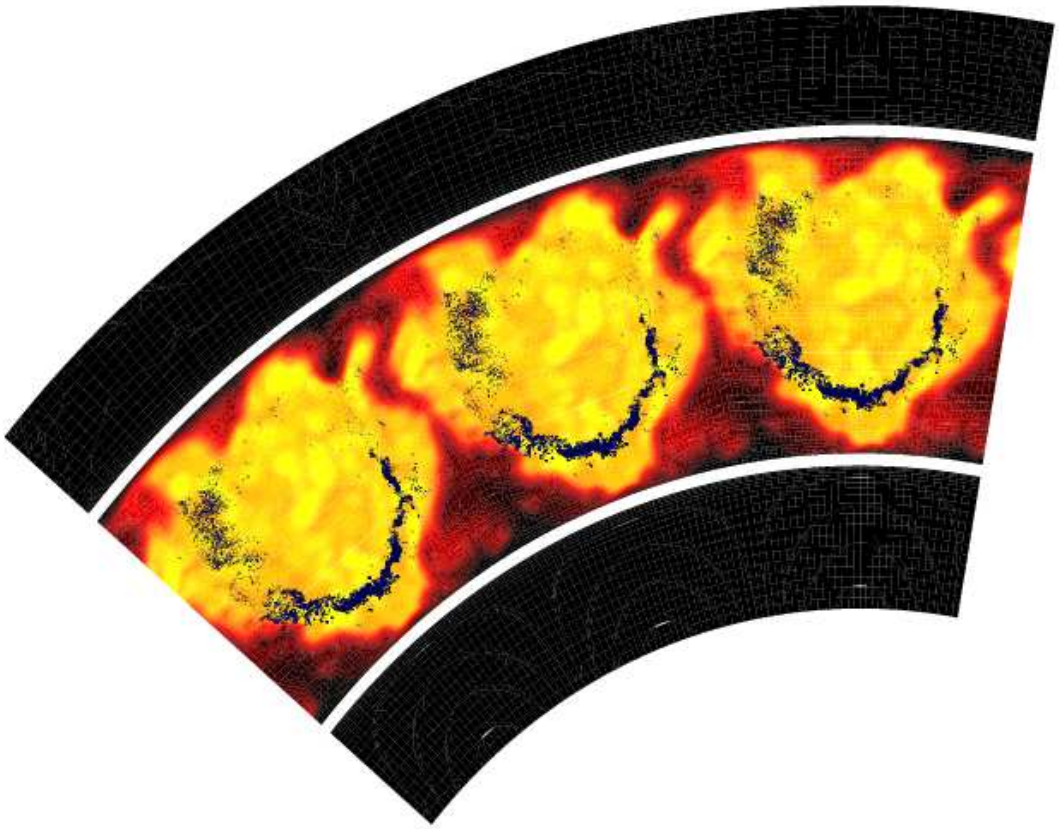


Figure 1.11: Instantaneous snapshot of temperature contours in PW combustor superimposed with instantaneous spray in three sectors.

1.4 Combustion Models for Large-Eddy Simulations

Simulation of turbulent combustion in practical combustors like aircraft engines require general and robust models that are valid in a wide-range of operating conditions. Complex phenomena like extinction-reignition as well as lifted flames need to be handled. In addition, the predictive computation of pollutants like CO, NO_x and soot levels is important for these simulations to be of practical use. With these goals in mind, several next-generation combustion models are being developed here. In the previous years, the flamelet-progress variable (FPV) approach of Pierce and Moin was extended to model extinction in flames. *A priori* analysis with DNS data showed excellent promise. A computationally inexpensive yet detailed model was formulated and tested with the DNS database. In addition, extensive research was also carried out in the area of conserved scalar modeling. Since the sub-filter variance of mixture-fraction is critical to flamelet-based combustion models, an accurate method for the evaluation of the scalar moments is required. Using a modeled transport equation for the second moment of the mixture-fraction, a robust and mathematically well-defined system of PDEs for the conserved scalar was devised. The model was tested both in simple flow configurations and complex combustor-type geometries, with all tests producing excellent results.

During this period, the combustion modeling group has focused its efforts on three different areas. First, the FPV has been extended to handle both extinction as well as non-adiabatic conditions. The model has been tested with the well-known Sandia D flame experiment. This provides a base case to analyze future improvements to the model. Simultaneously, the FPV model has been modified to handle non-adiabatic conditions like heat-transfer to the combustor walls. Again, the model has been implemented in a LES code and tested in a practical combustor configuration. As an alternative approach, the LES-Filtered-density function (LES-FDF) approach based turbulent combustion modeling has been pursued to understand the evolution of sub-filter FDF in reacting flows. A robust numerical scheme has been formulated and tested in a complex flow configuration. Thirdly, pollutant prediction has been studied in the context of NO_x emissions and soot formation. A flamelet-based model has been implemented for the NO_x prediction. Soot modeling is a very complex phenomena and a detailed analysis has been pursued using one-dimensional configurations. Model predictions shows promise and will be extended to multidimensional reactors. Thirdly, a new computational scheme for evaluating the sub-filter mixture-fraction variance has been formulated. Preliminary studies using a DNS database show promise. Current work focuses on extending this model to practical combustor geometries as well as more rigorous testing through the use of experimental flame configurations.

Many of the models developed have been implemented and tested in the combustor code. Several validation studies have been performed using the aircraft engine model combustor. Based on the model predictions and experimental data, further extensions and model developments are being carried out.

1.4.1 Validation of the Flamelet/Progress Variable Approach

In spite of the vast algorithmic and model developments in LES of turbulent combustion, direct validation with experimental data is still limited. During this year, we have focused our efforts on a validation procedure for combustion modeling. In order to assess the performance of a model under a wide range of operating conditions, we use several canonical experimental configurations to determine the model performance and accuracy. Here, we present the validation of the Flamelet/Progress variable (FPV) approach using two different flame configurations.

Simulation of the Sandia-D Flame

The FPV model in the original formulation as proposed by Pierce and Moin [38] is applied in a numerical simulation of the Sandia Flame D experiment. In this canonical flame configuration a partially premixed methane-air jet flame is stabilized by a premixed pilot flame and surrounded by a coflow of air at ambient conditions. This flame shows only a moderate level of local extinction and re-ignition. The comprehensive database of this well-characterized flame configuration serves as a method of testing the ability of presently used and developed models for turbulent non-premixed combustion.

The purpose of this study is

- to apply the FPV model to other unconfined geometry configurations with existing experimental databases, allowing for further assessment of the capabilities of FPV in an *a posteriori* sense,
- to establish a reference simulation for testing further model improvements to the probability density function (PDF) closure and the extension of the FPV model to the use of unsteady flamelets, as has been proposed in previously conducted *a priori* studies [7, 19].

The FPV combustion model has been already extensively described in a previous ASC-report [7] and other works [38, 41, 19] and only the geometrical setup, initial and boundary conditions will be briefly described in the following.

For this simulation a structured low-Mach-number LES code has been used [41]. The conservation equations for mass, momentum, mixture fraction, and a progress variable are solved in cylindrical coordinates. This simulation has been conducted on 32 processors on the ASCI Frost HPC machine in approximately 21,500 CPU-hours.

For a sufficient numerical resolution of a wide range of turbulent scales, a sufficiently fine underlying numerical grid is required. The geometry has been non-dimensionalized by the jet nozzle diameter $D = 7.2$ mm and the computational domain is $80D \times 23.5D \times 2\pi$ in the axial, radial, and circumferential direction, respectively. The computational grid consists of approximately 2.5 million cells with a finer grid spacing close to the centerline and near the nozzle.

After the flow field reaches statistically stationary state, statistics for the velocity vector, mixture fraction and all chemical species of interest are collected over a time interval $T \approx 200\ D/U_J$, where $U_J = 49.6$ m/s is the characteristic jet exit velocity.

Mean and root mean square values of axial velocity component u, mixture fraction Z, and temperature T along the centerline are shown in Fig. 1.12. The mean axial velocity and its rms value are in reasonable agreement with experimental data. The over-prediction of the mixture fraction in the lean region, $Z < Z_{\text{st}} = 0.35$, occurring after approximately $x \sim 45\ D$, results in elevated mean temperature and mass fractions for CO and CO_2 (not shown here). A further numerical simulation using a combined radiation-FPV model shows that a further temperature reduction of about 50 K can be obtained by accounting for radiative heat transfer. This shows that radiation, as relatively slow process, is of considerable importance for $x > 45\ D$.

A comparison of time-averaged mean quantities of chemical species conditioned on mixture fraction with the experimental data at different axial stations, $x/D = 15$, 30 and 45, is shown in Fig. 1.13. The conditional mean temperature obtained from the simulation is in good agreement with the experimental data. The fuel and oxidizer consumption in the rich part is slightly over-predicted, leading to an over-prediction of water, molecular hydrogen, and carbon monoxide, respectively. Even though CO and H_2O are over-predicted, carbon dioxide, as a stable reaction product of the water-gas shift reaction, is in excellent agreement with experimental data.

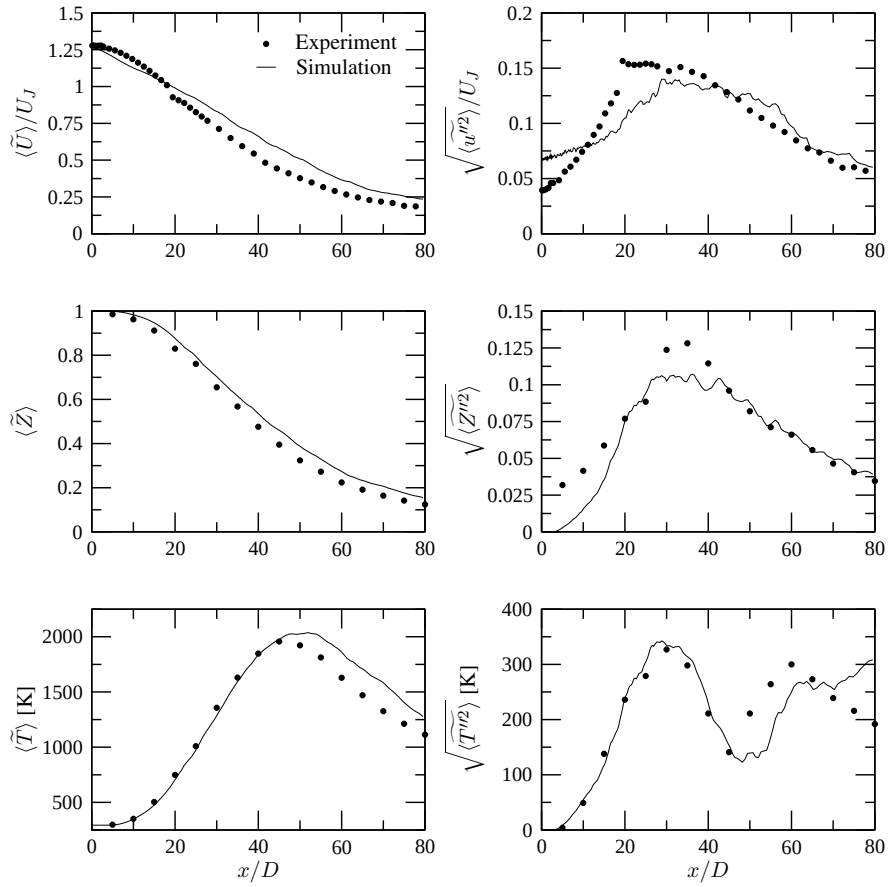


Figure 1.12: Comparison of mean and rms value of axial velocity component, mixture fraction, and temperature (lines) with experimental data (symbols) for centerline.

Simulation of the UTRC Swirl Combustor

A large-eddy simulation has been performed for the test case number 6 of Spadaccini et al. [51]. The geometry is a co-axial dump combustor designed as a simple geometry model of an aircraft engine combustor. The experiment is operated at ambient pressure with high temperature air and natural gas as a fuel. The fuel is introduced through the inner nozzle, swirling air through a co-axial nozzle. The simulation has been performed on a structured cylindrical mesh consisting of $320\times150\times32$ computational cells in axial radial and azimuthal direction. The mesh and the size of the domain are shown in Figs. 1.14 and 1.15.

The results of the reactive flow simulation of the test case number 6 of Spadaccini using the flamelet/progress variable model are shown in Figs. 1.16–1.22. The simulation has been performed using as inflow condition the results of a separate LES of a turbulent periodic pipe flow at the appropriate Reynolds number. Results of the combustor simulation are shown and compared with all available experimental data for the mean and RMS of the axial and swirl velocities in Fig. 1.16, for the temperature in Figs. 1.17 and 1.18, for the CO_2 mole fractions in Figs. 1.19 and 1.20, and for the CO mole fractions in Figs. 1.21 and 1.22.

The profiles for the axial velocity close to the nozzle shown in Fig. 1.16 show that both the mean

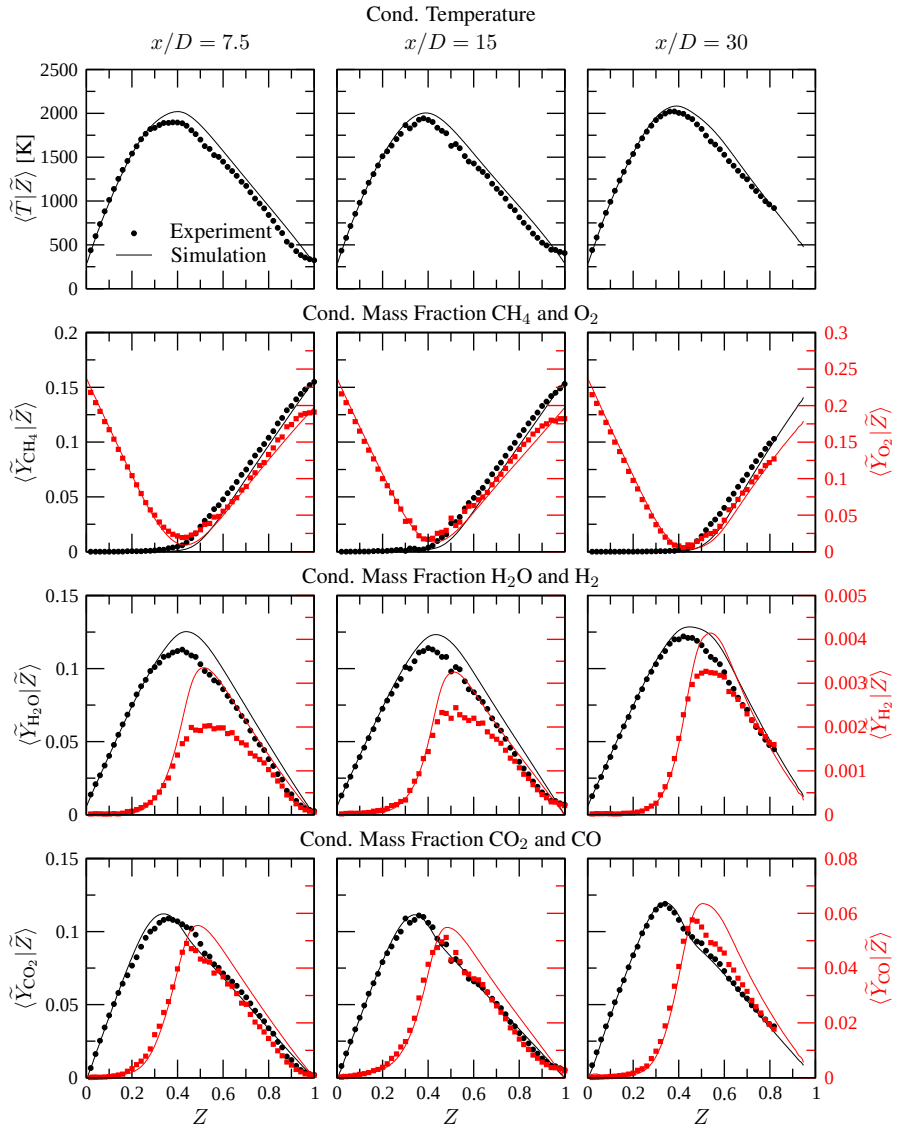


Figure 1.13: Comparison of mean temperature and mean product mass fractions conditioned on mixture fraction obtained from the simulation (lines) with experimental data (symbols).

and the RMS of the axial velocity are essentially well predicted. However, the swirl cone angle seems to be slightly underpredicted. This also seems to be observable from the swirl velocity shown in Fig. 1.16. Further downstream, at $x/R = 1.06$, again, both the mean and the RMS of the velocity components are essentially well predicted. Only the computed mean tangential velocity is lower than the experimental data. However, since the overall swirl velocity is well computed upstream and downstream of this location, the errors might be attributed to the experimental uncertainty. At the even further downstream position, shown in Fig. 1.16, again, all velocity components are well predicted. Only the centerline axial velocity component

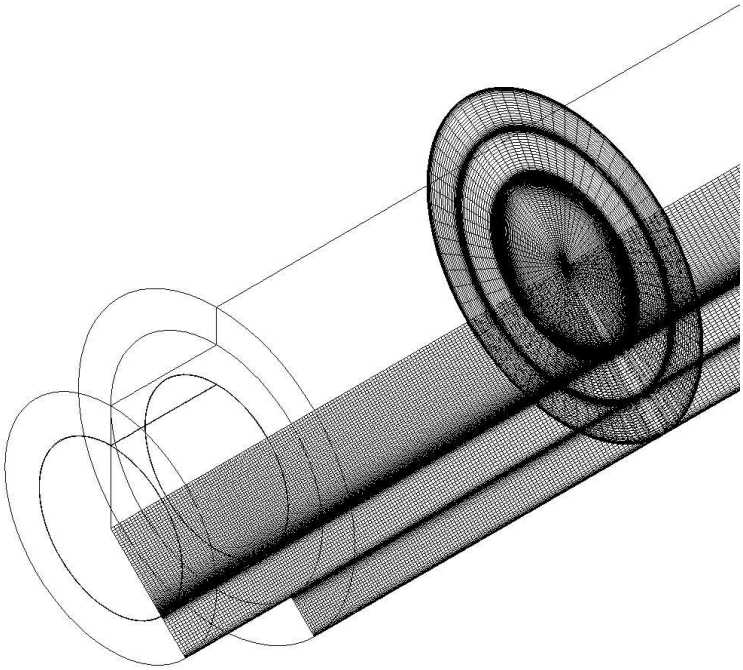


Figure 1.14: Computational mesh.

Figure 1.15: Computational domain.

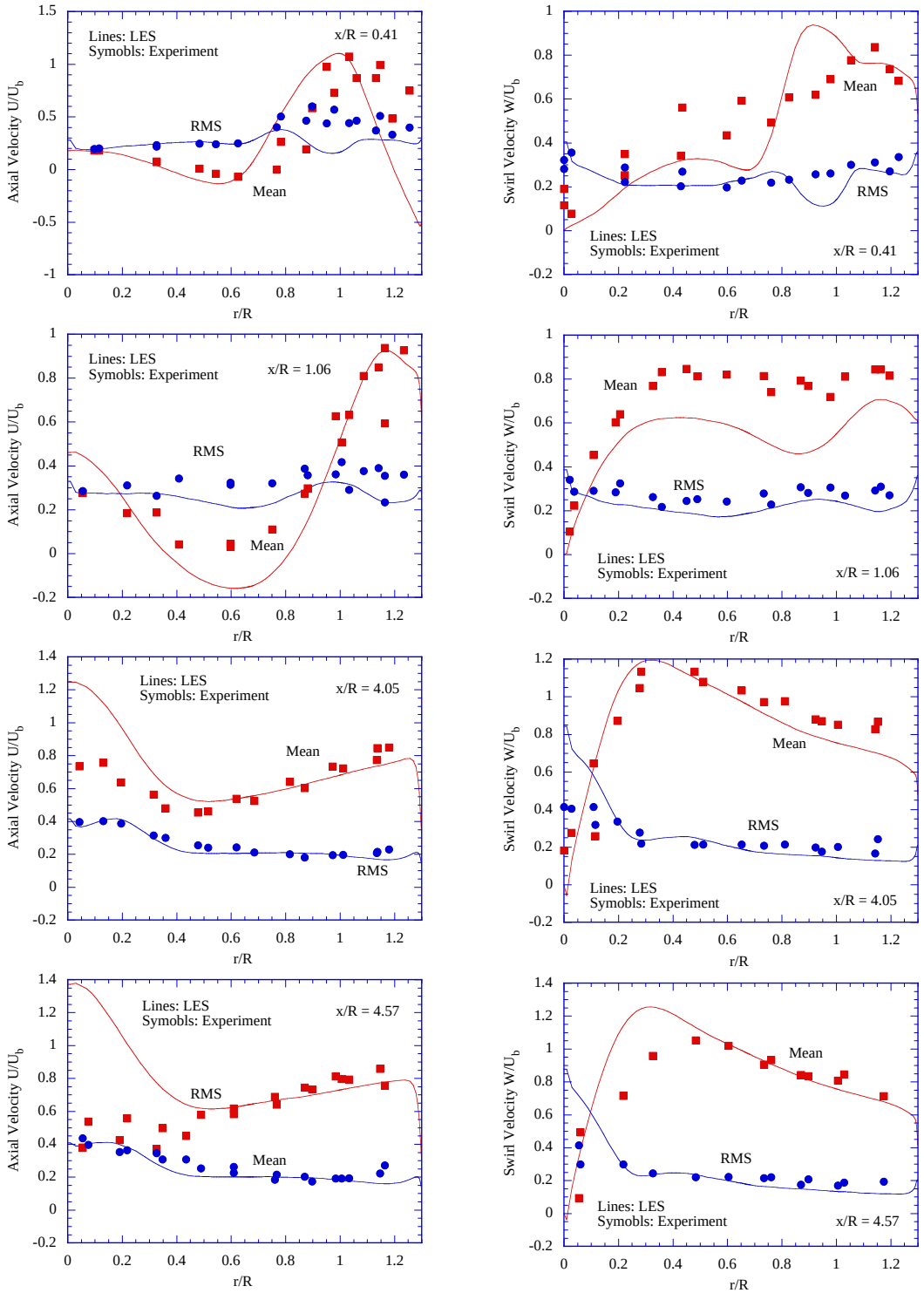


Figure 1.16: Axial (left) and swirl velocity (right) profiles from LES using the base model compared with experimental data at (Top to bottom) $x/R = 0.41,\ 1.06,\ 4.05,\ 4.57$.

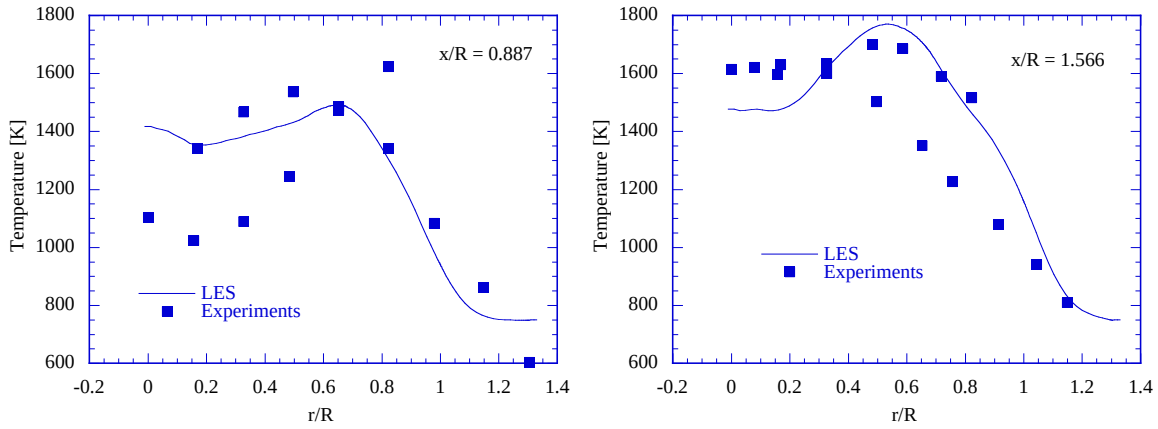


Figure 1.17: Temperature profiles from LES using the base model compared with experimental data at $x/R = 0.887,\ 1.566$.

is overpredicted. However, although slightly weaker, also the experimental data shows an elevated centerline velocity. The same features can be observed in the next downstream position, shown in Fig. 1.16. However, at this position, the overprediction of the centerline is even stronger.

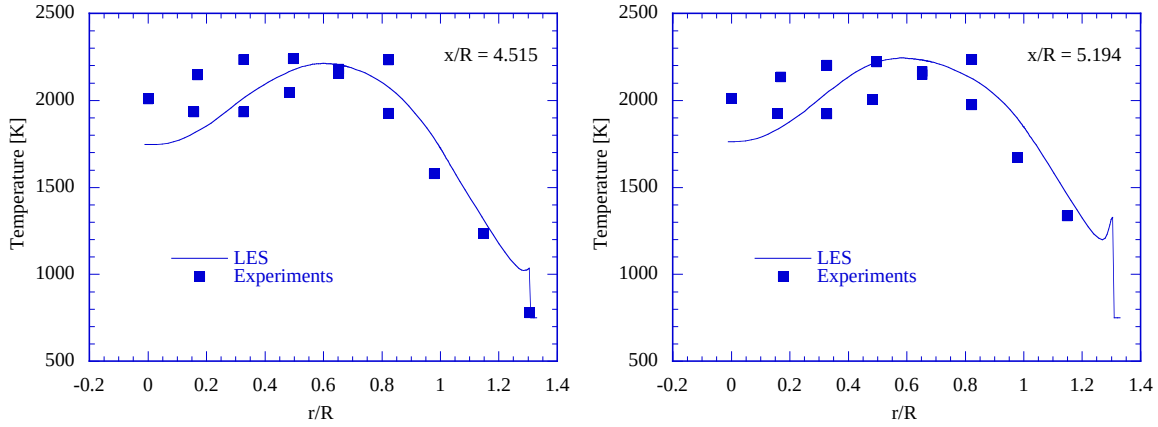


Figure 1.18: Temperature profiles from LES using the base model compared with experimental data at $x/R = 4.515,\ 5.194$.

The temperature profiles are shown in Figs. 1.17 and 1.18. The experimental data show strong scatter. In the display of the measured data, for instance in Fig. 1.17, clearly two profiles can be seen. The reason for this is that the data have been measured throughout the entire cross-section of the combustor. Here, we show the data for both halfs of the combustor. The differences between the two profiles are an indication of the repeatability of the experiment. For all four stations, considering these uncertainties, the data are predicted quite well.

The next set of figures, Figs. 1.19 and 1.20, show the CO_2 mole fraction. Both in the experiments and

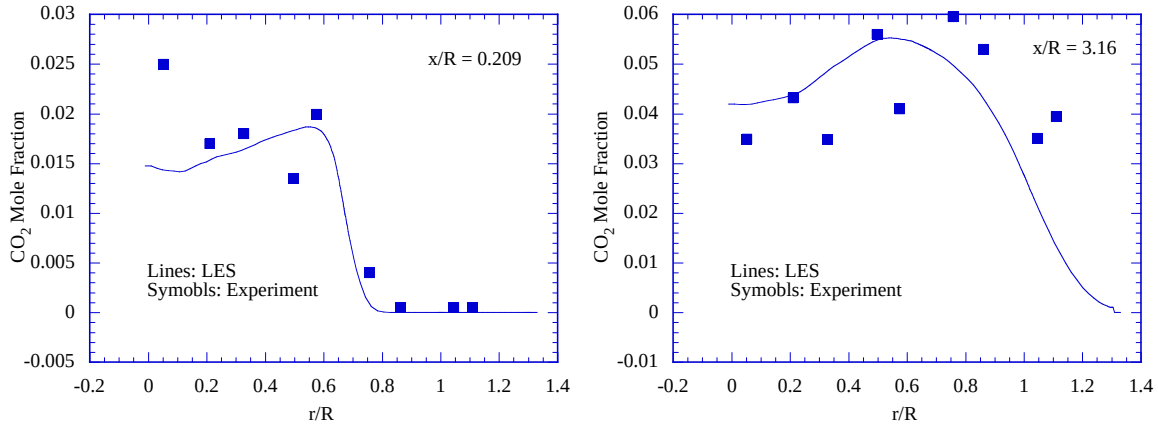


Figure 1.19: CO_2 mole fraction profiles from LES using the base model compared with experimental data at $x/R = 0.209,\ 3.16$.

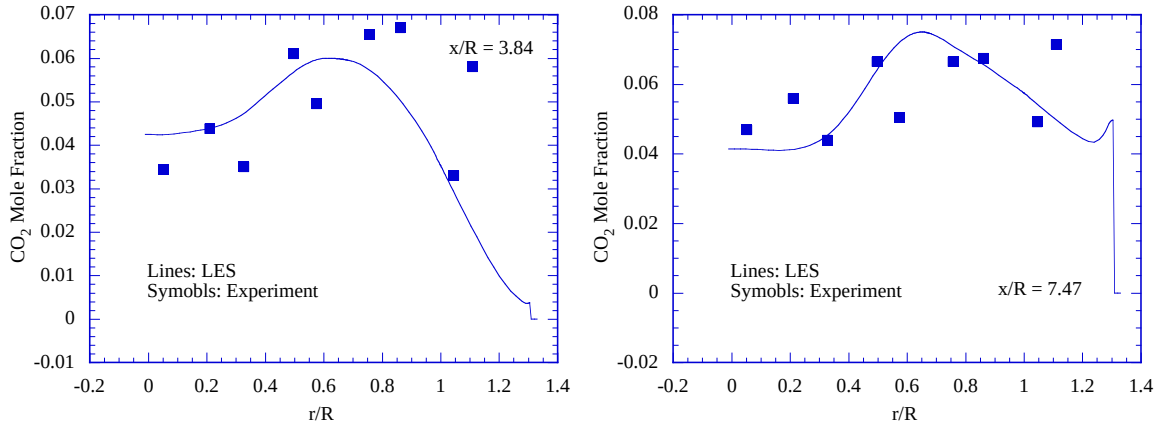


Figure 1.20: CO_2 mole fraction profiles from LES using the base model compared with experimental data at $x/R = 3.84,\ 7.47$.

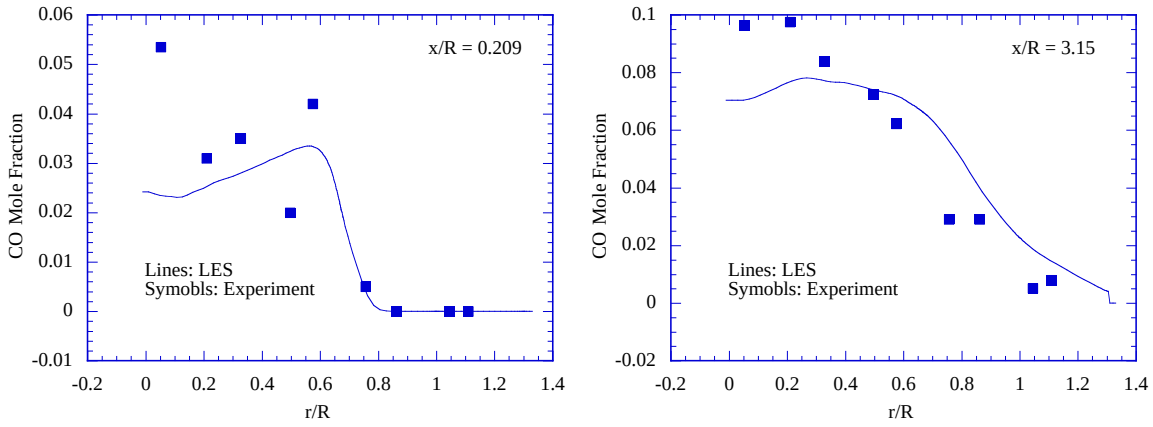


Figure 1.21: CO mole fraction profiles from LES using the base model compared with experimental data at $x/R = 0.209,\ 3.16$.

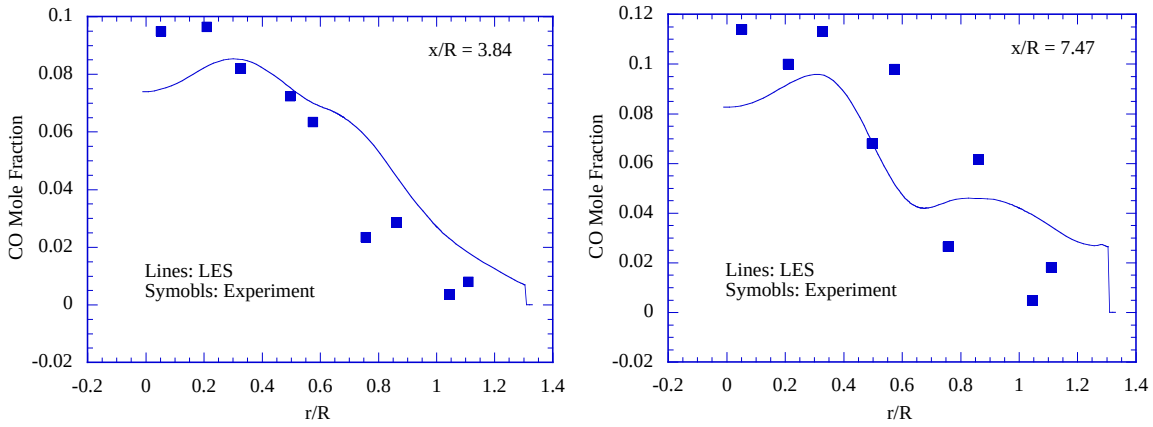


Figure 1.22: CO mole fraction profiles from LES using the base model compared with experimental data at $x/R = 3.84,\ 7.47$.

from the simulation results the mole fractions are evaluated on a dry basis. Again, all profiles are well predicted. Finally, CO is shown in Figs. 1.21 and 1.22. The CO mole fractions are very well predicted considering the fact that in an overall lean environment, this quantity is very sensitive to the flow conditions. The experiments show that the CO mole fraction increases with distance from the nozzle. This trend is captured quite accurately in the simulations, although the opposite trend could be expected, since in an overall lean environment, mixing should decrease the CO mole fraction.

Extensions to the FPV model - Heat Transfer Effects

The accurate prediction of heat interactions in combustion systems is essential to many engineering applications. In gas turbine engines, for example, prolonged heat-stress to components located downstream of the combustor can lead to premature fatigue and limit the operable lifespan of the engine. Reliable predictions of the thermal field within the combustor, therefore, are key factors in the successful design of engines. Thermal conditions within the reactor also strongly influence the formation of pollutants, an issue of increasing importance as regulations on emissions become more stringent. A realistic description of heat-transfer effects is imperative to reliably predict the behavior of these systems. Although global energy balances can be relatively simple to implement on combustion systems, a detailed assessment of local heat fluxes requires information about the immediate structure of the flow and the local composition of the combustion gases. LES is recognized as the natural choice for modeling such complex combustors. A major challenge in LES, however, is to develop computationally efficient models that can realistically describe phenomena that occur on the unresolved scales of the flow. The goal of this study is to develop and validate a chemistry model for use in LES that can describe the complex coupling that occurs between combustion chemistry and heat-transfer effects.

Pierce and Moin [41,] demonstrated the effectiveness of the flamelet/progress-variable method by comparing their LES results with experimental data from the methane-fueled, coaxial-jet combustor of Spadaccini, *et al.* [51]. While the simulations achieved good agreement with the experiment on many mean quantities, there were some notable deficiencies. In particular, the LES showed a slight, but consistent, overprediction in the temperature. This discrepancy was most notable near the outer-wall of the combustor and in well-mixed regions downstream, and likely stems from the neglect of convective heat-transfer effects in the LES. The two-scalar approach of Pierce and Moin indirectly implied adiabatic boundary conditions in the combustor. The experiment, on the other hand, employed water-cooling to maintain a fixed wall-temperature of approximately 500 K. A major motivation of the current study, therefore, is to evaluate how convective heat transfer influences the combustion chemistry and the development of the flow field.

The flamelet/progress-variable method can be extended to describe heat transfer to the combustor confinement by including a filtered energy equation in the LES. In order to fully utilize the additional information provided by this energy equation, an expanded chemistry library must be computed. This is accomplished by expressing the chemical state according to three independent variables: the mixture-fraction, the progress-variable, and the total enthalpy. A transport equation is solved for the filtered total enthalpy, $\widetilde{H}$, and the result is used in the retrieval of chemical information from the expanded library of solutions.

The procedure outlined above has been implemented in a computer code for LES and is currently undergoing validation and refinement. Preliminary results suggest that inclusion of the energy equation leads to a more realistic representation of the temperature field. It is anticipated that this will have far-reaching effects since the temperature is directly coupled to the chemical reactions and indirectly impacts the velocity via the density. Typical snapshots from simulations of a nonreacting coaxial-jet is presented in Fig. 1.23. Figure

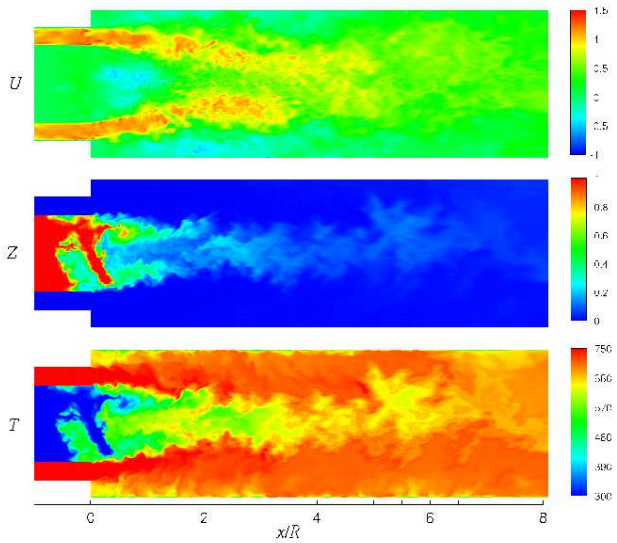


Figure 1.23: Axial velocity, mixture-fraction, and temperature from LES of an nonreacting coaxial-jet using the extended flamelet/progress-variable approach (isothermal wall conditions).

1.23 shows the same fields computed with the extended version of the model using the total enthalpy. It was found that the extended formulation leads to lower temperatures in the near-wall and downstream regions of the combustor. This is an encouraging step toward rectifying the short-comings of the original model mentioned above.

Ongoing work focuses on validating a reacting case computed with the new model and resolving a lingering numerical issue related to the boundary conditions in the enthalpy equation. An alternate formulation that replaces the total enthalpy with temperature, both in the table and in the simulation, is also currently being developed and tested. Future work might account for the effects of gas radiation and soot in the simulations by including a radiative loss term in the energy equation.

1.4.2 Consistent Hybrid LES-FDF scheme for Turbulent Reactive Flows

To describe reacting systems far from extinction, laminar flamelet models and first-order conditional moment closures are sufficient. However, in systems that exhibit local extinction, a higher-order multi-scalar approach is required. The extended FPV approach is one such model that uses two scalars to track combustion chemistry. In a general sense, the transported-filtered density function (FDF) [8,] method provides a natural starting point for such detailed description. The FDF technique has the key advantage that the reaction source term of the scalars appears closed and requires no modeling.

Although a joint velocity-composition FDF transport equation can be formulated, numerical implementations of this high-dimensional system pose stability and feasibility issues. To overcome this problem, a hybrid approach is used where the velocity and turbulence fields are solved using an Eulerian scheme (like RANS or LES) while the scalar transport is handled using the FDF approach. Although the FDF technique has been widely used in the RANS context, almost all the applications involve steady-state flows. Since LES technique is inherently transient, the coupled LES-PDF method needs to maintain temporal accuracy. Due to the statistical nature of the FDF scheme, such a coupled scheme poses numerical accuracy issues.

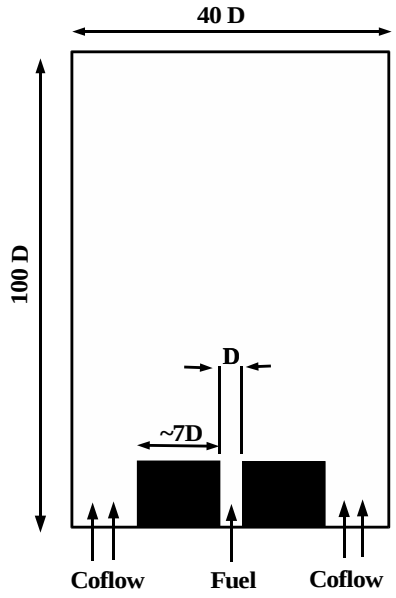


Figure 1.24: Schematic of the bluff-body flame configuration. The jet diameter D is 3.6 mm.

During this year we have developed a stable, robust and accurate formulation of the hybrid approach.

Hybrid LES-FDF Scheme

In the hybrid scheme implemented here, the LES technique is based on a low-Mach number approximation-based finite-volume scheme. Further details of the LES implementation can be found elsewhere [41,]. The Lagrangian method uses stochastic particles to evolve the FDF.

The particle method is obtained from the fundamental FDF transport equation. The FDF in a variable density flow can be defined as:

$$F_L(\boldsymbol{\psi};\mathbf{x},t) = \int_{-\infty}^{+\infty} \rho(\mathbf{y},t)\,\xi[\boldsymbol{\psi},\boldsymbol{\phi}(\mathbf{y},t)]\,G(\mathbf{y}-\mathbf{x})\,d\mathbf{y}, \tag{1.1}$$

$$\xi[\boldsymbol{\psi},\boldsymbol{\phi}(\mathbf{y},t)] = \delta[\boldsymbol{\psi}-\boldsymbol{\phi}(\mathbf{y},t)], \tag{1.2}$$

where δ is a N-dimensional delta function for a N-species system and $\boldsymbol{\psi}$ is the random variable in the composition domain. The FDF definition yields the following property:

$$\int_{-\infty}^{+\infty} F_L d\boldsymbol{\psi} = \int_{-\infty}^{+\infty} \rho(\mathbf{y},t)\,G(\mathbf{y}-\mathbf{x})\,d\mathbf{y} = \overline{\rho}. \tag{1.3}$$

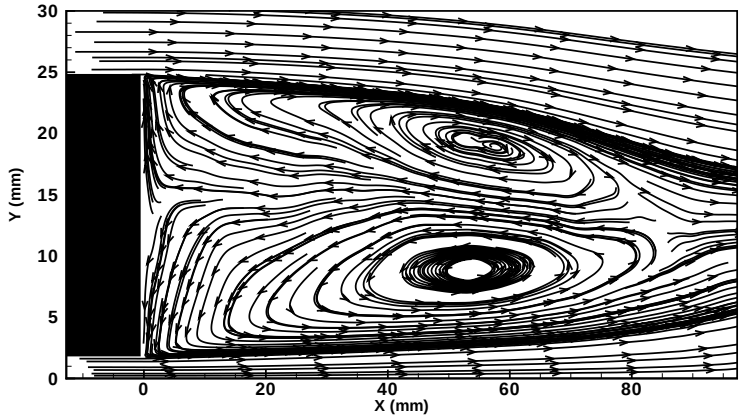


Figure 1.25: Streamtraces of the time-averaged velocity vector showing the counter-rotating vortices.

Similarly, the filtered mean of any scalar Q_ϕ can be defined as:

$$\widetilde{Q_\phi} = \int_{-\infty}^{+\infty} Q_\phi(\boldsymbol{\psi}, \mathbf{y}, t) F_L d\boldsymbol{\psi} = \frac{1}{\overline{\rho}} \int_{-\infty}^{+\infty} \rho\left(\mathbf{y}, t\right) Q_\phi\left(\mathbf{y}, t\right) G\left(\mathbf{y} - \mathbf{x}\right) d\mathbf{y}. \tag{1.4}$$

Using these definitions, the transport equation for the joint composition FDF can be written as [8, 20,]

$$\frac{\partial F_L}{\partial t} + \frac{\partial}{\partial \mathbf{x}}\left(\widetilde{\mathbf{u}} F_L\right) + \frac{\partial}{\partial \mathbf{x}}\left(\widetilde{\mathbf{u}'|\boldsymbol{\psi}} F_L\right) = -\frac{\partial}{\partial \boldsymbol{\psi}}\left[\left(\frac{1}{\overline{\rho}}\overline{\nabla \cdot \rho D \nabla \boldsymbol{\phi}|\boldsymbol{\psi}} + \mathbf{S}(\boldsymbol{\psi})\right) F_L\right], \tag{1.5}$$

where $\widetilde{\mathbf{u}}$ is the filtered velocity field, $\widetilde{\mathbf{u}'|\boldsymbol{\psi}}$ is the sub-filter velocity fluctuation conditioned on the scalar, $\overline{\nabla \cdot \rho D \nabla \boldsymbol{\phi}|\boldsymbol{\psi}}$ is the conditional micromixing term, and $\mathbf{S}$ is the reaction source term. The conditional velocity term is modeled using the gradient-diffusion hypothesis to give

$$\widetilde{u_i'|\boldsymbol{\psi}} F_L = -\overline{\rho} D_T \frac{\partial F_L/\rho}{\partial x_i}. \tag{1.6}$$

The conditional mixing term is closed using the Interaction-by-Exchange-with-the-Mean (IEM) model [56,].

$$\overline{\nabla \cdot \rho D \nabla \boldsymbol{\phi}|\boldsymbol{\psi}} = \nabla \cdot \overline{\rho} D \nabla \widetilde{\boldsymbol{\phi}} - \frac{\overline{\rho} C_\phi}{\tau}\left(\boldsymbol{\psi} - \widetilde{\boldsymbol{\phi}}\right), \tag{1.7}$$

where C_ϕ is scalar-to-mechanical time-scale ratio and τ is a turbulence time scale. In the present study we set C_ϕ to be 2 [35,] and use a turbulent-diffusivity-based time scale [8,].

The high dimensionality of the FDF equation makes finite-differencing-based solution techniques infeasible. A stochastic approach [43,] is used where the filtered momentum equations are solved using conventional grid-based techniques (like LES) while the FDF equation is solved using a particle-based Monte-Carlo approach. The Lagrangian system uses the filtered fields from the LES solver to advance the notional particles. Using the particle properties, mean fields are constructed that is fed back to the LES solver. The LES solver then advances the flow using these mean fields. Typically, the Lagrangian system provides the filtered density field that is then used by the LES solver.

To increase statistical accuracy, large particle numbers are needed making such methods computationally expensive. Numerical implementation of the LES-FDF scheme is an algorithmic challenge and novel techniques are used to reduce the computational expense of these schemes [44,]. A major issue in such implementations is to consistently couple a stochastic FDF scheme and a deterministic Eulerian scheme. Although steady-state-based flow solvers have been successfully used [29,], a consistent algorithm for a temporally variant system (like the LES-based approach) has not been studied in detail so far. Here we propose a criterion for a consistent implementation and test it with a challenging reacting flow problem.

Consistency Requirements

In the Lagrangian particle-based system, the computational domain is decomposed into a large number of notional particles that represent the fluid. The particles are initially distributed uniformly and evolve in space and time using stochastic differential equations [43,]. Each particle carries information about its location, a composition vector and a representative weight. In order to pass information from and to the LES solver, particle mean fields are obtained by a weighted summation process involving particle properties in a given computational cell. The particle weight is initially assigned to be the local fluid mass such that the sum of the particle weights in a computational cell equals the cell fluid mass, hence

$$w_k = \frac{V_i \overline{\rho}_i}{N_p}, \tag{1.8}$$

where w_k is the particle weight, V_i is the cell volume, $\overline{\rho}_i$ is the fluid density in cell i, and N_p is the number of particles in the cell. At any time step, the particle-weights based density can be obtained by using the sum of particle weights in a given cell.

$$\overline{\rho}_w = \frac{1}{V_i} \sum_{i=1}^{N_p} w_k. \tag{1.9}$$

In addition to the particle weight, a mean density can also be obtained from the particle composition vector based on thermochemical properties.

$$\overline{\rho}_p = \frac{\sum_{k=1}^{N_p} w_k / \rho\left(\boldsymbol{\phi}_k\right)}{\sum_{k=1}^{N_p} w_k}, \tag{1.10}$$

where $\rho(\boldsymbol{\phi})$ is the thermochemical density computed using the particle composition. The initial conditions are chosen such that $\overline{\rho} = \overline{\rho}_p$ at $t = 0$. As the particles evolve in space and time, the density fields evolve through different equations, though indirectly, they should all satisfy the continuity equation. By construction [44,], the thermochemical density ($\overline{\rho}_p$) and the LES density ($\overline{\rho}$) evolve closely. This is ensured by solving an ancillary enthalpy transport equation using Eulerian schemes. The source term for the enthalpy equation is provided using the particle properties. The Eulerian density is obtained directly using this enthalpy field [29,].

On the other hand, the particle weights-based density ($\overline{\rho}_w$) evolves with the particle motion. Based on the continuity equation, it can be shown [43,] that for the particles to be uniformly distributed, the particle-weights-based density should be equivalent to the Eulerian density field. However, the stochastic evolution scheme will make the particle-based mean fields noisy. Hence a strict equality can be obtained only by time-averaging a statistically stationary field. Finite particle number density can also introduce bias in the

mean fields. This will lead to a progressive divergence of the particle-weights-based density field from the Eulerian field. Such a bias will be readily observed through particle agglomeration in certain sections of the grid and depletion of particles in other regions. The sampling error induced by such low particle number density will further increase the error in mean-field estimation.

A consistent algorithm should hence maintain the equivalence of the three density fields described above. It is noted that this amounts to a consistent evolution of the zeroth moment of the FDF-transport equation. The accuracy of the scheme can be tested using higher-order moments of the same equation. For the single-scalar flamelet model used here, the first moment of the scalar can be evolved simultaneously by both the particle and Eulerian systems. Such a moments-based validation procedure ensures a robust yet simple way of demonstrating the accuracy of the numerical implementation. We illustrate the validation criteria using an experimental flame next.

Numerical Test - Application to a Bluff-body Stabilized Flame

The non-premixed methane-hydrogen bluff-body stabilized flame [9,] is used to test the hybrid LES-FDF approach. The schematic of the flow configuration and the computational domain is provided in Fig. 1.24. The fuel is a methane-hydrogen mixture with 1:1 volume ratio entering through a pipe with diameter, D, of 3.6 mm. The coflow is air and enters through an annulus separated from the fuel by a solid bluff-body with a diameter of 50 mm. Two different experimental configurations exist: the first termed HM1 used to measure the velocity fields, the second experiment termed HM1E was used for scalar measurement. Both flames have the same blowout characteristics, but the jet velocities are slightly different. HM1 was conducted with the fuel jet issuing at 108 m/s and coflow at 35 m/s. HM1E uses a fuel jet velocity of 118 m/s and coflow of 40 m/s. Here both configurations are simulated and compared with the corresponding experimental data.

In this study, a $256 \times 128 \times 32$ grid that spans 100 D in axial direction and 40 D in radial direction is used. The very low stoichiometric mixture fraction (≈ 0.055) necessitates high resolution in the outer shear-layer in order to resolve the steep gradients in density. Similarly, the inner shear layer contains the strongest gradients in velocity, and hence requires a finer grid to resolve the large scales. Consequently, the grid is heavily clustered in the inner shear layer and the outer vortex region leading to large non-uniformities in particle weights. The primary reaction zone, where the flame reverts to a normal jet flame is located at around X = 65 mm. Here again, the resolution of the grid is important. Although these observations are based on experimental data, no continuous adaptation of the grid is performed.

Due to the low extinction probability, the laminar flamelet model should adequately model combustion characteristics. Here, the flamelet table is constructed using the FlameMaster code [22,] with the mixture fraction and the mean scalar-dissipation rate as parameters. The GRI-2.11 chemistry mechanism [3,] is used with the steady-flamelet equations [35,].

In the present study, a mixture-fraction-based flamelet model is used for describing the chemistry. For the Lagrangian solver, a single scalar, namely mixture-fraction, is evolved. For the sake of comparison, the mixture-fraction equation is also solved using an Eulerian scheme. To ensure that the second moment of the mixture-fraction evaluated from the FDF scheme is accurate, an equivalent transport equation for the second moment is also solved in the Eulerian context. All the scalars use appropriate inflow conditions based on the species composition at the inlet plane. The outflow conditions are purely convective for both the momentum and scalar equations. Further details of the LES solver are detailed elsewhere [41,].

The FDF solver was initialized with 15 particles per cell. During the course of the simulation, the total number of particles in the domain was in the range of 10.5–21 million. The simulations were started from

cold-flow-converged results after which mixture was ignited using the flamelet solution. All simulations were continued for 7 flow-through times where each flow through time is defined by the time taken for a particle traveling along the centerline to move from inflow to exit. The simulation is time-averaged for 1.5 flow-through times starting at two different time-steps separated by 1 flow-through time. The two time-averaged profiles differed by less than 2 % for the mixture-fraction radial profiles ensuring that statistical stationarity has been reached. The LES-FDF simulation took roughly 200 hours on an 8-processor 600 MHz computer to reach statistical stationarity.

The methane-hydrogen flame is stabilized by the two counter-rotating vortices seen in Fig. 1.25. The primary reaction zone is located at the end of the vortices, where the jet-like structure is recovered. The inner vortex helps to entrain the fuel into the recirculation zone. The outer vortex then pre-heats the coflow that comes in contact with the fuel in the center of the recirculation zone.

The consistency of the particle scheme can be assessed by comparing the density implied by the particle weights $\overline{\rho}_p$ and the density obtained from the particle compositions using the flamelet table, $\overline{\rho}$. The Eulerian density field corresponding to the continuity equation should be identical to $\overline{\rho}_p$ as implied by the FDF evolution equations. Figure 1.26 shows the comparison of the three density fields at different axial positions. The Eulerian field and the particle-weights-based field are identical since the particle evolution is forced by the Eulerian density field. This serves to show that the numerical implementation is accurate. The comparison with the particle-composition-based density shows similar excellent agreement at all axial locations. This confirms that the feedback loop is accurate and that, inspite of the differences in the numerical algorithms for physical transport, both the Eulerian and Lagrangian density fields agree with negligible differences.

Figure 1.27 shows the time-averaged mixture-fraction and RMS mixture-fraction radial profiles at different axial locations. To aid in the comparison, the Eulerian mixture-fraction and RMS mixture-fraction fields obtained from the finite-volume solution of the scalar transport equation are also included. It is observed that the mean mixture-fraction profiles obtained from the FDF solver and the Eulerian solver show excellent agreement with one another and the experimental data as well. The near-bluff-body profile shows a flat profile in the recirculation zone, further affirming the large-scale mixing in this region. The sharp decay of the mixture fraction near the edge of the bluff body is a region of large temperature changes, and consequently large density gradients. At $X = 30\,\text{mm}$, the recirculation region is slightly overpredicted, indicated by the large mixture-fraction values as compared to experiments as well as a sharper decay at the outer vortex signifying a thin reaction zone. Further downstream the profiles are in much better agreement.

The RMS profiles show good agreement with experimental data although certain discrepancies are noticed near the centerline. In general, it is observed that the results agree well with the Eulerian computation as well. At the first axial position considered, the FDF as well as Eulerian calculation show the right RMS profile indicating that the large-scale recirculation has been captured accurately. It is noted that the sub-filter or the unresolved variance in this zone is very small since the large-scale mixing renders the fluid homogeneous. Similar trends are observed $X = 30, 45\,\text{mm}$, but the extent of the recirculation zone in the radial direction decreases as implied by the streamtrace profile (Fig 1.25).

Figure 1.28 shows the radial profiles of temperature and CO_2 mass fraction. From the plots, it is evident that both scalars behave similarly in this flame. Similar to the mixture-fraction profiles, the mean temperature and species mass fraction exhibit close agreement with experimental data in regions close to the bluff body. At downstream locations the peak in the temperature is underpredicted. These regions correspond to the outer shear layer and regions of maximum reaction rate. At $X = 45\,\text{mm}$, the mean and RMS mixture fraction are accurately predicted in the outer shear layer. Hence, the discrepancy can be attributed to the sub-filter mixing model.

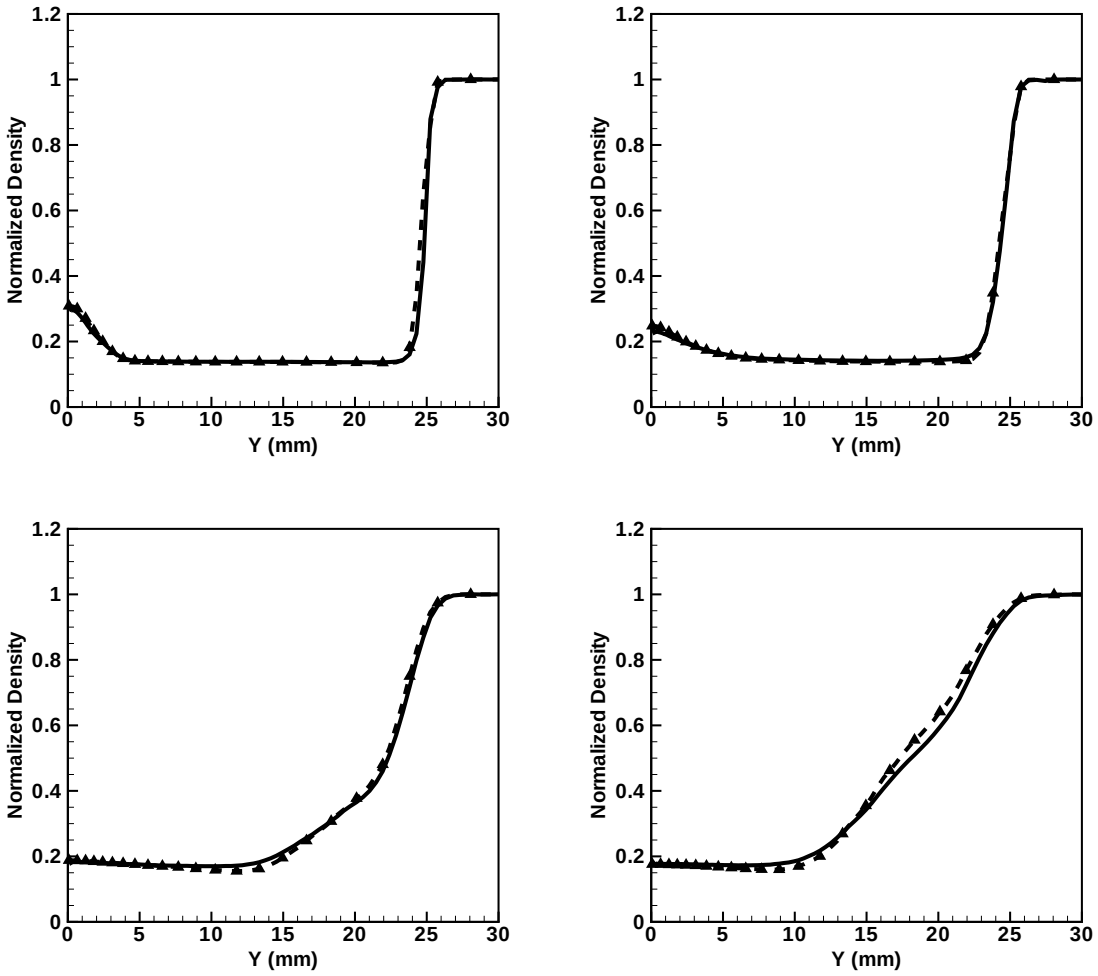


Figure 1.26: Comparison of density obtained from particle weights (dashed line), particle composition (solid line) and LES flow solver (symbols). The four plots are at downstream locations of (top) 13, 30, (bottom) 65, and 90 mm, respectively. The density values have been normalized by the density of the coflow.

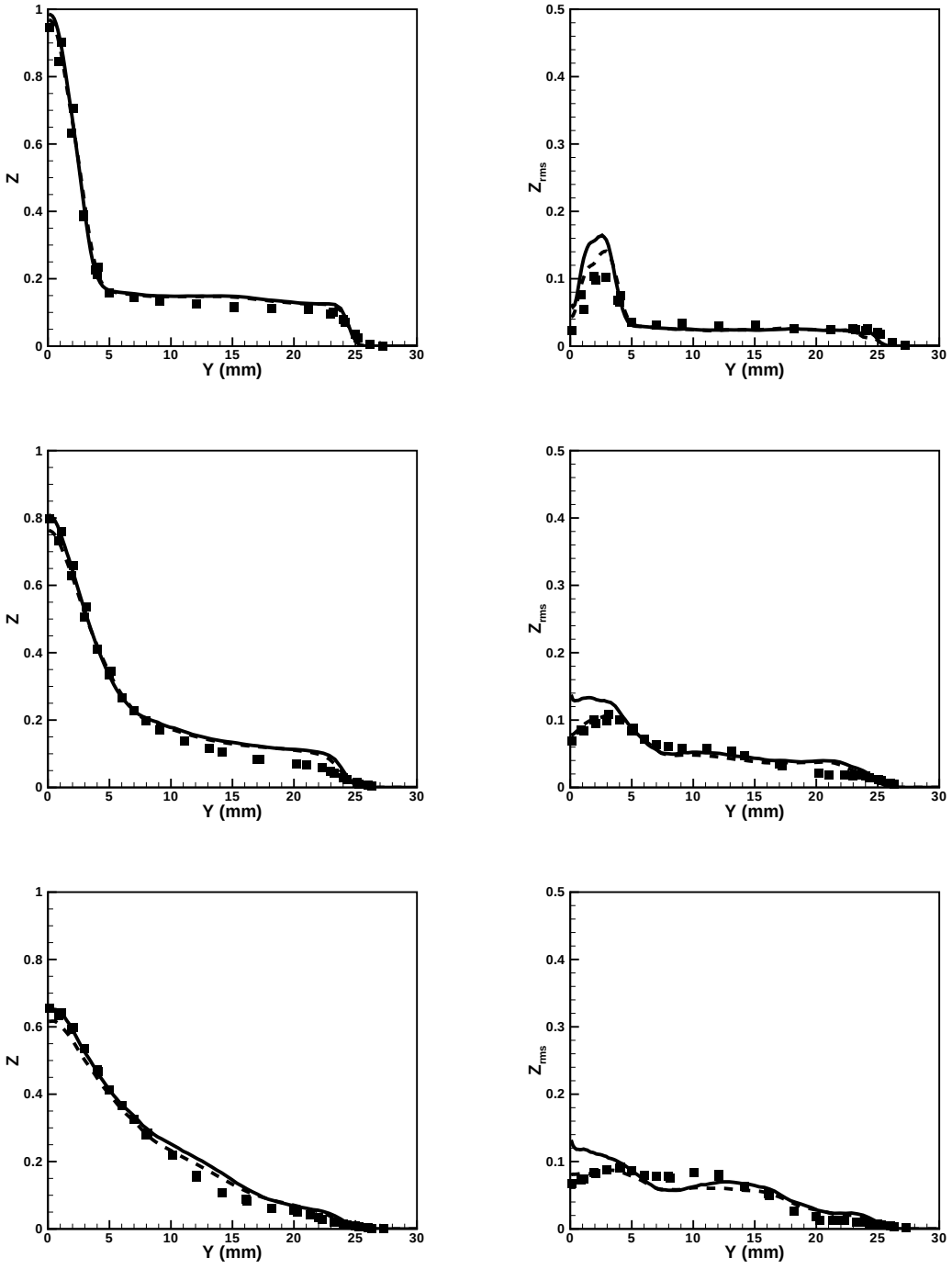


Figure 1.27: Comparison of mean mixture fraction and RMS mixture fraction profiles with experimental data at different axial locations. From top to bottom, $X = 13$, 30, and 45 mm. Symbols are experimental data, solid lines show FDF-based results and dashed lines show beta-function-based Eulerian calculation.

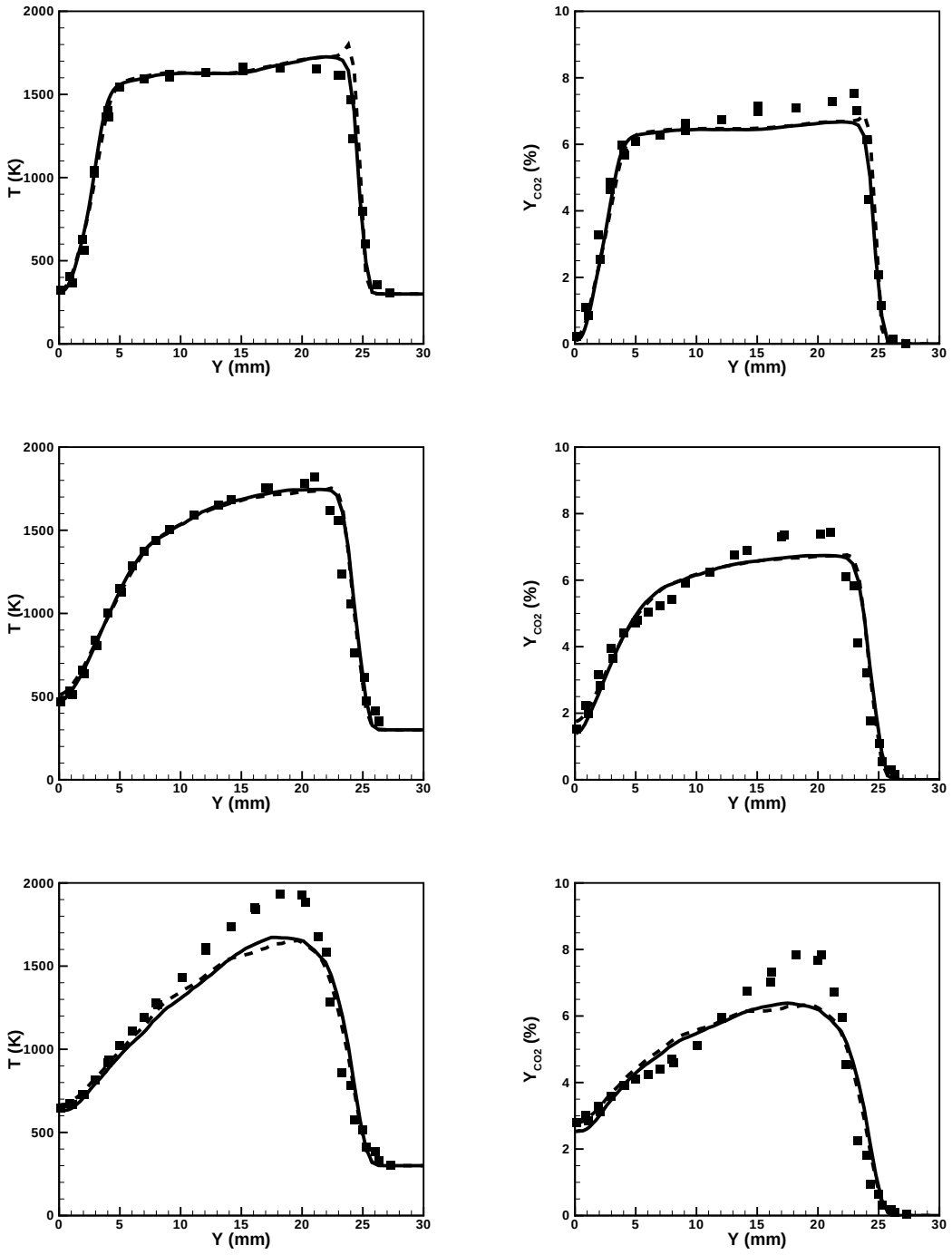


Figure 1.28: Comparison of mean temperature and CO_2 mass-fraction profiles with experimental data at different axial locations. From top to bottom, $X = 13$, 30, and 45 mm. Symbols are experimental data, solid lines show FDF-based results and dashed lines show beta-function-based Eulerian calculation.

Figure 1.29 shows the radial profiles of CO and OH mass fractions. Considering the simplicity of the chemistry model, the CO profiles show good agreement with experimental data. Prediction of OH profiles in any combustion simulation is particularly challenging due to the strong non-linearity of the species evolution. At $X = 13$ mm, the production is limited to the outer shear layer. The FDF scheme predicts this peak quite accurately but the Eulerian scheme shows a pronounced peak. Looking at the temperature profile at this location (Fig. 1.28), a small spike is noticed at the outer edge of the bluff body. This clearly results in the overprediction of the hydroxyl radical. This inaccuracy is a result of the numerical scheme used, as it is well known that the QUICK discretization scheme contains some spurious oscillations in regions of large scalar gradients [41,]. Typically, central-difference-based numerical discretizatiosn schemes will contain some level of oscillations. At $X = 30, 45$ mm, the FDF as well as Eulerian predictions agree well with experiments. The reaction zone also widens compared to $X = 13$ mm. Similar trends are observed at $X = 65, 90$ mm.

In conclusion, a consistent stable algorithm for hybrid LES/FDF simulations has been devised and implemented. The low-Mach-number-approximation-based LES solver was coupled to the notional-particle-based Lagrangian FDF solver using a time-accurate coupling scheme. The particle scheme was implemented using scalable algorithms with distributed-memory-based parallelism. Linear scale-up was obtained up to 64 processors. Particle motion in physical space was implemented using a cell-faced tracking strategy. In composition space, particles were moved through mixing and reaction. A simple IEM model with a turbulent-diffusivity-based time scale was used for describing mixing. Since a mixture-fraction-based flamelet approach is used, the reaction source term was set to zero.

The coupling algorithm between the solvers is non-trivial in the LES/FDF approach as compared to a RANS-PDF approach due to the time-dependent evolution of the flow. The particle velocities are updated using velocity fields from the LES solver at every time step. The chemical reaction information is supplied to the flow field indirectly through density changes. Since the particle solver uses a statistical method, direct feedback of particle mean fields contain stochastic noise which can lead to numerical instabilities. To overcome this issue, an equivalent enthalpy equation is used with the particle scheme supplying the exact source term for the Eulerian transport equation of this quantity. This feedback algorithm is found to ensure numerical stability.

1.4.3 Modeling Soot Formation In Gas Turbine Engines

Pollutant formation has become a great concern because of public health and environmental consequences. In the near future, additional regulations will limit soot emission levels and require knowledge of the soot particle size distribution. The present work intends to develop an accurate model to describe formation of soot particles, which can be applied to technical devices such as gas turbine engines. The model involves detailed chemistry and a comprehensive description of the interaction of soot particles dynamics and chemistry.

Soot model

Soot particles are formed from aromatic gas phase species and grow by both chemical and physical processes (Fig. 1.30). The chemical processes consist of molecular reactions on the surface of the particles. A HACA (H-Abstraction, C_2H_2-Addition) mechanism is used for the growth of soot particles while oxidation extracts mass from the particles back into the gas phase. Soot particles are also affected by various physi-

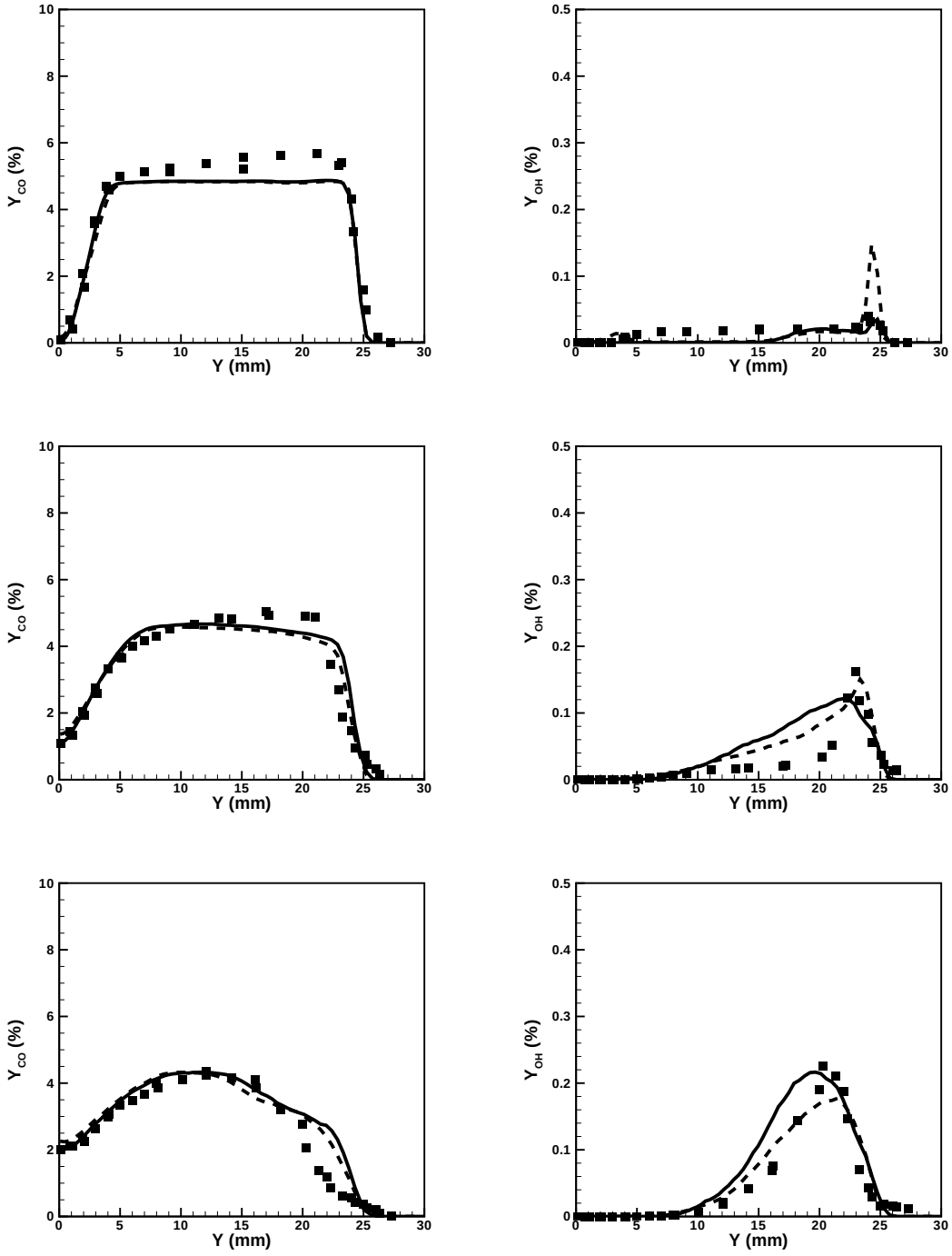


Figure 1.29: Comparison of mean CO and OH mass-fraction profiles with experimental data at different axial locations. From top to bottom, $X = 13$, 30, and 45 mm. Symbols are experimental data, solid lines show FDF-based results and dashed lines show beta-function-based Eulerian calculation.

cal processes, namely, nucleation (formation of one particle from two Polyaromatic Hydrocarbons (PAH)), coagulation (fusion of two soot particles by collision), and condensation (addition of mass by collision with aromatic molecules).

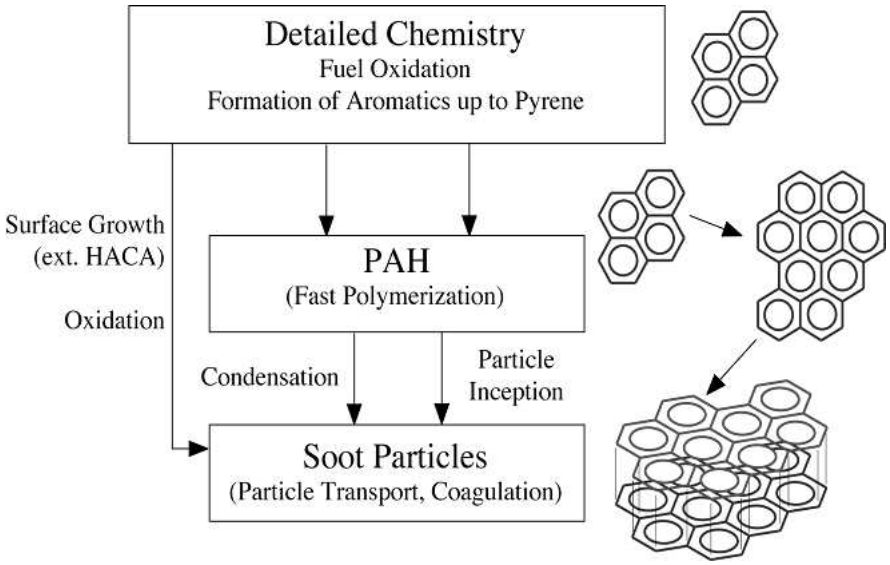


Figure 1.30: Chemical and physical process of soot formation

The current model uses the method of moments introduced by Frenklach et al. [11, 26]. A statistical representation of soot particles is derived from the Smoluchowski equation that represents the effects of the different chemical and physical processes on the particle size distribution. Two transport equations are solved for soot: number density of soot particles and soot volume fraction corresponding to the first two moments of the particle size distribution. Mean soot properties (like mean diameter) can easily be deduced from these two moments.

Validation of soot chemistry and model

The chemical mechanism used in the current model is based on the work of Frenklach and Mauss [11, 26]. This mechanism uses kinetic reactions from Ref [26] which was originally developed for ethylene, but is modified for the current configuration. Based on validation using laminar premixed and diffusion flames, the rates of the several reaction steps were optimized. The current mechanism is composed of 102 species and 1045 reactions.

The quality of the soot prediction is highly dependent on the underlying chemistry mechanism. Therefore, a careful validation process has to be used to improve soot chemistry. The chemical mechanism and the soot model are currently being tested against a database that contains a wide range of laminar flames. These flames include premixed and diffusion flames with varying importance of the different chemical and physical soot processes. The validation is also performed using different fuels, which has a strong influence on the resultant soot concentration (CH_4, C_2H_2, C_2H_4, C_2H_6, C_3H_8, C_6H_6). The rate of the kinetic reactions are optimized in order to match the temperature, main species and soot volume fraction profiles.

Figure 1.31 shows a typical configuration from the database corresponding to a counterflow diffusion

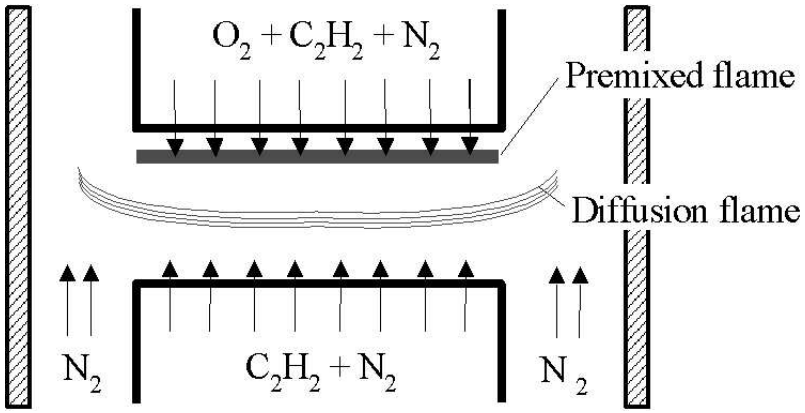


Figure 1.31: Configuration of a typical counterflow diffusion flame.

flame [34]. Through a coordinate transformation, the axisymmetric problem can be reduced to a $1D$ problem. The simulation results for this configuration are shown on Fig. 1.32. As it can be observed the temperature and the species concentration are accurately represented. Soot volume fraction and particle diameter are also in good agreement with the experiments.

Figure 1.33 shows the results for a typical ethylene premixed flame. Once again the soot volume fraction prediction is in good agreement with the experimental data. Furthermore the temperature dependance of soot volume fraction is predicted quite accurately. It is the first time that a chemical mechanism is optimized for both diffusion and premixed flames and is shown to have very good agreement with experimental data.

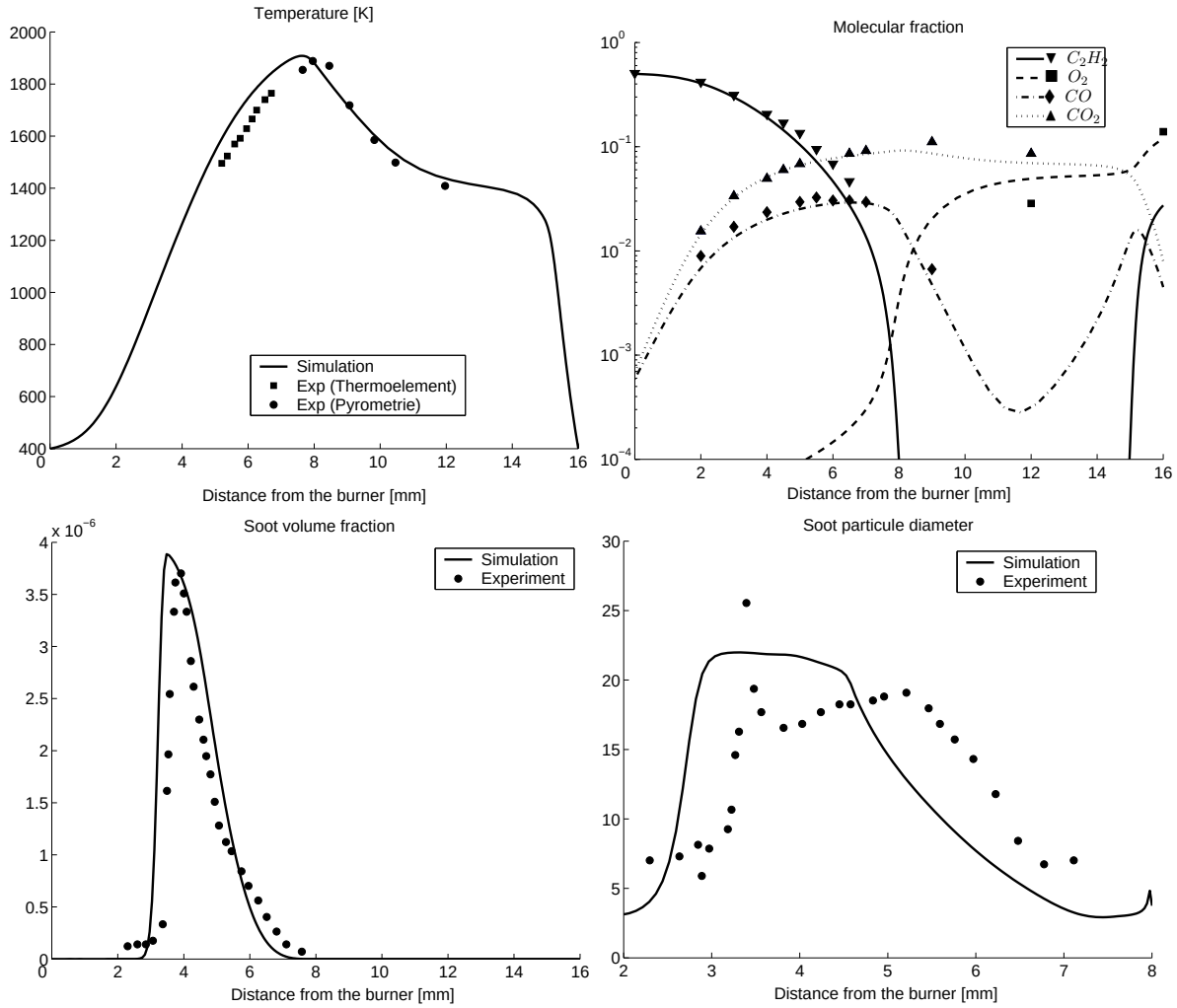


Figure 1.32: Results of the simulation of a counterflow diffusion flame. Temperature (upper left), species mole fraction (upper right), soot volume fraction (lower left), soot particle diameter (lower right).

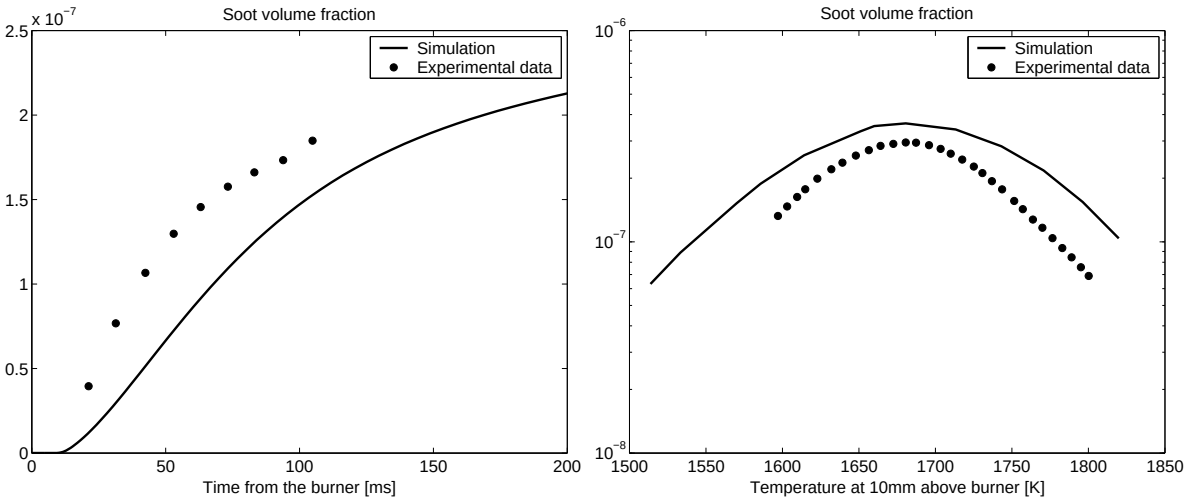


Figure 1.33: Results of the simulation of an ethylene premixed flame. Time evolution of soot volume fraction (left), temperature dependence of soot volume fraction (right)

1.5 Advanced Spray Models in CDP

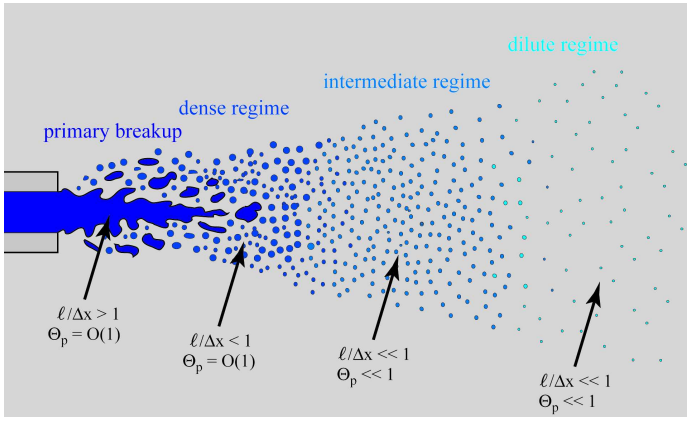


Figure 1.34: Regimes of liquid spray in gas-turbine combustors.

The objective of the spray simulation effort is to develop a numerical framework based on Lagrangian Particle Tracking (LPT) models to perform high-fidelity large-eddy simulation (LES) of reacting multiphase flows encountered in realistic gas-turbine combustion chambers. Three regimes of flow development are commonly observed inside these combustors as shown in Fig. 1.34: 1) 'dense regime' where the liquid film interacts with gas-phase and disintegrates into filaments by forming Kelvin-Helmholtz type instability, 2) 'intermediate regime' where the liquid blobs formed undergo secondary breakup, and 3) 'dilute regime' where the droplets evaporate, the fuel vapor mixes with the surrounding hot gas giving turbulent spray flames. Droplet deformation and collision are also important features in the intermediate regime. In CDP, the dispersed phase is treated as 'point-particles' with models for inter-phase mass, momentum, and energy transport as well as stochastic models for liquid drop breakup. This formulation was validated by Apte et al. [24, 1] to show good predictions for both phases in the dilute regime. The main task in FY04 was to develop and implement advanced models for particle dispersion, dense spray regime, and liquid film formation commonly encountered in advanced gas-turbine injectors. The following issues were addressed:

- *Effect of subgrid scale fluctuations on particle dispersion:* In standard LPT based methods, the drag on a particle (or droplet) depends on the particle velocity, the gas-phase velocity at the particle location, and the particle relaxation time. However, in LES the gas-phase velocity only contains the resolved scale motions. The effect of subgrid scale motions on particle drag and lift become important especially when the unresolved kinetic energy content is large. This effect is accounted for by developing a model based on Filtered Particle Tracking (FPT) techniques.

- *Models for dense spray regime:* In simulations of realistic gas-turbine combustors, the droplets (parcels) introduced near the injector wall are comparable to the grid size and the local droplet concentrations are high. The volume displaced by these droplets is significant and the standard point-particle approach is invalid in the dense-spray regime. The point-particle formulation is extended to account for the finite-size effects of the dispersed phase. The model and its implementation is verified by simulating gravitational settling of solid particles and fluidization by air jet in a vertical channel. In

addition, it is shown that the point-particle approach fails in predicting lift of closely packed particles in a Poisuille flow whereas the dense-spray model captures this important effect.

- *Modeling liquid film formation:* In advanced gas-turbine combustor designs, the liquid fuel is injected in the form of a sheet which impinges upon the injector walls and forms an intermittent liquid film. Accurate prediction of the growth and dynamics of the liquid film plays an important role in capturing breakup and drop-size distribution in combustion chamber. The effectiveness of the dense-spray model in capturing these features is evaluated. In a parallel effort, a model based on lubrication theory involving solution of a transport equation for film-height was developed and implemented in CDP.

A subgrid scale model accounting for the effects of unresolved gas-phase velocities onto the particle motion is described. This model is tested in simple homogeneous isotropic flows with particles. Next, a brief summary of Eulerian/Lagrangian equations for dense spray is given. Effect of finite-size of the dispersed phase is elucidated by simulating fluidization of solid spheres in a Poisuille flow. A model for liquid film formation and growth based on a equation for film height is described. The dense spray and liquid film models are applied to simulate impingement of a liquid spray on a flat plate, depicting liquid film formation on the walls of advanced injector designs.

1.5.1 Modeling Droplet Dispersion in LES of Two-Phase Flows

In LES of practical combustor applications, the content of sub-grid scale kinetic energy could be large in regions of poor grid resolution. These unresolved motions for the carrier phase are typically modeled using a dynamic Smagorinsky model. The equations of the dispersed phase velocity field, requires an instantaneous *undisturbed* carrier phase velocity at the particle location. With two-way coupling between the two-phases, the resolved velocity field of the carrier phase is modified due to the presence of the dispersed phase. In addition, the subgrid scale velocity field may influence the statistics of the dispersed phase and alter the spray evolution significantly. We capture some of these effects by developing a stochastic model based on Filtered Particle Tracking (FPT) which statistically reconstructs the unresolved carrier-phase velocity along particle trajectories. This model is tested on forced isotropic turbulent flow and results are compared against DNS studies.

A reasonable assumption for LES is to consider the residual turbulent motion as locally homogeneous and isotropic. Then, the fluid velocity *seen* by particles is computed as $U_i^* = \tilde{U}_i(\mathbf{x}_p, t) + u_i^*$, i.e. the sum of the filtered LES velocity $\tilde{U}_i$ interpolated at the particle location and the residual velocity *seen* by the particle. Crucial ingredients of an FPT model are the *sgs* turbulent kinetic energy of the fluid and a *sgs* time scale. By analogy to modeling turbulent diffusion of fluid elements [46], we assume that $\mathbf{u}^*$ is governed by the Langevin equation

$$du_i^* = -\frac{u_i^*}{\tau_L^*}\,dt + \sqrt{\frac{2\sigma_{\rm sg}^2}{\tau_L^*}}\,dW_i \tag{1.11}$$

where $\sigma_{\rm sg}$ and τ_L^* stand for the respective velocity and time scales of residual motions *seen* by the particle; moreover, $\tau_{\rm sg}$ denotes the time scale of residual motions. They are estimated from

$$\sigma_{\rm sg} = \sqrt{\frac{2}{3}k_{\rm sg}}\ , \qquad \tau_L^* = f(\tau_{\rm sg}, \tau_p, \sigma_{\rm sg}/g)\ , \quad \tau_{\rm sg} = C\frac{\Delta_f}{\sqrt{\frac{2}{3}k_{\rm sg}}}\ . \tag{1.12}$$

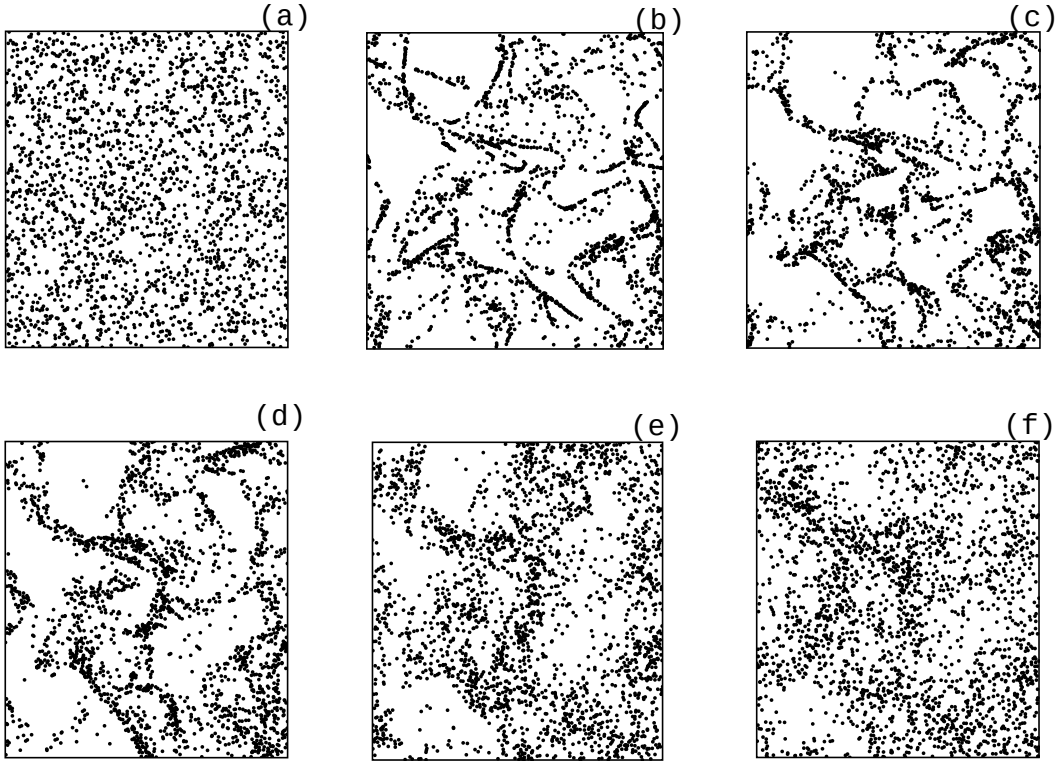


Figure 1.35: Snapshots of particle positions at different Stokes numbers using DNS. Snapshots correspond to same gas-phase velocity field. The DNS computations were done using CDP for $Re_\lambda \approx 40$ on (96^3) grid. Predictions are in good agreement with DNS of Reade & Collins [47].

The model constant $C = \mathcal{O}(1)$ accounts for the uncertainty concerning the time scale of the residual velocity autocorrelation. The prediction $\tau_L^* = \tau_{\text{sg}}$ is expected to work well for small St (also in the limit case of fluid diffusion). A discrete version of the model (unconditionally stable, first-order accuracy in time) becomes

$$u_i^{*(n+1)} = a u_i^{*(n)} + b\xi_i \tag{1.13}$$

where $\Delta t = t^{(n+1)} - t^{(n)}$ is the time interval and ξ_i are random numbers from the standard Gaussian distribution, $\xi_i \in \mathcal{N}(0,1)$. The coefficients a and b are given by the explicit solution of the stochastic differential equation (SDE), Eq. (1.11), with frozen coefficients over a time step Δt:

$$a = e^{-\Delta t/\tau_L^*} , \qquad b = \sigma_{\text{sg}} \sqrt{1 - e^{-2\Delta t/\tau_L^*}} . \tag{1.14}$$

In the particular example of Eq. (1.11), which is a SDE with constant coefficients, the solution provided by Eqs. (1.13) and (1.14) is exact. However, the construction of higher-order numerical schemes for general (variable coefficients) SDEs remains an open issue. It may be interesting to note that in the limit of $\Delta t \gg \tau_L^*$

the scheme (1.13) boils down to generating a series of independent successive velocities $\mathbf{u}^*$, i.e.

$$\mathbf{u}^{*(n+1)} = \sigma_{\rm sg}\boldsymbol{\xi} \ . \tag{1.15}$$

However, for a physically-consistent use of Eq. (1.15), it is imperative that the time intervals for generating a series of independent velocity realizations be $2\tau_L^*$ in order to preserve the correlation time scale.

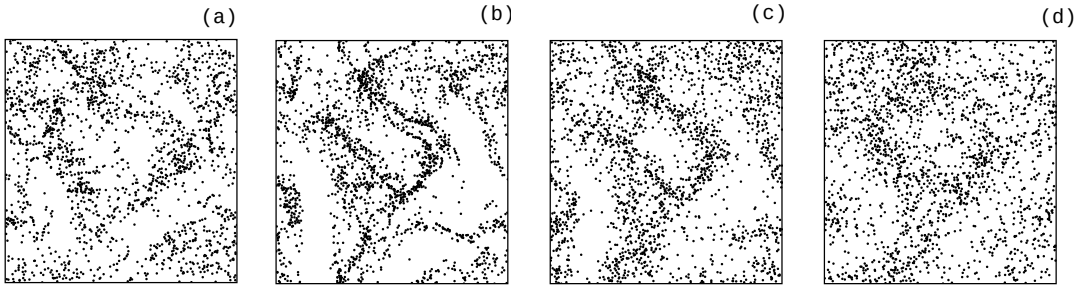


Figure 1.36: Snapshots of particle positions; runs for particles of St $= 2$. a) DNS; b) *a priori* LES with no FPT model; c) *a priori* LES with FPT model and C=0.05; d) DNS for St $= 4$.

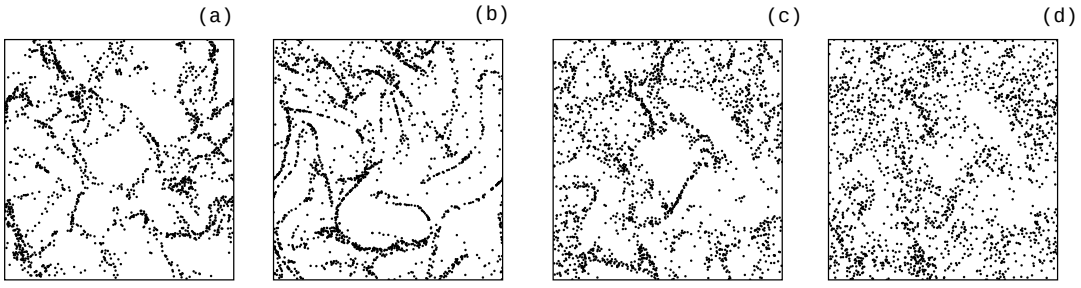


Figure 1.37: Snapshots of particle positions; runs for particles of St $= 0.7$. a) DNS; b) *a priori* LES with with no FPT model; c) *a priori* LES with FPT model and C=0; d) with C=0.05.

In order to test the model, we first performed a DNS of forced isotropic turbulence for $Re_\lambda \approx 40$ on (96^3) with particles of different size (Stokes numbers). Figure 1.35 shows the effect of particle Stokes number on the preferential concentrations obtained from DNS. Here, we use one-way coupling to focus on the subgrid scale effects on particle motion. These computations were performed using CDP and the results obtained were compared to spectral simulations by Reade & Collins [47] to show good agreement. Different Stokes numbers affect the particle relaxation time and preferential concentrations of particles. It is observed that for (St $= 0.7$) we obtain strong preferential concentrations. Next, we filter the gas-phase velocity fields and advect the particles based on this filtered (*a priori LES*) fields. We then apply the model to reconstruct the residual velocity and investigate the effects of the model constant C in equation (1.12) [45]. Here, $k_{\rm sg}$ is readily obtained from the DNS data (raw and filtered). In practical applications, the subgrid kinetic energy of the flow can be estimated from its transport equation [52],[58].

The computational results for two values of the Stokes number and some choices of the model constant are shown in Figs. 1.36 and 1.37. The impact of the residual velocity field, reconstructed in FPT, is readily noticed. As expected, the one-point stochastic model introduced here has a *randomizing* effect on particle locations. For particle sizes larger than that of maximum preferential concentration effect (roughly $\mathrm{St} \approx 0.7$ in our case) the randomizing effect of small scales is lost (the picture of preferential concentration becomes overly sharp), so the model is meant to restore it as shown in Fig. 1.36. In the case of smaller particles that are most influenced by smaller eddies and correlated on a shorter length scale, it is shown in Fig. 1.37 that the filtering does an inverse effect, i.e. it partly kills this short-scale preferential concentration (the picture of preferential concentration becomes somewhat *blurred*), so a potentially successful model for this case should rather be of an *anti-diffusive* character and scale-similarity arguments can be used for its construction.

Here we have shown the impact of LES filtering on particle motion in turbulent flows. In particular, the particle preferential concentration patterns change. A stochastic model to reconstruct the residual velocity field along particle trajectories was developed and implemented in CDP. It was observed that, an improved model should consist of a random ingredient (since the details of residual fluid motion are unknown) and possibly also of a deterministic ingredient, dependent on the structure of the resolved field.

1.5.2 Accounting for Volumetric Effects of Dispersed Phase

Gas-Phase Equations

The fluid mass per unit volume satisfies a continuity equation,

$$\frac{\partial}{\partial t}\left(\rho_g \Theta_g\right) + \bigtriangledown \cdot \left(\rho_g \Theta_g \mathbf{u}_g\right) = 0 \tag{1.16}$$

where ρ_g, Θ_g, and $\mathbf{u}_g$ are fluid density, volume fraction, and velocity, respectively. This indicates that the average velocity field of the fluid phase does not satisfy the divergence-free condition even if we consider an incompressible fluid with droplets. The gas-phase volume fraction is given as $\Theta_g = 1 - \Theta_p$, where Θ_p is the particle phase volume fraction obtained as

$$\Theta_p\left(\mathbf{x}\right) = \sum_{k=1}^{N_p} V_k \mathcal{G}_\sigma\left(\mathbf{x} - \mathbf{x}_{\mathbf{p}k}\right) \tag{1.17}$$

where the summation is over all particles N_p. Here $\mathbf{x}_{\mathbf{p}k}$ is the particle location, $\mathbf{x}$ centroid of a control volume, and V_k the volume of a particle. The function $\mathcal{G}_\sigma$ is the interpolation operator given as

$$\mathcal{G}_\sigma(\mathbf{x}_p) = \frac{\xi_p}{\sigma^3\sqrt{(2\pi)}} exp\left[\frac{-(x-x_p)^2 + (y-y_p)^2 + (z-z_p)^2}{2\sigma^2}\right] \tag{1.18}$$

The resulting Θ_p will be smooth and mass-conserving provided $\mathcal{G}_\sigma$ is smooth. A Gaussian interpolation function with filter-width (σ) proportional to grid size and satisfying $\int_V \mathcal{G}_\sigma\left(\mathbf{x} - \mathbf{x}_{\mathbf{p}k}\right) dV = 1$ is used. The fluid momentum equation is given as

$$\frac{\partial}{\partial t}\left(\rho_g \Theta_g \mathbf{u}_g\right) + \bigtriangledown \cdot \left(\rho_g \Theta_g \mathbf{u}_g \mathbf{u}_g\right) = - \bigtriangledown \left(\Theta_g p\right) + \bigtriangledown \cdot \left(\mu^* \mathbf{D}_c\right) + \mathbf{F} \tag{1.19}$$

where p is the pressure in the fluid phase and $\mathbf{D}_c = \bigtriangledown \mathbf{u}_g + \bigtriangledown \mathbf{u}_g^T$ is the average deformation-rate of the fluid, and F is the force per unit volume the particles exert on gas. These equations are derived in

detail for constant density flows by Joseph & Lundgren [21] and extensions to variable density flows are straightforward. For particulate flows and dilute suspensions at low Reynolds numbers, the fluid viscosity should be replaced by an effective viscosity μ^* by using [53] correlation,

$$\mu^* = \mu_g \left(1 + 2.5\Theta_g + 10.05\Theta_g^2 + 0.00273e^{16.6\Theta_g}\right) \tag{1.20}$$

Dispersed phase equations

The dispersed phase equations in Lagrangian framework are,

$$\frac{d}{dt}\left(\mathbf{x}_p\right) = \mathbf{u}_p \tag{1.21}$$

$$m_p \frac{d}{dt}\left(\mathbf{u}_p\right) = F_p = m_p \mathbf{a}_p \tag{1.22}$$

The corresponding force on the gas-phase is given as

$$F(\mathbf{x}, \mathbf{t}) = -\frac{\mathbf{1}}{V} \sum_{\mathbf{x_p} \mathbf{in} V(\mathbf{x})} \mathbf{F_p} \tag{1.23}$$

Particle Forces

The main issue is to model the force on a particle. This may consist of the standard hydrodynamic drag force, dynamic pressure gradient, gradient of viscous stress in the fluid phase, a generalized buoyancy force, and inter-particle collision. For the present spray application, the density ratio between the liquid and gaseous phase is large ($\approx 10^3$), and we neglect the pressure gradient, viscous stress and buoyancy forces, the total acceleration of the particle $\mathbf{a}_p$ is given as

$$\mathbf{a}_p = \frac{(\mathbf{u}_g - \mathbf{u}_p)}{\tau_r} + \mathbf{a}_{coll} + \left(1 - \frac{\rho_f}{\rho_p}\right)\mathbf{g} \tag{1.24}$$

$\mathbf{a}_{coll}$ is the acceleration due to inter-particle forces. The drag force is caused by the motion of a particle through the gas. The standard expression for τ_r, the particle relaxation time is

$$\frac{1}{\tau_r} = \frac{3}{8} C_d \frac{\rho_g}{\rho_p} \frac{|\mathbf{u}_g - \mathbf{u}_p|}{R_p} \tag{1.25}$$

where C_d is the drag coefficient and is dependent on the particle Reynolds number, volume fraction and viscosity given by [15].

$$\begin{aligned} C_d &= \frac{24}{Re}\left(1 + aRe_p^b\right)\Theta_g^{-1.8}, \quad for \;\; Re_p < 1000 \\ &= 0.44\Theta_g^{-1.8}, \quad for \;\; Re_p \geq 1000 \end{aligned}$$

where C_d is the drag coefficient for spherical particles, $R_p = (3V_p/4\pi)^{1/3}$ is the particle radius. The particle Reynolds number (Re_p) is given as

$$Re_p = \frac{2\rho_g \Theta_g |\mathbf{u}_g - \mathbf{u}_p| R_p}{\mu_g}. \tag{1.26}$$

Computational domain	$63 \times 12 \times 12\ cm$
Grid	$20 \times 21 \times 20$
Fluid density	$1\ g/cm^3$
Fluid viscosity	$1\ poise$
Particle Density	$10.0\ g/cm^3$
Diameter of particles	$0.95\ cm$
Number of Parcels	3780
Particles per parcel	1
Initial array height	$4.75\ cm$
Initial centerline velocity	$360\ cm/s$
Pressure gradient	$20\ dyne/cm^3$

Table 1.3: Parameter description for the simulation of fluidization of spherical particles in a plane Poisuille flow.

There is an indirect collective effect in this drag term: when there is a dense collection of particles passing through the fluid interphase momentum exchange term in equation (1.19) will cause $\mathbf{u}_g$ to approach the particle velocity, $\mathbf{u}_p$ thus decreasing the drag on a particle, a drafting effect.

The acceleration of particles due to inter-particle interactions ($\mathbf{a}_{coll}$) is an important term in dense particulate flows. A simple model for acceleration of particles due to collision ($\mathbf{a}_{coll}$) is

$$\mathbf{a}_{coll} = -\frac{1}{\Theta_p \rho_p} \bigtriangledown \tau \tag{1.27}$$

where τ is the interparticle stress that provides a pressure type force that prevents packing of particles beyond the close-packing limit. Expression for τ is given as (from [50])

$$\tau = \frac{P_s \Theta_p^{\beta}}{\Theta_{cp} - \Theta_p} \tag{1.28}$$

where P_s has units of pressure, Θ_{cp} is the particle volume fraction at close packing and β is a constant. The values for P_s and β are obtained from [50]. In this model, the inter-particle acceleration due to particle collision is assumed to be independent of its size and velocity. This model is least expensive in terms of computational cost as particle binary pairs are not formed and the collision force is directly obtained by interpolation from equation (1.27).

Finite-size effects in Poisuille flow

Choi & Joseph 2001 performed a DNS study of fluidization of circular cylinders (300 particles) arranged at the bottom of a channel in plane Poisuille flow. They observed that with sufficient pressure gradient across the channel, the particles initially at rest in the lower half of the channel start moving and roll over the wall. Particle rotation in a shear flow generates lift and the channel is fluidized after some time. We attempt to capture this phenomenon by our Eulerian-Lagrangian model. The flow parameters are given in Table 1.3. As opposed to [6], we are performing three-dimensional simulations and our particles are spheres. As shown in Figs. (1.38,1.39) we observe that the effect of volume displacement due to particles is to set up instability

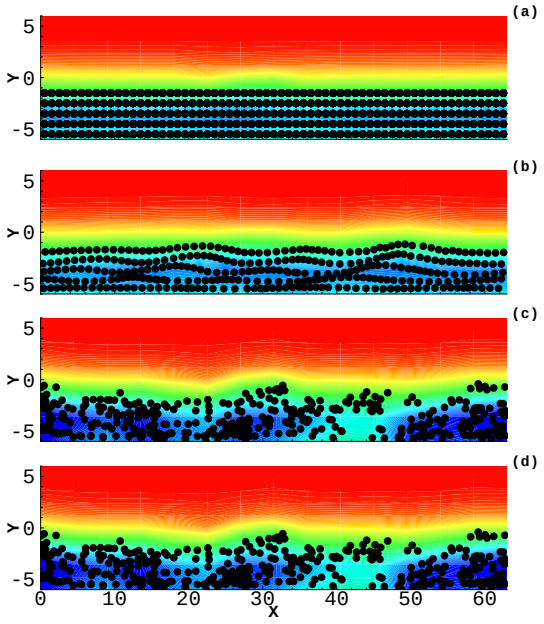


Figure 1.38: Temporal evolution of particle distribution during fluidization by lift in a plane Poisuille flow. Initially all particles are arranged at the bottom of the channel. Superimposed on particle positions is the contour plot of gas-phase volume fraction Θ_g. As particles accelerate from rest, variations in Θ_g gives pressure gradients in vertical direction and produce lift of particles.

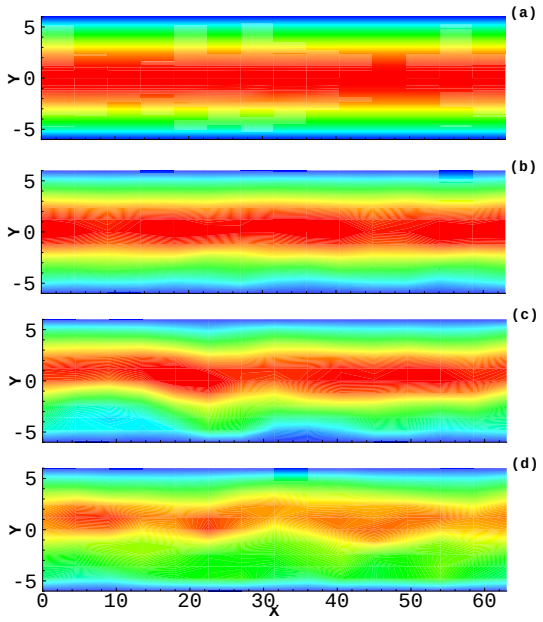


Figure 1.39: Temporal evolution of gas-phase axial velocity in Poisuille flow. Instability waves in axial velocity are clearly seen as heavy particles in layers accelerate underneath a lighter fluid. Similar conditions exit in spray injectors where a liquid sheet is injected into a lighter fluid developing Kelvin-Helmholtz instability waves. We postulate that this effect can be better represented by the dense-spray model compared to standard point-particle approach.

waves between the fluid and particle layers. A vertical pressure gradient is created and gives vertical velocity to the particles and the channel gets fluidized. We also did several test cases, with higher grid resolution, increased density ratios to obtain similar results. With increased particle density, the two-way coupling between the particle and fluid momentum equation, decelerates the fluid in the bottom half of a channel and an inflection point is created in the axial velocity profile. This eventually causes lift and particle dispersion. The same computation was also performed using point-particle approach and fluidization by lift was not observed. Particles tend to move in laminar layers. This indicates that, in dense regions, the point-particle approach is invalid and does not give the correct flow evolution for the gas and particle phases. In liquid-fuel combustor applications, we believe that similar mechanism can be observed in the dense-spray regime near the injector and will allow us to capture some of the important features of primary atomization using the model for finite-size of particles.

1.5.3 Modeling Liquid Films

Advanced injectors of gas-turbine combustors are designed such that liquid spray impinges onto the injector walls and forms an intermittent liquid film. The formation and growth of liquid film in these injectors plays an important role in deciding the liquid breakup process and the final spray size distributions. The standard point-particle approach cannot capture these effects accurately through two-way coupling. We investigated impingement of liquid jet onto a flat plate by using the dense-spray model described above. In order to systematically evaluate the model, we first extended the point-particle approach by including the collision model described by equation(1.28). In addition, in a parallel effort a separate model for film formation was developed where an equation for film-height is solved. These models were implemented into CDP and applied to test problems to study their effects. Detailed validation studies are necessary to assess the accuracy of the models.

Approach 1: Dense Spray Model

Figure 1.40 shows the results obtained from the dense-spray model for impingement of a liquid jet onto a flat plate. In this model problem, there is no gas-phase flow over the flat plate. Liquid jet impinges vertically and spreads to form a film. Here the liquid jet is modeled as a collection of closely packed particles of same density as the liquid. As a first approximation, the particles are assumed as rigid spheres. These particles have an initial velocity of $25\ m/s$ in the vertical direction. They impinge the flat plate and undergo a plastic collision losing all of its normal momentum. A stack of particles builds near the boundary. These stagnant particles oppose the incoming particles through the collision model and spread them in the horizontal plane. The strong momentum coupling between the two-phases creates a velocity field (Fig. 1.40C) and a pressure field (Fig. 1.40D). Formation and growth of the liquid film is clearly observed. We plan to perform a detailed validation study of the model and use it for prediction of spray size distributions in a new injector designed by Pratt & Whitney.

Approach 2: Thin Film Model

The liquid film formed by the impact of a fuel spray onto the injector walls of advanced gas-turbine designs can typically be treated as a thin film. Introducing the lubrication approximations for thin films into the Navier-Stokes equations, an equation for the evolution of the film height h above a no-slip wall can be

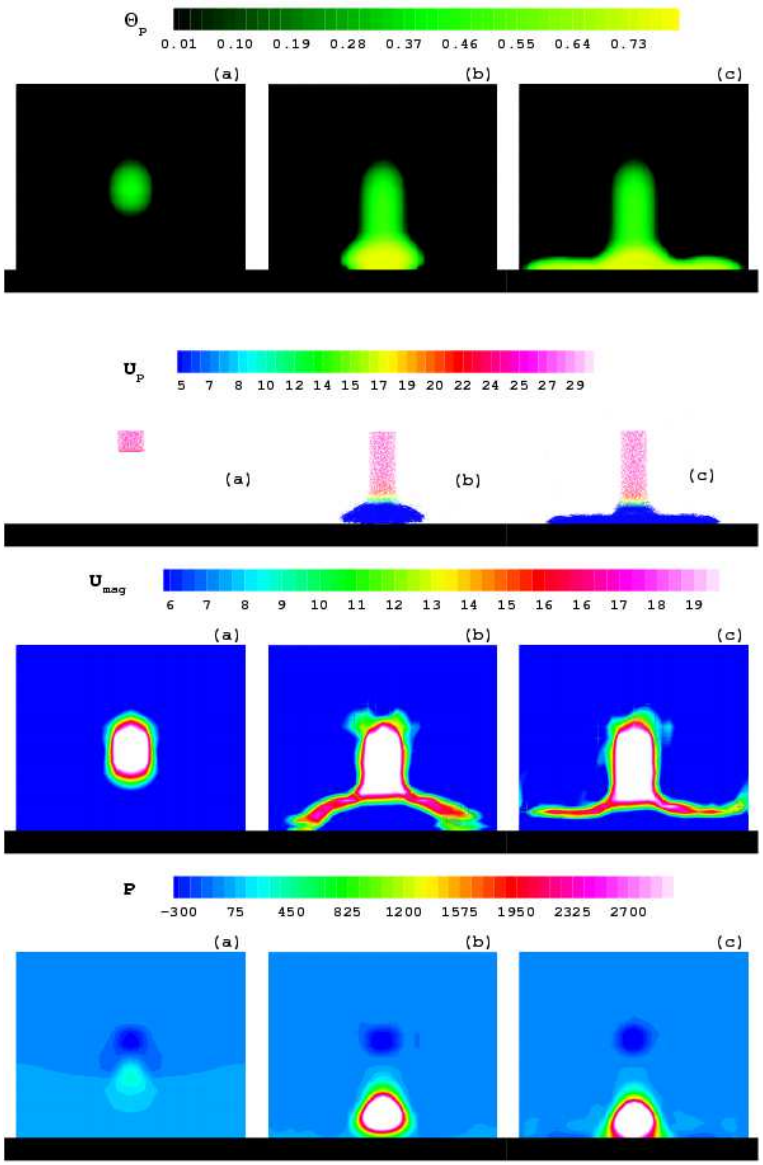


Figure 1.40: Temporal evolution of film formation in direct impact of liquid spray onto a flat plate. Liquid spray is represented by discrete particles of same density. Particles initially clustered at the top move vertically down, impact the flat plate and build up. Particle build up opposes incoming particles, creates a stagnation point, develops a strong gas-phase velocity field through two-way coupling and spreads the film. The present computation includes only the collision model keeping the particle centroids from overlapping each other. (A): Particle volume fraction, (B): particle positions color coded with particle velocity magnitude, (C): gas-phase velocity magnitude, (D): pressure.

derived [32],

$$\frac{\partial h}{\partial t} + \nabla_1 \cdot \left[\frac{1}{2} h^2 \boldsymbol{\tau}_g + \frac{1}{3} h^3 \nabla_1 \left(N_g - \frac{\mathrm{Re}}{\mathrm{Fr}^2} h + \frac{1}{C} \nabla_1^2 h \right) \right] = 0 \,. \tag{1.29}$$

Here, $\boldsymbol{\tau}_g$ and N_g are the tangential respectively normal stresses exerted by the outside gas onto the liquid film surface, cp. Fig. 1.41, Re and Fr are the Reynolds and Froude number, ∇_1 is the Nabla-operator in the x-y-wall coordinate system, and C is the capillarity number,

$$C = \frac{u_0 \mu_l}{\sigma} \,, \tag{1.30}$$

with u_0 a reference velocity, μ_l the liquid viscosity, and σ the surface tension coefficient.

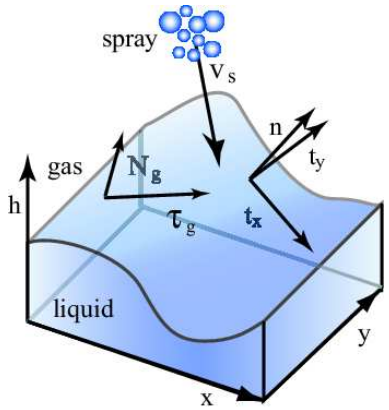


Figure 1.41: Film definition

To include the effect of the impinging spray, (1.29) has to be extended by appropriate source terms. Assuming, as an initial step, that all drops coming into contact with the film surface are fully absorbed into the film such that no splashing occurs, the impinging spray constitutes a mass respectively height source q_s,

$$q_s = \frac{\dot{m}_s}{\rho_l \Delta A} \,, \tag{1.31}$$

where $\dot{m}_s$ is the mass flux of the impinging spray, ρ_l is the liquid density, and ΔA is the size of the wall grid cell where impingement occurs. The inclusion of a detailed splashing model is straightforward and will be considered in the future. Besides its mass, each absorbed drop transfers its momentum to the film. Here, we assume that all momentum is transferred to the film within the single grid cell where absorption occurs. This results in additional normal and tangential stresses,

$$N_s = \mathrm{Re} \frac{(m\boldsymbol{v})_s}{\delta t_I \Delta A} \cdot \boldsymbol{n} \tag{1.32}$$

$$\boldsymbol{\tau}_s = \mathrm{Re} \frac{(m\boldsymbol{v})_s}{\delta t_I \Delta A} \cdot \boldsymbol{t} \,, \tag{1.33}$$

with $(m\boldsymbol{v})_s$ the impacting spray momentum, $\boldsymbol{n}$ and $\boldsymbol{t}$ the film surface normal and tangential vector, cp. Fig. 1.41, and δt_I an impact time scale.

Including all spray impingement effects, the evolution equation for the film height then reads

$$\frac{\partial h}{\partial t} + \nabla_1 \cdot \left[\frac{1}{2} h^2 \left(\boldsymbol{\tau}_g + \boldsymbol{\tau}_s \right) + \frac{1}{3} h^3 \nabla_1 \left(N_g - \frac{\mathrm{Re}}{\mathrm{Fr}^2} h + N_s + \frac{1}{C} \nabla_1^2 h \right) \right] = q_s \,. \tag{1.34}$$

Finally, the effect of the liquid film on the outer flow field has to be taken into account by modifying the respective boundary conditions. First, the boundary conditions have to be moved a distance of h away from the wall, and second, the usual no-slip velocities have to be replaced by the film surface velocities,

$$\boldsymbol{v}^{CDP} = \boldsymbol{n} \cdot w_h + \boldsymbol{t} \cdot \boldsymbol{v}_{t,h} \,, \tag{1.35}$$

with

$$\boldsymbol{v}_{t,h} = h \boldsymbol{\tau}_g + \frac{1}{2} \nabla_1 \cdot \left(N_g - \frac{\mathrm{Re}}{\mathrm{Fr}^2} h + N_s + \frac{1}{C} \nabla_1^2 h \right) h^2 \tag{1.36}$$

and

$$w_h = \frac{\partial h}{\partial t} + \boldsymbol{v}_{t,h} \cdot \nabla_1 h \,. \tag{1.37}$$

Numerical method

The film equation, (1.34), constitutes a fourth order PDE with a stable time step restriction of

$$\begin{aligned} \Delta t_h = \min \quad \Bigg(& \alpha_1 \frac{\Delta x_{\min}}{|\boldsymbol{\tau}_g + \boldsymbol{\tau}_s|_{\max} h_{\max}}, \alpha_2 \frac{C \Delta x_{\min}^4}{h_{\max}^3}, \alpha_2 \frac{C \Delta x_{\min}^3}{h_{\max}^2}, \\ & \alpha_3 \frac{1}{|q_s|_{\max}}, \alpha_4 \frac{\Delta x_{\min}}{\max\left(\partial N/\partial x, \partial N/\partial y\right) h_{\max}^2} \Bigg) \,, \end{aligned} \tag{1.38}$$

where α_i are CFL numbers of order 1. Since, in the worst case, $\Delta t_h \sim \Delta x^4$, severe time step restrictions can be imposed on an explicit solver, usually necessitating the use of an implicit solver. However, since (1.34) is incorporated into CDP, the CPU time spent to solve the film equation on a typical CDP grid employing an explicit solver is negligible compared to the CPU time spent in a single CDP time step. This is due to the fact that (1.34) is only two-dimensional.

In practice, (1.34) is solved on a separate, two-dimensional wall grid, using a first order upwind scheme for the advection term involving the tangential stresses and a positivity preserving scheme [10] employing central differences for all higher order terms. Integration in time is first order. Bilinear interpolation is used to transfer quantities from the wall grid to the CDP grid and vice versa. However, (1.35) has been neglected for now.

To capture the spreading of the film on the underlying wall, a precursor film model is employed [13, 54]. This circumvents the explicit treatment of contact line angles and thereby avoiding any contact line singularities.

Results

The implementation of the film equation has been verified comparing the numerical solutions of traveling surface waves under shear stress to analytical solutions. The obtained results show excellent agreement with the theory.

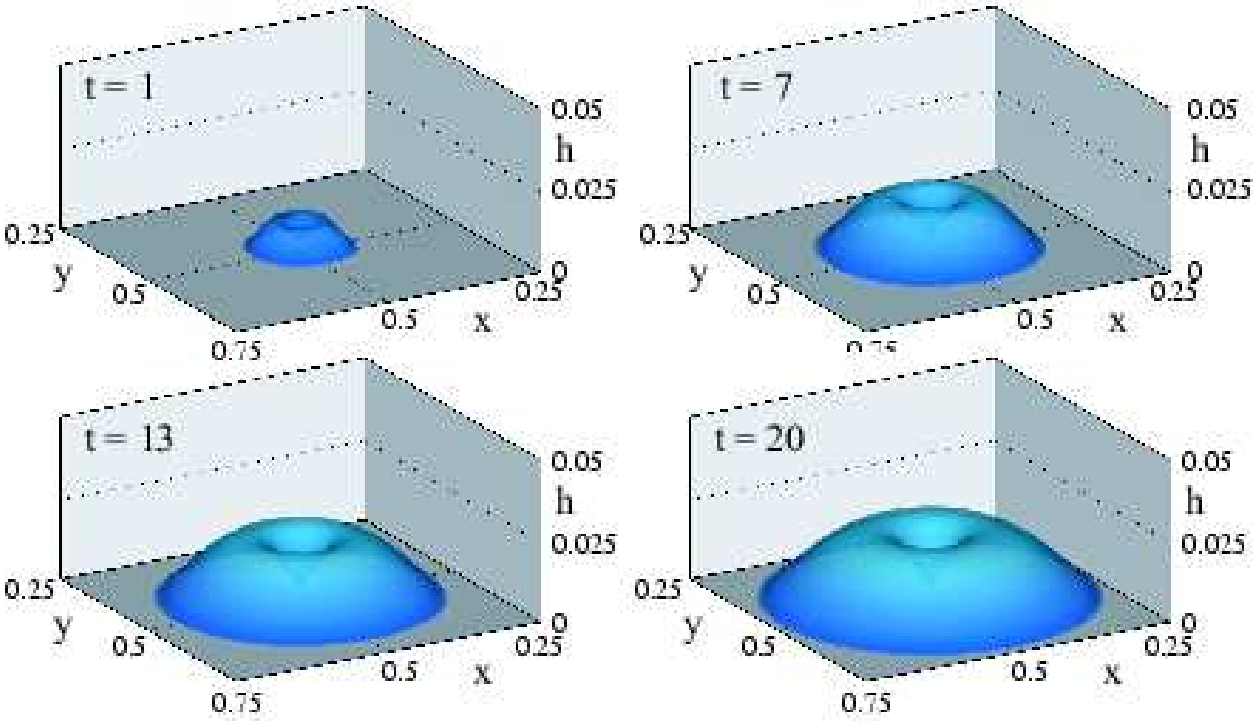


Figure 1.42: Spreading of a film on a flat plate

To demonstrate the performance of the film model in CDP, the results of two different example cases are presented in the following. In the first case, a spray with density $\rho_l = 850\,\mathrm{kg/m^3}$ and $\dot{m} = 1\,\mathrm{kg/s}$ impacts vertically onto a $1\,\mathrm{m}^2$ square flat plate in a square area of $16\,\mathrm{cm}^2$ located at the center of the plate. The spray is injected just above the film, so that the momentum coupling between the spray and the gas-phase can be neglected. The capillarity number is $C = 0.018$, the Reynolds number $\mathrm{Re}_s = 9700$, and the Froude number $\mathrm{Fr} = 0.023$. Figure 1.42 shows the spreading of the developing film. The effect of the momentum transfer from the spray to the film is clearly visible in the impact area, where the film is visibly depressed. As time progresses, the film starts to spread and forms a circular wetting area, resulting in a maximum film height of roughly $2\,\mathrm{mm}$.

In the second example, a spray with $\dot{m} = 10.9\,\mathrm{g/s}$ is injected in a 45^o angle onto a $2\pi \times \pi$ sized wall of a turbulent channel with Reynolds number $\mathrm{Re} = 395$. The capillarity number is $C = 0.25$, the film Reynolds number $\mathrm{Re} = 1.3 \cdot 10^5$, and the Froude number $\mathrm{Fr} = \infty$. To highlight the effect of the outside flow field on the film evolution, the gas-phase stresses acting on the film surface have been enhanced by a factor of 100. Figure 1.43 shows the evolution of this film. After the initial impact, the film is quickly pushed along the channel wall by the outside gas-phase flow and starts to spread in the transverse direction. Note that due to the turbulent fluctuations of the gaseous shear stresses, fluctuations in the film height are generated.

In conclusion, the presented model to simulate the evolution of a liquid film generated by an impinging spray in a turbulent gas-phase flow shows great promise in describing the physics of advanced gas turbine injector designs.

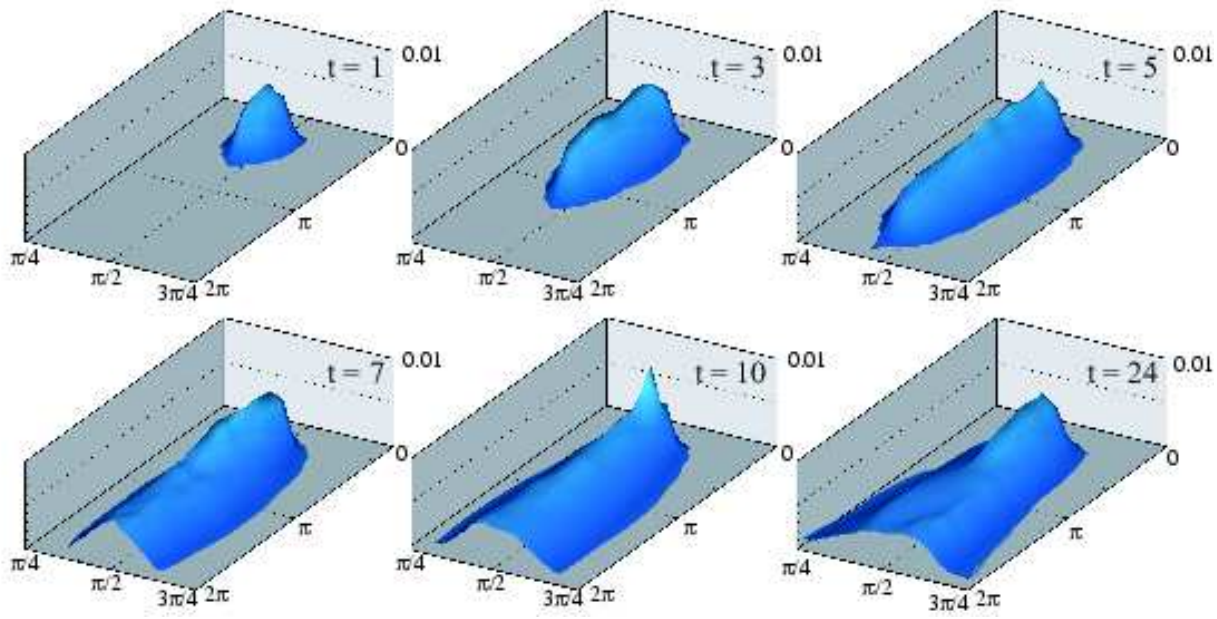


Figure 1.43: Spreading of a film in a turbulent channel

1.6 Compressible Formulation

A non-dissipative, robust, implicit algorithm is proposed for direct numerical and large–eddy simulation of compressible turbulent flows. The algorithm addresses the problems caused by low Mach numbers and under–resolved high Reynolds numbers. It colocates variables in space to allow easy extension to unstructured grids, and discretely conserves mass, momentum and total energy. The Navier–Stokes equations are non–dimensionalized using an incompressible scaling for pressure, and the energy equation is used to obtain an expression for the velocity divergence. A pressure–correction approach is used to solve the resulting equations, such that the discrete divergence is constrained by the energy equation. As a result, the discrete equations analytically reduce to the incompressible equations at very low Mach number; i.e. the algorithm overcomes the acoustic time-scale limit without preconditioning or solution of an implicit system of equations. The algorithm discretely conserves kinetic energy in the incompressible inviscid limit, and is robust for inviscid compressible turbulence on the convective time-scale. These properties make it well-suited for DNS/LES of compressible turbulent flows.

1.6.1 Motivation for proposed approach

A robust algorithm for compressible turbulence may be derived by requiring that the discrete equations reduce to the incompressible equations at low Mach numbers, and that the discretization conserves kinetic energy in the inviscid incompressible limit. The behavior of the divergence of the velocity is described by Thompson (1988). The continuity, momentum and energy equation (expressed in terms of entropy) can be

rearranged to obtain

$$\frac{\partial u_j^*}{\partial x_j^*} = \frac{c_r^2}{2c^2} M_r^2 u_j^* \frac{\partial u_i^* u_i^*}{\partial x_j^*} + \frac{c_r^2}{c^2} \frac{l_r}{c_r t_r} \left(\frac{M_r}{2} \frac{\partial u_i^* u_i^*}{\partial t^*} - \frac{1}{\rho^*} \frac{\partial p^*}{\partial t^*} \right)$$
$$- \frac{c_r^2}{c^2} \frac{1}{\rho^*} \frac{M_r^2}{Re_r} \left[u_j^* \frac{\partial^2 u_j^*}{\partial x_k^* \partial x_k^*} + \left(\frac{\mu_B}{\mu} + \frac{1}{3} \right) u_j^* \frac{\partial}{\partial x_j^*} \frac{\partial u_k^*}{\partial x_k^*} \right]$$
$$+ \frac{1}{Re_r} \left(\frac{\partial v^*}{\partial T^*} \right)_p \left[(\gamma - 1) M_r^2 \Gamma^* + \frac{1}{Pr} \frac{\partial^2 T^*}{\partial x_j^* \partial x_j^*} \right]. \tag{1.39}$$

Here, u_i, ρ, p, T denote the velocities, density, pressure and temperature respectively, μ, μ_B and κ denote the dynamic viscosity, bulk viscosity and thermal conductivity respectively, c denotes the speed of sound, and $\Gamma = \tau_{ij} \frac{\partial u_i}{\partial x_j}$ denotes the viscous dissipation. τ_{ij} is the viscous stress in the momentum equation. The superscript, '*' denotes non–dimensional variables, and

$$u_i^* = u_i/u_r, \;\; x_i^* = x_i/l_r, \;\; p^* = p/\left(\rho_r u_r c_r\right), \;\; \rho^* = \rho/\rho_r, \tag{1.40}$$
$$T^* = T/T_r, \quad t^* = t/t_r, \;\; \Gamma^* = \Gamma \left(l_r^2/\mu u_r^2\right).$$

Also, M_r and Re_r denote the Mach number and Reynolds number based on the reference state, and c_p, v and γ denote the specific heat at constant pressure, specific volume and ratio of specific heats respectively. Note that the reference time–scale, t_r can be chosen to be convective (l_r/u_r) or acoustic (l_r/c_r). Equation 1.39 shows that if $M_r = u_r/c_r$ is small, and the flow is tracked on the convective time–scale ($t_r = l_r/u_r$), then all terms on the right–hand side drop out and the velocity–field is divergence free. If on the other hand, the variation of the flow on the acoustic time–scale is sought ($t_r = l_r/c_r$), the unsteady pressure term will produce a finite divergence. This implies that a combination of the continuity, momentum and energy equations may be used to provide a constraint on the divergence of the velocity field, which will ensure that the compressible equations yield the incompressible equations in the limit of zero Mach number. This is the approach used in this paper to derive an algorithm, which in the limit of small Mach number is discretely energy–conserving.

1.6.2 Governing Equations

The governing equations are the continuity, and compressible Navier-Stokes equations for an ideal gas,

$$\frac{\partial \rho^d}{\partial t^d} + \frac{\partial \rho^d u_j^d}{\partial x_j^d} = 0, \tag{1.41}$$
$$\frac{\partial \rho^d u_i^d}{\partial t^d} + \frac{\partial \rho^d u_i^d u_j^d}{\partial x_j^d} = -\frac{\partial p^d}{\partial x_i^d} + \frac{\partial \tau_{ij}^d}{\partial x_j^d}, \tag{1.42}$$
$$\frac{\partial \rho^d E^d}{\partial t^d} + \frac{\partial (\rho^d E^d + p^d) u_j^d}{\partial x_j^d} = \frac{\partial \tau_{ij}^d u_i^d}{\partial x_j^d} + \frac{c_p^d}{Pr} \frac{\partial^2 \mu^d T^d}{\partial x_j^d \partial x_j^d}. \tag{1.43}$$

where the superscript 'd' denotes dimensional values. The variables ρ and u_i denote the density and velocity respectively, and $E = c_v T + u_i u_i/2$ denotes the total energy per unit mass. $\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right)$ is the viscous stress tensor, c_p is the specific heat at constant pressure, and Pr denotes the Prandtl number.

The above equations are non–dimensionalized using $u_r,\ L,\ \rho_r,\ T_r$ as the reference velocity, length, density and temperature respectively. The reference pressure, $p_r = \rho_r R T_r$. The following non–dimensional variables are defined:

$$\rho = \frac{\rho^d}{\rho_r},\ \ u_i = \frac{u_i^d}{u_r},\ \ t = \frac{t^d}{L/u_r},\ \ \mu = \frac{\mu^d}{\mu_r},$$
$$p = \frac{p^d - p_r}{\rho_r {u_r}^2},\ \ T = \frac{T^d}{T_r},\ \text{ and }\ M_r = \frac{u_r}{a_r} = \frac{u_r}{\sqrt{\gamma R T_r}}. \tag{1.44}$$

Note that the pressure has been non–dimensionalized as $p = \frac{p^d - p_r}{\rho_r {u_r}^2}$. Such non–dimensionalization is motivated by Thompson's (1988) analysis, and the work of Bijl & Wesseling (1998) and van der Heul et al (2002). This yields the following non–dimensional equations:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0, \tag{1.45}$$
$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial \tau_{ij}}{\partial x_j}, \tag{1.46}$$

$$M_r^2 \left[\frac{\partial}{\partial t}\left(p + \frac{\gamma - 1}{2}\rho u_i u_i \right) + \frac{\partial}{\partial x_j}\left(\gamma p + \frac{\gamma - 1}{2}\rho u_i u_i \right) u_j \right] + \frac{\partial u_j}{\partial x_j}$$
$$= \frac{(\gamma - 1) M_r^2}{Re}\frac{\partial \tau_{ij} u_i}{\partial x_j} + \frac{1}{RePr}\frac{\partial}{\partial x_j}\left(\frac{\mu \partial T}{\partial x_j} \right). \tag{1.47}$$

The non–dimensional equation of state is:

$$\rho T = \gamma M_r^2 p + 1. \tag{1.48}$$

Note that as M_r tends to zero, the energy equation (1.47) requires that $\frac{\partial u_j}{\partial x_j} = \frac{1}{RePr}\frac{\partial}{\partial x_j}\left(\frac{\mu \partial T}{\partial x_j}\right)$. Along with the continuity equation, this shows that the velocity field is divergence free if the density and temperature are constant. On the other hand, if the Boussinesque approximation is made, an advection–diffusion equation is obtained for temperature. The equation of state similarly reduces to $\rho T = 1$. The above non–dimensional equations therefore naturally yield the incompressible equations in the limit of very small Mach number. Also, all spatial derivatives in the above equations are in divergence form, and hence conservative. The above set of governing equations are therefore very attractive in that at high Mach numbers, they would yield the proper jump in variables across shock waves, and at very small Mach numbers, variations on the fast, acoustic time–scale would be projected out at time–steps larger than the acoustic time–scale. The primary reason for the above behavior of the non–dimensional equations is the definition of the non–dimensional pressure as $p = \frac{p^d - p_r}{\rho_r {u_r}^2}$. Note that alternative definitions of non–dimensional pressure as $p = p^d/(\rho_r u_r^2)$ or $p = p_d/p_r$ yield

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{1}{M_r^2}\frac{\partial p}{\partial x_i} \tag{1.49}$$

for the inviscid momentum equation, which is singular when M_r tend to zero.

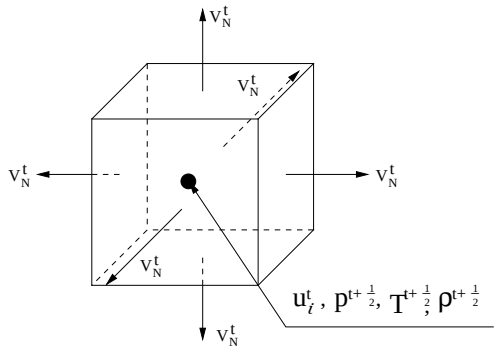


Figure 1.44: Schematic of the storage of variables for a three–dimensional Cartesian control volume. The Cartesian velocities, pressure, temperature and density are stored at the centroids of the control volumes. The face–normal velocity is stored at the centers of the faces.

1.6.3 Discretization

Figure 1.44 shows the storage of variables. The Cartesian velocities, pressure and density are colocated in space at the centroids of the control volumes. Also, density, pressure and temperature are staggered in time from velocity. This feature makes the discretization symmetric in space and time, and is essential to ensuring zero dissipation at finite time-steps. Such staggering in time was also used by Pierce & Moin (2001), and Wall et al. (2002) for the zero–Mach number equations, and the compressible Euler equations respectively. The face normal velocity is located at center of the faces of the control volume and denoted by subscript v_N in this paper. As seen below, v_N is obtained by projection and not interpolation. At every time step,the velocity components u_i, and v_N are advanced from time t to $t+1$ and the thermodynamic variables, p, ρ and T are advanced from $t+\frac{1}{2}$ to $t+\frac{3}{2}$.

Integrating the governing equations over a control volume, and using Gauss' theorem to transform volume integrals into surface integrals yields the following discrete equations. The discrete continuity equation is

$$\frac{\rho_{cv}^{t+\frac{3}{2}} - \rho_{cv}^{t+\frac{1}{2}}}{\Delta t} + \frac{1}{V}\sum_{faces} \rho_{face}^{t+1} v_N^{t+1} A_{face} = 0, \tag{1.50}$$

where ρ_{cv} denotes $\rho_{i,j,k}$, and $\sum_{faces}$ denotes summation is over all the faces of the control volume. The variables, ρ_{face} and v_N denote the density and normal velocity at each of the faces of the volume. For example $\rho_{face} = \frac{1}{2}\left(\rho_{i,j,k} + \rho_{i+1,j,k}\right)$ denotes the density at the face at $i+\frac{1}{2}, j, k$, and is obtained by interpolation. The corresponding face–normal velocity is $u_{i+\frac{1}{2},j,k}$ which is obtained by projection and not interpolation. The variables, V and A_{face} denote the volume of the control volume and the area of the faces of the control volume. Also, ρ_{face}^{t+1} is obtained by symmetric interpolation in time; i.e. $\rho_{face}^{t+1} = \left(\rho_{face}^{t+\frac{1}{2}} + \rho_{face}^{t+\frac{3}{2}}\right)/2$.

The discrete momentum equation for the i component of momentum is

$$\frac{g_{i,cv}^{t+1} - g_{i,cv}^{t}}{\Delta t} + \frac{1}{V}\sum_{faces} g_{i,face}^{t+\frac{1}{2}} v_N^{t+\frac{1}{2}} A_{face} = -\frac{\partial}{\partial x_i} p_{cv}^{t+\frac{1}{2}} + \frac{1}{Re}\frac{1}{V}\sum_{faces}\left(\tau_{ij}^{t+\frac{1}{2}}\right)_{face} N_j A_{face}. \tag{1.51}$$

Here, $g_i = \rho u_i$ denotes the momentum in the i direction and $(\tau_{ij})_{face}$ is the stress tensor at the face. N_j is the outward normal vector at the face. $p_{cv}^{t+\frac{1}{2}}$ is obtained by applying the trapezoidal rule to integrating the pressure–gradient term; i.e.,

$$\begin{aligned}
\frac{\partial}{\partial x_i} p^{t+\frac{1}{2}} &= \frac{1}{2}\left[\frac{\partial}{\partial x_i} p^{t} + \frac{\partial}{\partial x_i} p^{t+1}\right] \\
&= \frac{1}{4}\left[\frac{\partial}{\partial x_i} p^{t-\frac{1}{2}} + \frac{\partial}{\partial x_i} p^{t+\frac{1}{2}} + \frac{\partial}{\partial x_i} p^{t+\frac{1}{2}} + \frac{\partial}{\partial x_i} p^{t+\frac{3}{2}}\right] \\
&= \frac{\partial}{\partial x_i}\left(\overline{p}^{t}\right),
\end{aligned} \tag{1.52}$$

where

$$\overline{p}^{t} = \frac{p^{t-\frac{1}{2}} + p^{t+\frac{1}{2}} + p^{t+\frac{1}{2}} + p^{t+\frac{3}{2}}}{4}. \tag{1.53}$$

The gradient theorem is used to compute the pressure gradient as,

$$\begin{aligned}
\frac{\partial}{\partial x_i}\overline{p}^{t} &= \frac{1}{V}\sum_{faces} \overline{p}_{face}^{t} N_i A_{face} \\
&= \frac{1}{V}\sum_{faces} \frac{\overline{p}_{cv}{}^{t} + \overline{p}_{nb}^{t}}{2} N_i A_{face} \\
&= \frac{1}{V}\sum_{faces} \frac{\overline{p}_{nb}^{t}}{2} N_i A_{face}.
\end{aligned} \tag{1.54}$$

Here, the subscript 'nb' denotes the neighboring control volume that is attached to the face under consideration. The discrete energy equation is given by

$$\begin{aligned}
&M_r^2\left[\frac{\partial}{\partial t}\left(p_{cv} + \frac{\gamma-1}{2}\rho u_i u_i\right)^{t+\frac{1}{2}} + \frac{1}{V}\sum_{faces}\left(\gamma p_{cv} + \frac{\gamma-1}{2}\rho u_i u_i\right)_{face}^{t+\frac{1}{2}} \cdot v_N^{t+\frac{1}{2}} A_{face}\right] \\
&+\frac{1}{V}\sum_{face} v_N^{t+\frac{1}{2}} A_{face} = \frac{(\gamma-1)M_r^2}{Re}\frac{1}{V}\sum_{faces}\left(\tau_{ij}u_i\right)_{face}^{t+\frac{1}{2}} N_j A_{face} \\
&+\frac{1}{RePr}\frac{1}{V}\sum_{faces}\left(\mu\frac{\partial T^{t+\frac{1}{2}}}{\partial N}\right) A_{face}.
\end{aligned} \tag{1.55}$$

Note that the density in the momentum g, has to be interpolated in time to relate the momentum and velocity, because of the time staggering between density and velocity. The central differences in time and space make the algorithm second order on regular grids. Also, the algorithm is fully implicit, and hence not limited by viscous, convective or acoustic stability limits. The discrete energy equation, shows that M_r and time step, Δt determine whether high frequency acoustics are captured in a time–accurate manner. At small Mach numbers, a timestep of the order of M_r^2 allows acoustic waves to be represented in a time–accurate manner. When high frequency acoustics are not of physical importance, the time–step may be of the order of the convective time–scale, and the energy–equation discretely projects out acoustic effects and yields zero–divergence for the velocity field.

1.6.4 Implementation

The following iterative procedure is used to solve the above discrete equations.

1. Initialize the outer loop; i.e.

$$u_i^{t+1,0} = u_i^t,\ \rho^{t+\frac{3}{2},0} = \rho^{t+\frac{1}{2}},\ T^{t+\frac{3}{2},0} = T^{t+\frac{1}{2}},\ v_N^{t+1} = v_N^t.$$

2. Advance the continuity equation to get $\rho^{t+\frac{3}{2},k+1}$ by using the face normal velocity $v_N^{t+1,k}$.

3. Advance the momentum predictor equation to get a provisional value of g_i^* by using pressure and velocity at current iteration step.

4. Obtain velocities at the control volume centers using $u_i^* = g_i^*/\rho^{t+1,k+1}$ where $\rho^{t+1,k+1} = \left(\rho^{t+3/2,k+1} + \rho^{t+1/2}\right)/2$. Interpolate u_i^* to obtain v_N^* at the faces.

5. Solve the pressure correction equation to get δp.

6. Update the pressure, momentum and the velocities at center of the control volumes, and update the face normal velocity by equation respectively. Note that this implies that the face normal velocity is obtained from a projection and not by interpolating the cell–centered velocities.

7. Check convergence for the pressure correction, density and momentum between outer loop iterations.

A parallel, algebraic multigrid approach is used to solve the equation systems arising from each of the continuity, momentum and energy equations. The structured grid interface of the *Hypre* library (Lawrence Livermore National Laboratory 2003) is used for this purpose.

Table 1.4: Comparison between predicted and calculated values for periodic shock tube.

	ρ_2	ρ_3	T_2	T_3	U_2	s
Predicted	0.613	1.223	1.393	0.698	0.821	1.608
calculated	0.614	1.214	1.394	0.702	0.821	1.590

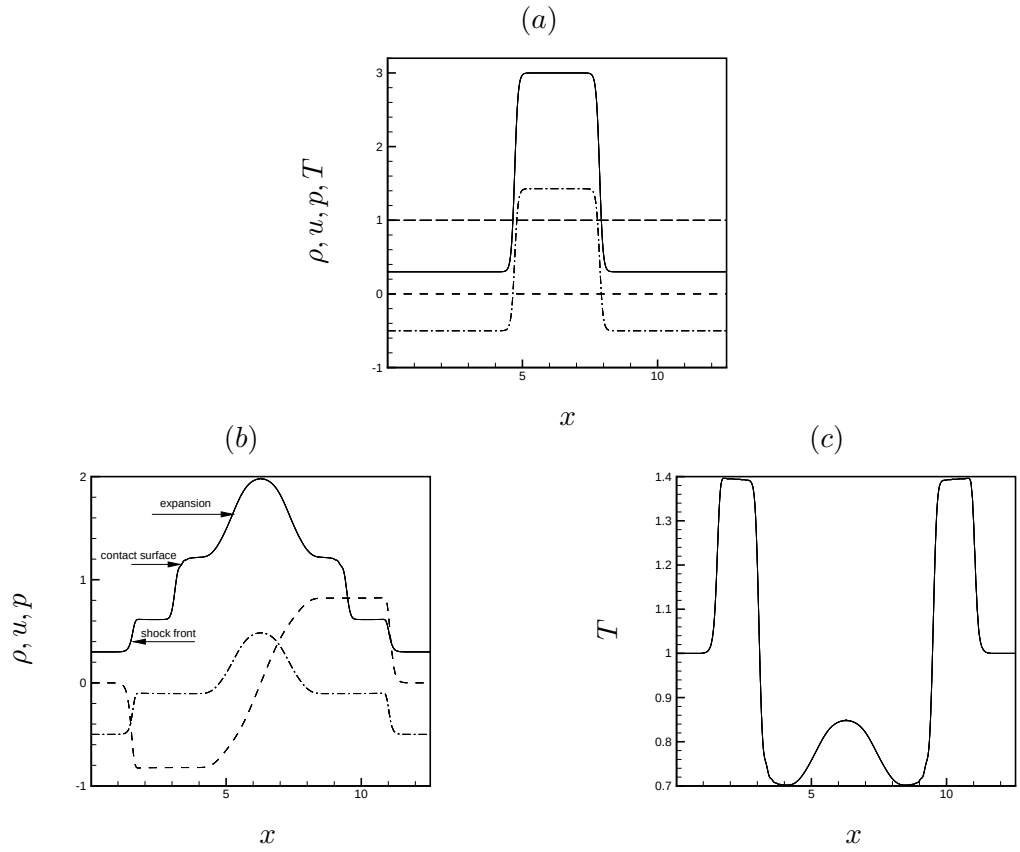


Figure 1.45: (a) Initial conditions for the periodic shock tube problem.—— density, - - - - velocity, —·— pressure ····· temperature, (b) Instantaneous solution at t=1.0. —— density, - - - - velocity, —·— pressure. (c) Instantaneous temperature at t=1.0.

1.6.5 Results I: Shock tube problem

A periodic shock tube problem is computed to illustrate the conservative properties of the algorithm. Figure $1.45(a)$ shows the initial condition. The gas is initially at rest with uniform temperature, pressure and a jump in density. When the flow is advanced, the initial density discontinuity generates a shock wave (with height ρ_2, U_2, T_2 moving with speed s) and an expansion wave that moves in the opposite direction. The shock and expansion waves are separated by a contact discontinuity with height ρ_3, T_3. Figure 1.45 shows instantaneous profiles of pressure, density and velocity at time $t = 1$ for a case where $Re = 200$. The reference Mach number $M_r = 1.0$, and the computations were performed on a domain of length 4π, using a uniform grid of 512 nodes at a time–step of 10^{-3}. The thickness of the shock wave is resolved using approximately 6 points under these conditions. The jumps in density, pressure, velocity and temperature across the shock and expansion waves are compared to analytical solution in table 1.4, and seen to agree within 0.7%.

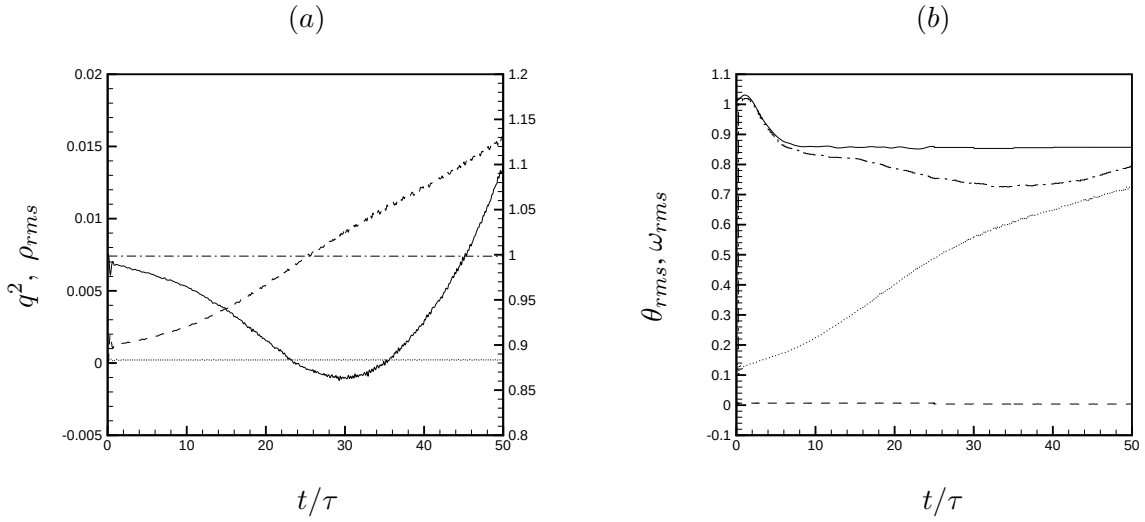


Figure 1.46: Long–time evolution of inviscid, isotropic, compressible turbulence. (a) —— turbulence kinetic energy at $M_t = 0.4$, ---- rms density at $M_t = 0.4$, —·— turbulence kinetic energy at $M_t = 0.01$, ····· rms density at $M_t = 0.01$. The range of kinetic energy is shown on the right of the plot, while the left of the plot shows the range of rms density. (b) —— rms vorticity at $M_t = 0.01$, ---- rms dilatation at $M_t = 0.01$, —·— rms vorticity at $M_t = 0.4$, ····· rms dilatation at $M_t = 0.4$. Both quantities are normalized by the initial rms vorticity.

1.6.6 Results II: Isotropic Turbulence

The algorithm is applied to simulate homogeneous, decaying isotropic turbulence on a very coarse grid without a subgrid model. This problem poses a severe test of the ability of the algorithm to ensure robustness at high Reynolds numbers without numerical dissipation. The initial fluctuation Mach number, $M_t = q/a$ and turbulent Reynolds number, $Re_\lambda = \frac{u_{rms}\lambda}{\nu}$. Here, $q = \sqrt{u_i' u_i'}$, $a = \sqrt{\gamma R T_0}$ is the mean speed of sound, and λ denotes the initial Taylor microscale. The initial radial energy spectrum is

$$E(k) = 16\sqrt{\frac{2}{\pi}}\frac{u_0^2}{k_0}\left(\frac{k}{k_0}\right)^4 e^{-2(\frac{k}{k_0})^2}. \tag{1.56}$$

This initial spectrum is used to generate divergence–free, initial velocity field as suggested by Rogallo (1981). $k_0 = 5$, and the initial fluctuations in pressure, temperature and density are set to zero.

Simulations are performed for compressible ($M_t = 0.4$), and nearly incompressible ($M_t = 0.01$) conditions. For each simulation, two Reynolds numbers, ($R_\lambda = 100$ and $R_\lambda = \infty$; i.e. inviscid) are considered. The domain is $(2\pi)^3$, and the computational grid has 32 points in each direction. The time–step is fixed at 0.025 τ for all the simulations. Here, τ is an 'eddy–turnover' time–scale which is equal to the initial value of λ/u_{rms}. Note that even the lower Reynolds number solution is not resolved by the computational grid. Also, no subgrid model is used.

An interesting aspect of homogeneous, compressible turbulence is that acoustic fluctuations generated by the initial transient will be constrained by the periodic boundary conditions, and steepen due to the nonlinearity, while decaying at a rate determined by viscosity. In the inviscid limit therefore, even the $M_t = 0.4$ and $M_t = 0.01$ flows will form shock waves in finite time. A non–dissipative, scheme will

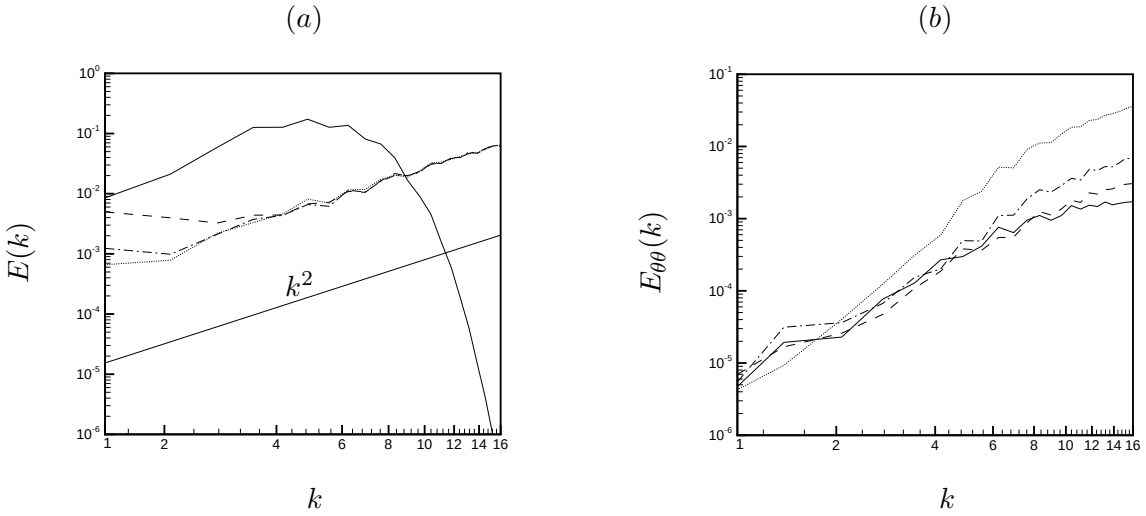


Figure 1.47: Three–dimensional energy ($E(k)$), and dilatation ($E_{\theta\theta}$) spectra as a function of time for inviscid, isotropic, compressible turbulence. The energy spectrum is normalized by the initial value of q^2, and dilatation spectrum is normalized by initial ω_{rms}^2. (a) Kinetic energy spectrum for $M_t = 0.01$, —— $t/\tau = 0$, - - - - $t/\tau = 5$, —·— $t/\tau = 10$, ····· $t/\tau = 70$, (b) Dilatation spectrum for $M_t = 0.4$, —— $t/\tau = 1$, - - - - $t/\tau = 4$, —·— $t/\tau = 8$, ····· $t/\tau = 56$.

capture this behavior provided it is stable on the shock–formation time–scale. This is illustrated in figure $1.46(a)$ which shows the solution for very long times. Note that the $M_t = 0.4$ solution is stable for about 30τ, following which the kinetic energy, and density fluctuations increase rapidly. This behavior is due to the formation of shock waves in the domain. Figure $1.46(b)$ shows that rms level of dilatation is comparable to that of vorticity when the kinetic energy starts increasing. In contrast the $M_t = 0.01$ flow maintains its initial kinetic energy over this considerable length of time; also the level of dilatation fluctuations is noticeably lower.

This behavior is explained by figure 1.47 which shows three–dimensional spectra of the turbulence kinetic energy for the $M_t = 0.01$ flow, and dilatation for the $M_t = 0.4$ flow. Note that the nearly incompressible flow exhibits equipartition; i.e. its spectrum at long times varies as k^2. The dilatation spectrum for the $M_t = 0.4$ flow explains its behavior at long times. Note that the levels of dilatation fluctuations increase with time, with most of the increase being at the smallest scales. This behavior is consistent with the formation of shock waves. These results therefore show that the algorithm is stable even in the inviscid limit on the convective time-scale, and is unstable only when shock waves form. This behavior is in contrast to other commonly used non–dissipative schemes which become unstable at very short times even at very low Mach numbers; i.e. they are unstable on the convective time–scale ($t/\tau \sim 1$) at high Reynolds numbers (e.g. see figure 11 in Nagarajan et al 2003).

1.7 Verification and Validation

The combustor group draws on the strong culture and history of turbulent flow code verification and validation of the Center for Turbulence Research. In addition to using many of the standard CFD code verification methods (e.g. analytic and canonical flows, grid and time step convergence studies, Galilean invariance, primary conservation), our group carefully analyzes the discrete errors associated with kinetic energy conservation. To accurately predict turbulent flows, schemes are required that minimize the discrete non-conservation of kinetic energy (in the inviscid limit), so these errors are carefully characterized and confirmed in CDP [23].

For the last year, the development of CDP has been managed using the CVS version control software. As the number of contributors to the code has increased, this has allowed us to manage the steady expansion of the source distribution in a clear, modular way. In addition, the code is designed so that users can implement their own capabilities entirely in "hook" routines. Hook routines are full fortran 90 subroutines that encapsulate the case-specific information associated with a simulation, and are required to run the final CDP executable. These routines pass control to the user at critical points in the code execution, and expose all the CDP capability and data structures to the user. As a result, most users do not need to touch the source code to implement their desired models or capabilities. This programming model naturally supports the use of a common source, minimizing the tendency for users to have their own customized "versions" of the code (apart form their customized hook routines). Almost without exception, this development model is used to implement and verify new capabilities in CDP, which are then later incorporated into the source if deemed beneficial to the general pool of users.

To illustrate this "hook"-based model for development, consider the following simplified main control loop from CDP's fractional step incompressible flow solver:

```
type(ifp_t) :: ifp ! incompressible flow partition
!...
call init_ifp(ifp)
call initial_hook(ifp)
done = done_cdp()
do while (.NOT.done)
   time = time + dt
   call solve_scalars(ifp)
   call solve_u(ifp)
   call solve_p_and_correct_u(ifp)
   call temporal_hook(ifp)
   done = done_cdp()
end do
call final_hook(ifp)
!...
```

The user is initially passed control in the routine **initial_hook()** were they can initialize any capability specific to their case, register their own variables, read in any input files required, etc. In the example above, the user is also passed control at the end of each time step – **temporal_hook()** – where they can implement their own processing of the simulation, and at the end of the simulation – **final_hook()** – where they can write their own results. The real version of CDP is only slightly more complex than this example, and all

hooks required by the code are documented in the CDP User's Manual, a dynamic document managed in the same cvs tree as the CDP source. Additionally, smart makefiles provide template (basically empty) versions of all required hooks at compile time if they have not already been provided by the user. This ensures the backward compatibility of the code as new hook routines are added/exposed, and also allows a user to rapidly initiate new simulations.

The remaining subsections present some specific examples of solution verification and validation using the CDP code, along with a discussion of our group's work on flow visualization as a component of V&V.

1.7.1 Staggered Jets in Crossflow

The problem of opposed jets discharging normally into a plane channel flow has been investigated previously by our group [7]. This case represents a geometric simplification of the more complex configuration present in the dilution zone of a real combustor, where relatively cold air jets penetrate and mix with hot combustion products to achieve an outlet temperature distribution acceptable to the turbine.

When the pressure gradient reconstruction was modified from the least-squares formulation described in [23] to a conservative Green-Gauss reconstruction on a relatively coarse grid (0.4M control volumes), we found the mean velocity magnitude at the first measurement station dropped relative to the experimental data. Previously, using the least squares reconstruction, agreement had been significantly better. Figure 1.48 compares these results.

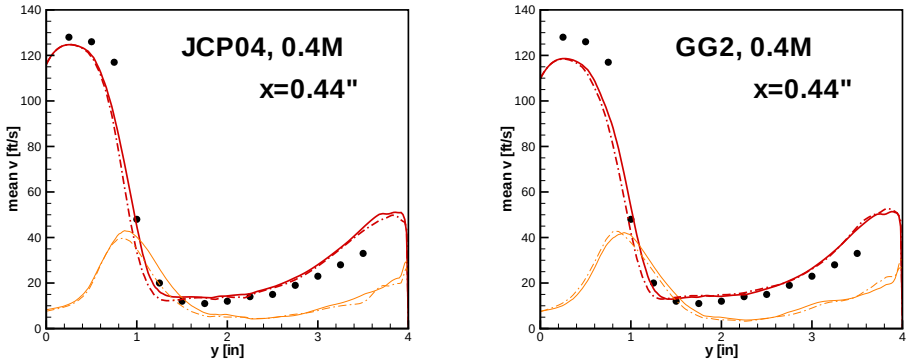


Figure 1.48: Effect of pressure reconstruction on computed means at the first measurement station for staggered jets in crossflow: relatively coarse grid of 0.4M control volumes. The left result (JCP04) uses the least-squares pressure reconstruction reported in Mahesh *et al.* (2004). The right result (GG2) uses a Green-Gauss reconstruction. The two red lines in each plot (solid and dashed) correspond to the mean velocity magnitude at the two lower holes present in the simulation – their similarity in the same plot is an indication of the adequacy of the sample size for the statistics. The lighter orange curves are the rms velocity magnitudes.

In both cases, the jet mass flux is identical. The discrepancy in mean velocity at the centerline is due to a difference in the 3-dimensional flow distribution through the hole. Figure 1.49 compares this mean distribution through the hole and also shows the relatively coarse unstructured grid associated with the 0.4M simulations.

In an attempt to understand this discrepancy, a grid refinement study was undertaken. Simulations were made on grids of 3M and 13M control volumes. Figure 1.50 presents the means at the first measurement station from these finer simulations. Previously, simulations of this problem by our group had only been performed on grids of up to 2M control volumes.

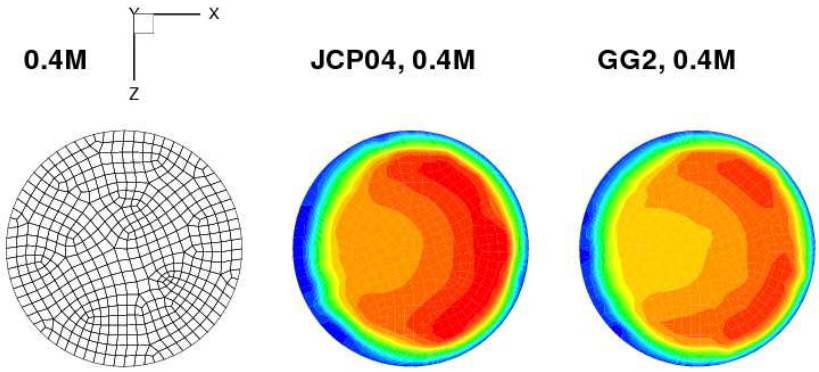


Figure 1.49: Comparison of mean vertical velocity magnitude in lower hole for the two pressure reconstructions.

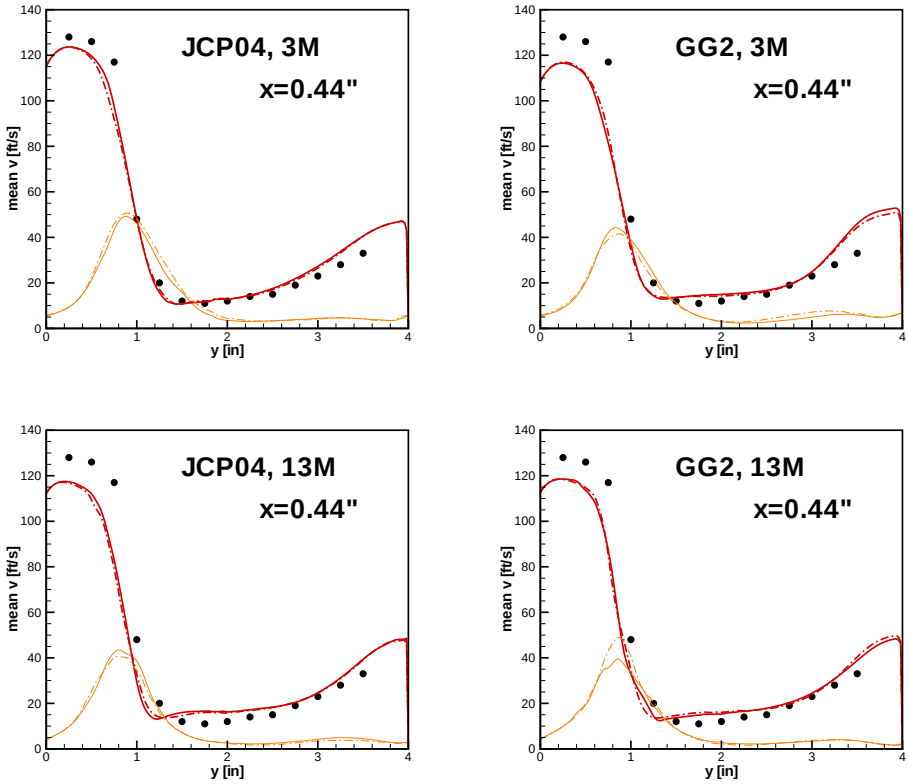


Figure 1.50: Effect of pressure reconstruction on computed means at the first measurement station for staggered jets in crossflow: medium and fine grids of 3M and 13M control volumes respectively.

Discussion

Even at 3M control volumes, there remains a significant difference between the two results, with the JCP04 reconstruction matching the experimental data better. At 13M control volumes, however, both reconstructions are reproducing approximately the same results, and both are significantly below the experimental data. It appears that the lower resolution calculations with the JCP04 reconstruction represents an example of getting the "right" result for the "wrong" reason. Figure 1.51 compares mean distributions in the vertical plane through the lower hole for the coarsest and finest simulations. The lack of smoothness downstream of the jet certainly indicates that more averaging time is required for downstream results. At the hole centerline, however, the statistics converge rapidly, and more averaging time is not expected to affect these conclusions.

It remains to answer the question why the apparently grid-converged simulation result does not match the experimental data. This is a subject of ongoing investigation in our group, and we are pursuing several possible reasons, including:

1. Inlet turbulence: the simulations were conducted with constant velocity inflow conditions. While turbulence levels in the experiment were reported low, they were certainly non-zero, so some inlet

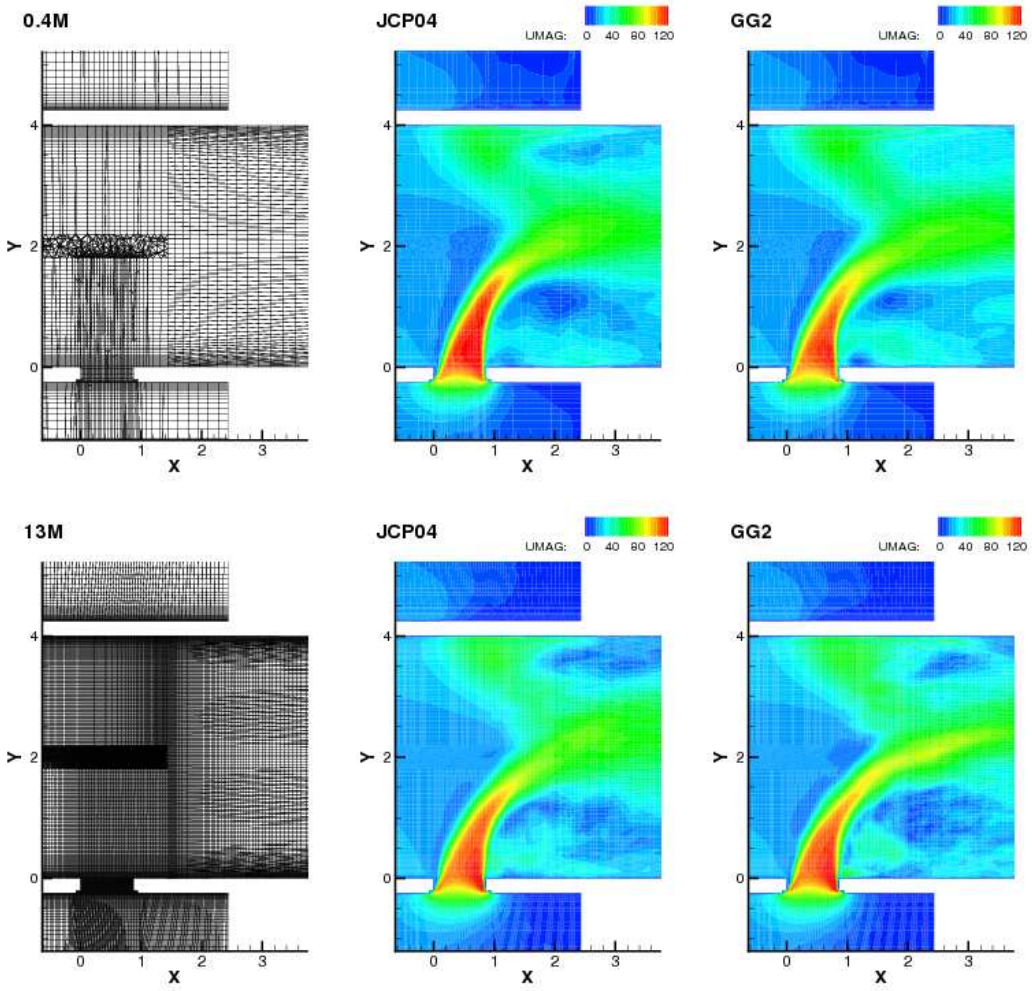


Figure 1.51: Effect of pressure reconstruction on computed means for staggered jets in crossflow. Top cases, from left to right, show section through coarse grid (400,000 control volumes), and results for two different pressure reconstructions. Bottom cases show same reconstructions applied to a 13 million control volume grid. Contour levels are identical in all plots.

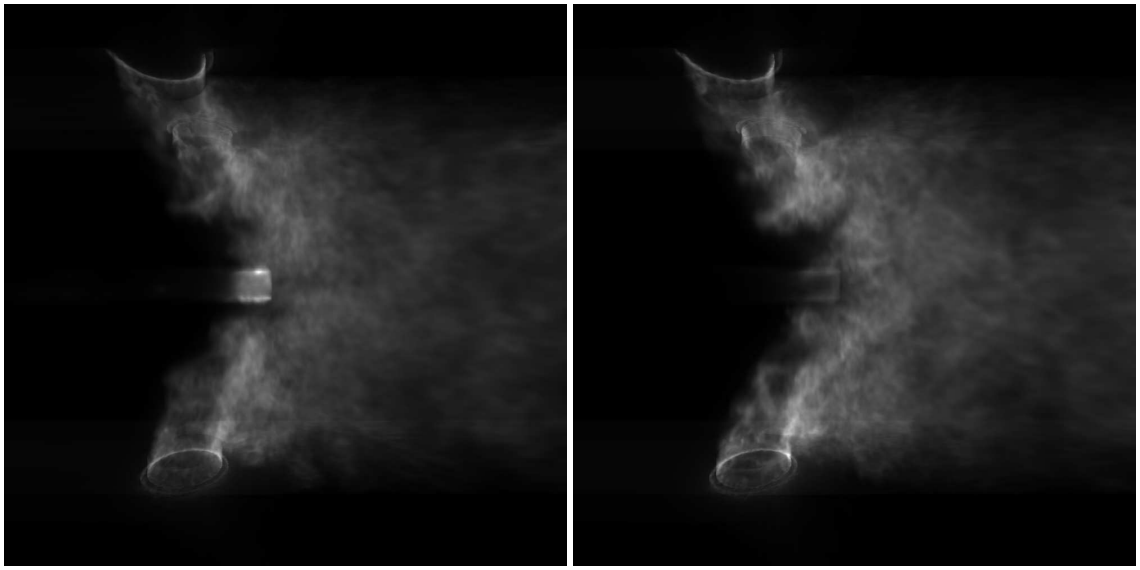


Figure 1.52: Volume visualization of the 13 M control volume jets in crossflow simulation with intensity related linearly to the instantaneous vorticity magnitude. Top: JCP04, Bottom: GG2 pressure reconstructions.

turbulence will be incorporated into the inlet profile to test the sensitivity of simulation results to this factor,

2. Proximity of outlet boundary: For confined swirling flows, the proximity and nature of the outlet boundary is known to have a significant effect on the character of upstream recirculation zones [40]. A similar effect may be causing the present discrepancy, because the staggering of the jets introduces mean swirl,

3. Three-dimensional effects: The assumptions of spanwise periodicity made in our simulations is only approximately true. Experiments were performed using a test rig approximately seven holes wide with end walls.

While the first and even second order statistics are very similar for the two reconstructions on the 13 M control volume grid, the least-squares reconstruction was found to increase the rms velocity magnitude in regions of grid transition (from hex to prism or tetrahedral control volumes). This is illustrated qualitatively by the volume visualizations presented in figure 1.52, where the intensity is related to the instantaneous vorticity magnitude. Due to the spanwise staggering of the jets, a rectangular region of unstructured grid is required to join the hex-dominant grids extruded off the top and bottom walls. The transitions associated with this region are clearly evident as a numerical source of vorticity in both simulations, although much more so in the case of the least-squares reconstruction. These volume visualizations are described in more detail in a subsequent section.

1.7.2 Calculation of flow in a jet engine fuel nozzle rig

Injector nozzle produced swirling flow is important for flame stabilization in jet engine combustors. Inclusion of nozzle geometry in combustor numerical simulation is desirable since the swirling from the nozzle

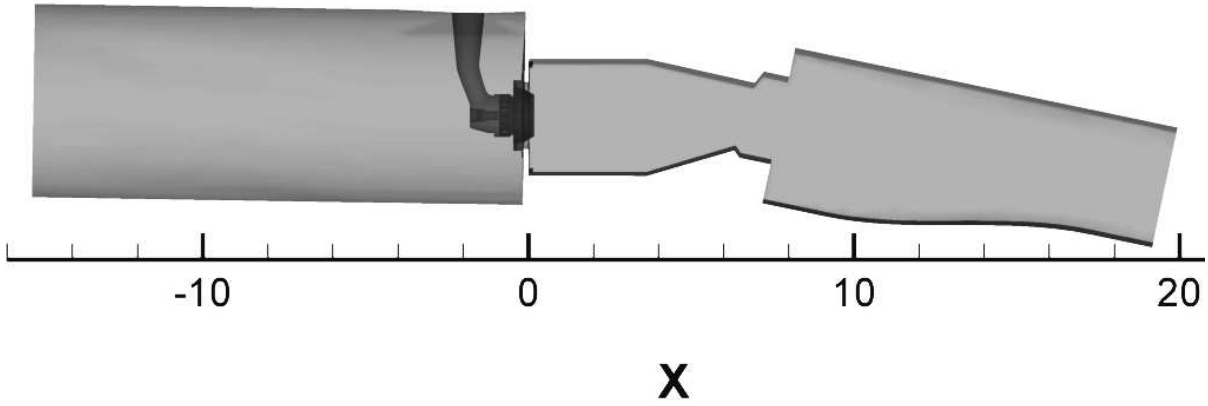


Figure 1.53: Geometry of the fuel nozzle rig.

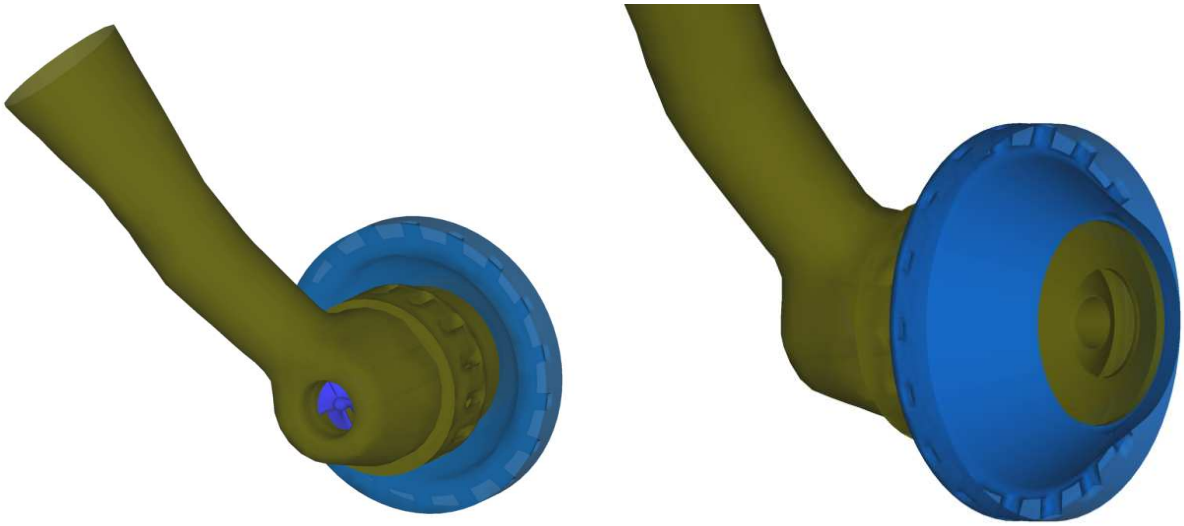


Figure 1.54: Overall view of the fuel nozzle.

is difficult to emulate faithfully using indirect means such as that developed by Pierce & Moin (1998). Brankovic *et al* (2000) reported comparison of laser velocimetry measurements and unstructured Reynolds-averaged predictions of incompressible swirling flow in a jet engine fuel nozzle rig. They found that quantitative accuracy was not achieved by the calculation, though their predicted mass flow splits showed that trends of flow distribution into the various nozzle intakes were correctly predicted. Other recent calculations of the swirling flow in jet engine combustor with full injector configuration include the work of Grinstein *et al* (2002) and Roux *et al* (2004).

As a validation test case, CDP was used to compute the incompressible flow in a fuel nozzle rig. The geometry is substantially similar to that of Brankovic *et al* (2000), see figure 1.53 and figure 1.54, but the present geometry does not have purge hole and associated intake. At the far upstream uniform inlet velocity $U_{in} = 29.134$ in/sec was prescribed. Reynolds number based on U_{in} and unit length is 1246. Except for the inlet and outlet all the other boundary surfaces of the geometry are solid wall on which no-slip conditions were applied. Calculations with three different mesh sizes ranging from 2 to 6 million cells were used and they yield similar results. Maximum CFL number was set at 4.5 and the corresponding time step is

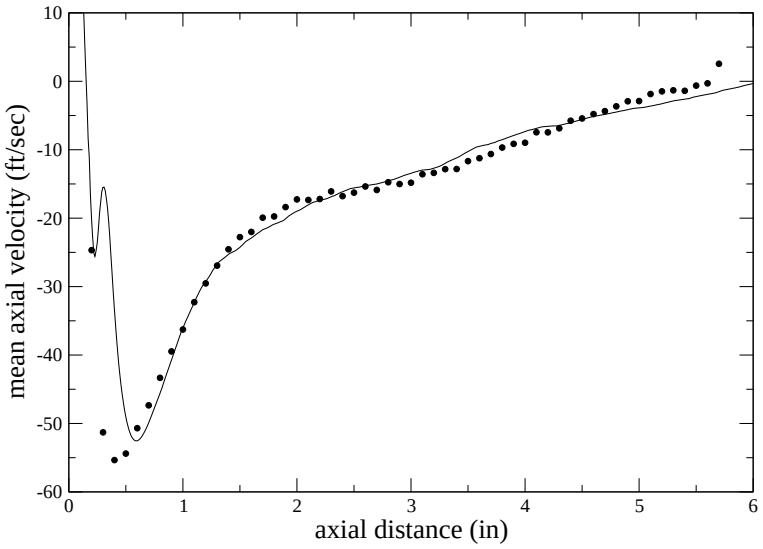


Figure 1.55: Mean axial velocity along the injector geometric centerline. • experiment; solid line: present LES.

approximately 7.0×10^{-7} seconds. It was found 2×10^5 time steps were needed to converge the mean flow statistics. Velocity statistics presented in the next section were converted to ft/sec following the convention used in the experiments.

Results

We found that the mean centerline axial velocity is a good indicator for monitoring convergence of LES solutions in the present rig swirling flow. The predicted mean centerline axial velocity in figure 1.55 is seen to agree with experiments over a large portion of the axial distance with the exception of the range $0.2 < x < 0.6$. The comparison is significantly better than that shown in figure 12 of Brankovic *et al* (2000). The peak at $x \approx 0.35$ was found to be very persistent through many test simulations with various mesh density and composition (tetrahedral versus hexhedral), and is likely a result of the interactions among the three swirling jet streams produced by the nozzle configuration as in figure 1.54. This point is reinforced through our test simulation in which the outer guide inlet was sealed off. Interaction of the two remaining inlet swiring jets does not result in the peak as found in the original three-jet case. The calculated mean axial and radial velocities are compared with experimental data in Figure 1.56. The under prediction of the negative peak at $x = 0.4$ is directly related to the discrepancy shown in figure 1.55. The swirling vortex is generated near the nozzle as a result of the interaction of the streams. This is visualized in figure 1.57 using contour surfaces of the negative second eigenvalue of the rate of deformation tensor.

1.7.3 Flame visualization

Turbulent flames typically exhibit highly complex topologies. Using advanced LES combustion methods, a wide range of topology scales can be resolved by numerical simulations, making the detailed analysis of the instantaneous turbulent flame structure difficult. Using standard tools, like two-dimensional slices through

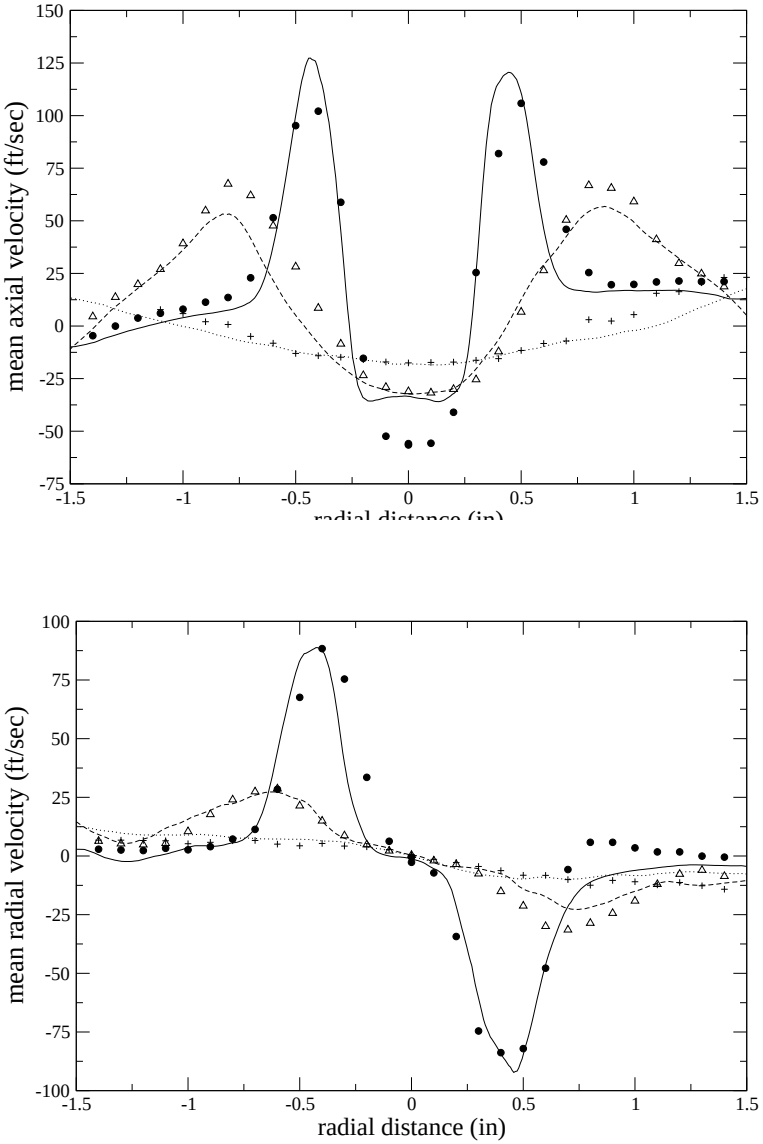


Figure 1.56: Upper: mean axial velocity; lower: mean radial velocity. • solid line $x = 0.4$; $\triangle$ dashed line $x = 1.1$; $+$ dotted line $x = 2.1$. Symbols are from experiments and lines are present LES.

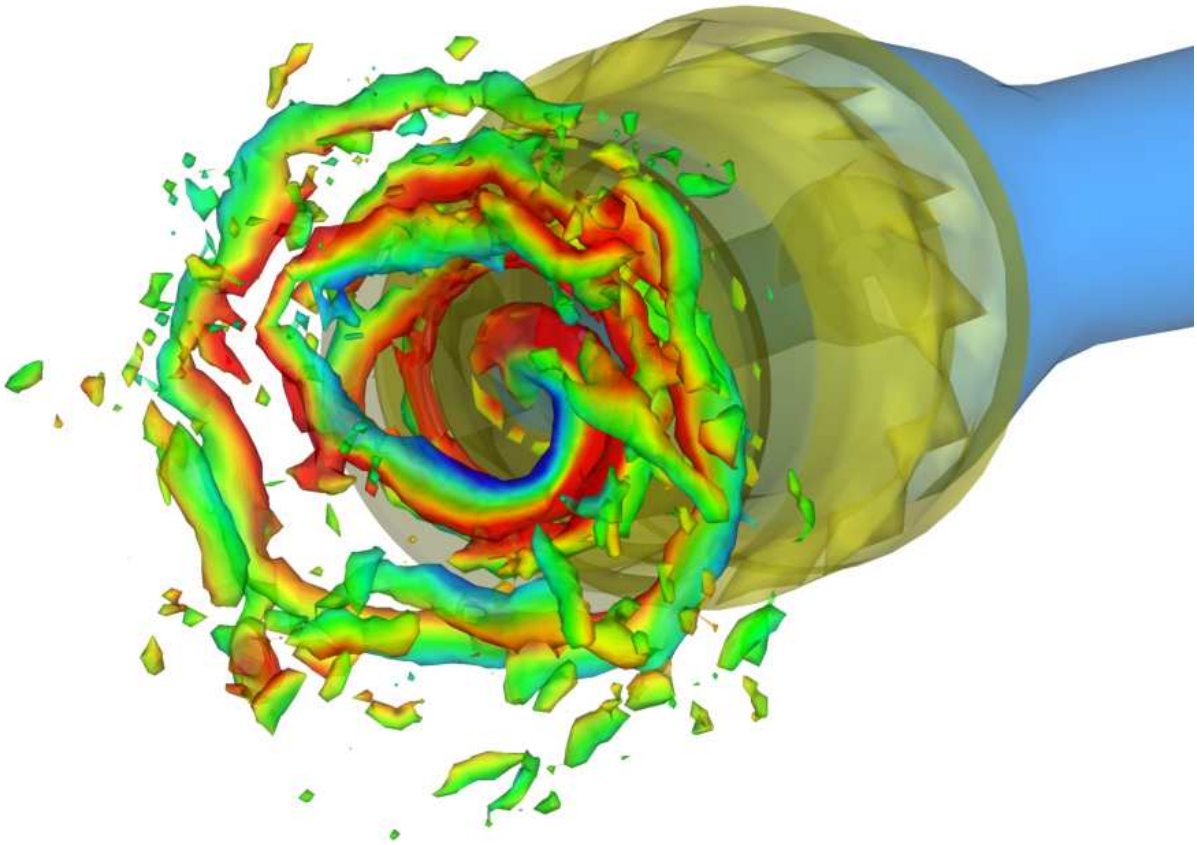


Figure 1.57: Visualization of swirling vortex near the injector.

the three-dimensional data set or iso-surface representations of scalar values, typically make it hard to fully grasp at once the complex structure of turbulent combustion that a simple high speed shutter photograph of the real flame might convey. Advanced visualization aids address this problem. However, they are not widely available and thus hard to use in every day analysis.

To overcome this problem, a tool has been developed that allows for the realistic optical visualization of turbulent flames using volume rendering. Volume rendering of flames in itself is not new, see for example [31]. However, the combination with advanced combustion models to make full use of the available numerical data for visualization purposes has not been achieved to date.

Governing equations

In technical flames, typically only a small number n_Y of species actually emit light in the visual range, for example only CH, C_2, and soot emit noticeable levels of light in hydro-carbon flames. Assuming black body behavior, the intensity $I(\lambda)$ of light emitted at a wavelength λ from a given spatial coordinate $\boldsymbol{x}$ can be approximated by

$$I(\lambda, \boldsymbol{x}) = \sum_{i=1}^{n_Y} \frac{w_i(\lambda)}{\iota_i} \widetilde{Y}_i(\boldsymbol{x}) \frac{2\pi h c^2}{\lambda^5 \left(e^{\frac{hc}{\lambda k \widetilde{T}(x)}} - 1 \right)}, \tag{1.57}$$

with w_i and ι_i weighting factors, h the Planck constant, c the speed of light, k the Boltzmann constant, $\widetilde{Y}_i$ the Favre filtered mass fraction of species i, and $\widetilde{T}$ the Favre filtered temperature. The weighting factors $w_i(\lambda)$ determine the hue of the emitted light from a given species and can be derived from visual observations of flames summarized by Gaydon [12],

$$\begin{aligned} w_{\mathrm{CH}}(438\,\mathrm{nm}) &= 0.2\,, \quad w_{\mathrm{CH}}(474\,\mathrm{nm}) = 0.9\,, \quad w_{\mathrm{CH}}(517\,\mathrm{nm}) = 1.0\,, \\ w_{\mathrm{CH}}(564\,\mathrm{nm}) &= 0.8\,, \quad w_{\mathrm{CH}}(619\,\mathrm{nm}) = 0.3\,, \\ w_{\mathrm{C_2}}(387\,\mathrm{nm}) &= 0.5\,, \quad w_{\mathrm{C_2}}(432\,\mathrm{nm}) = 1.0\,, \quad w_{\mathrm{C_2}}(494\,\mathrm{nm}) = 0.2\,, \\ & w_{\mathrm{soot}}(380 - 780\,\mathrm{nm}) = 1.0\,, \end{aligned} \tag{1.58}$$

and $w_i(\lambda) = 0$ for all other wavelengths. Since these factors have meaning only relative to the weighting factors of the same species, an additional factor ι_i is used to weight the species amongst each other. Here, ι_i is simply the integral over all visible wavelengths of the Planck function evaluated at a given species reference temperature $T_{i,\mathrm{ref}}$,

$$\iota_i = \int_{380\,\mathrm{nm}}^{780\,\mathrm{nm}} \frac{2\pi h c^2}{\lambda^5 \left(e^{\frac{hc}{\lambda k T_{i,\mathrm{ref}}(x)}} - 1 \right)} \, d\lambda\,, \tag{1.59}$$

and

$$T_{\mathrm{CH,ref}} = 2500\,\mathrm{K}\,, \qquad T_{\mathrm{C_2,ref}} = 3000\,\mathrm{K}\,, \qquad T_{\mathrm{soot,ref}} = 2300\,\mathrm{K}\,. \tag{1.60}$$

If noticeable levels of soot are present in the flame, absorption effects become important. However, since the numerical flame data visualized below does not contain any soot, absorption effects are not considered in the following.

Equation (1.57) is solved using 81 discreet, $5\,\mathrm{nm}$ wide wavelength bins covering the whole visual range from $380\,\mathrm{nm}$ to $780\,\mathrm{nm}$. The chromaticity associated with a given intensity spectrum $I(\lambda, \boldsymbol{x})$ can then be

determined using $5\,\mathrm{nm}$ color matching functions to calculate the CIE XYZ color space coordinates [5, 59], resulting in $\mathrm{XYZ}(\boldsymbol{x})$.

The chromaticity of a given picture pixel coordinate $\boldsymbol{\xi}$ can then be calculated by integrating $\mathrm{XYZ}(\boldsymbol{x})$ along a ray s going from the observer through $\boldsymbol{\xi}$ and through the computational domain,

$$\mathrm{XYZ}(\boldsymbol{\xi}) = \int \mathrm{XYZ}(\boldsymbol{x}(s))ds\,. \tag{1.61}$$

In a final step, the XYZ color coordinates are transformed into RGB values and a gamma correction is applied.

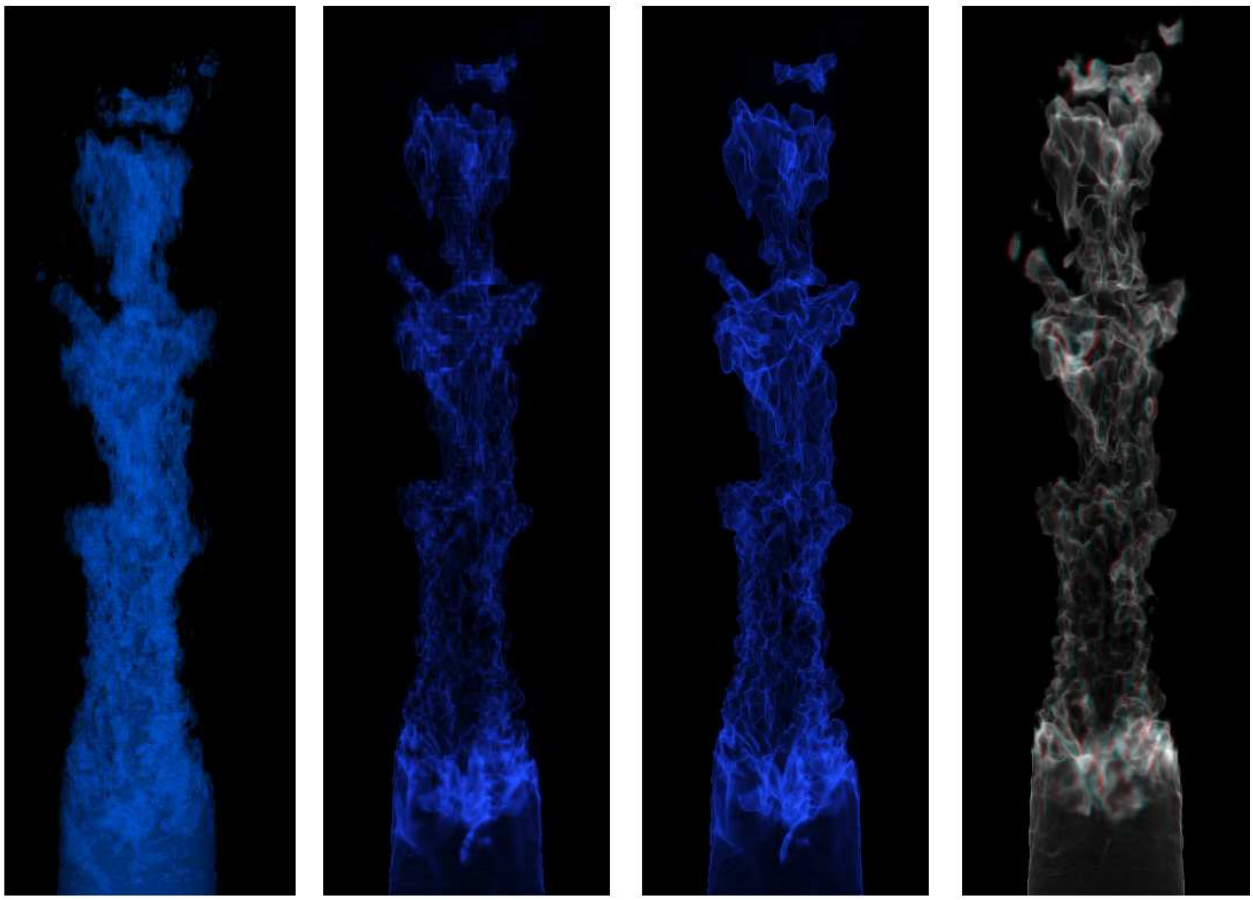


Figure 1.58: Visualization of a bluff body flame. Panel (a): CH iso-surface visualization, panel (b): volume rendering using nodal $\widetilde{Y}_i^{ijk}$, panel (c): volume rendering using subgrid resolution, panel (d): three dimensional volume rendering.

Figures 1.58 and 1.59 show the bluff body flame discussed in section 1.4.2 using four different visualization methods. Panel (a) shows the result of a typical CH iso-surface visualization, generated by a commercial visualization package, while panels (b), (c), and (d) depict the visualization results obtained using the volume rendering method described above.

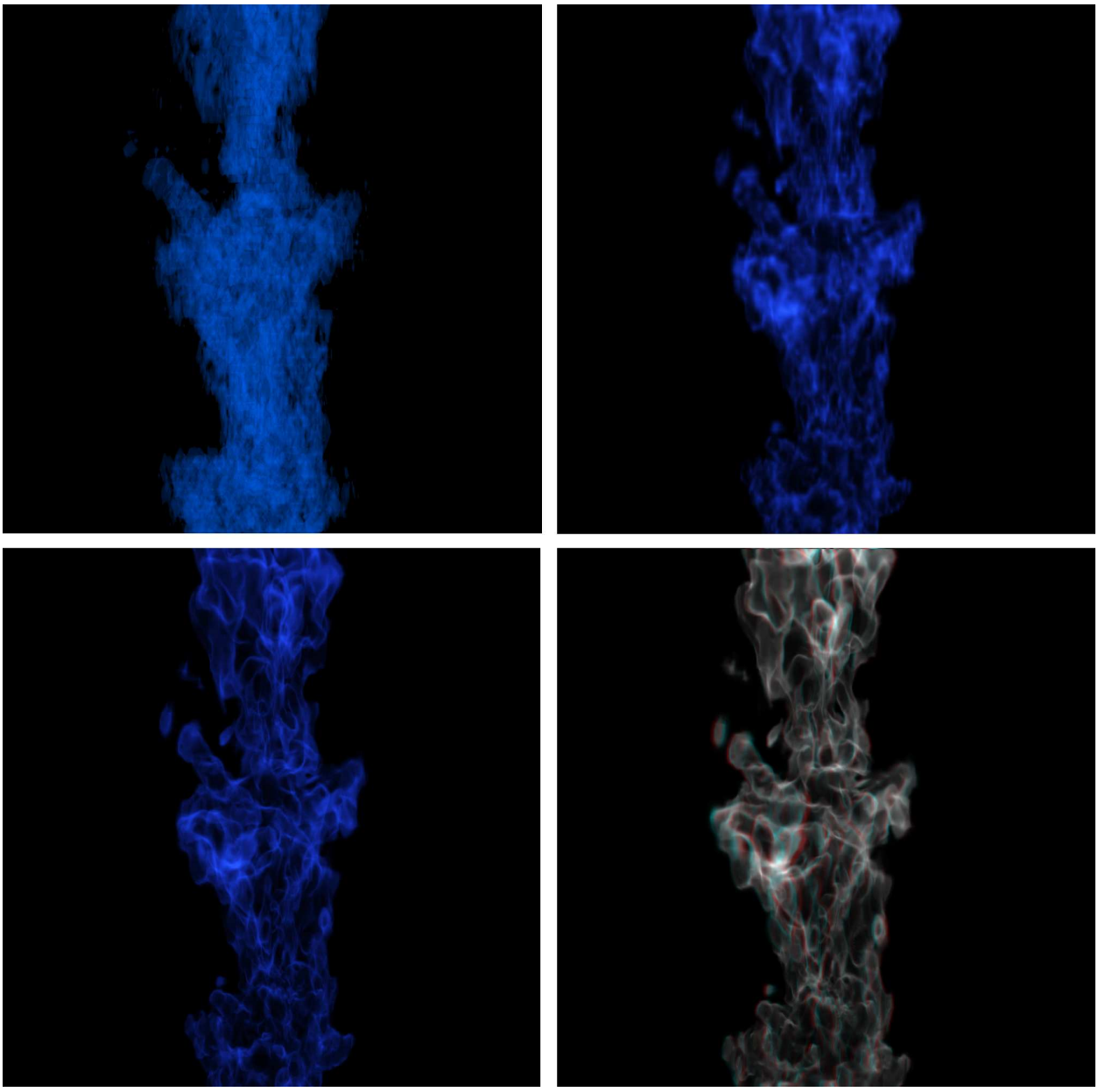


Figure 1.59: Zoom into a bluff body flame. Panel (a): CH iso-surface visualization, panel (b): volume rendering using nodal $\widetilde{Y}_i^{ijk}$, panel (c): volume rendering using subgrid resolution, panel (d): three dimensional volume rendering.

While some limited information about the structure of the flame can be discerned from the iso-surface visualization, panel (a), hardly any three dimensional volume information is conveyed in the image. The panels generated by the volume rendering method, on the other hand, clearly conveys more information about the detailed structure of the flame.

The difference between panel (b) and panel (c) lies in how the Favre filtered mass fraction $\widetilde{Y}_i$ at the ray location $\boldsymbol{x}(s)$ is calculated. In panel (b), $\widetilde{Y}_i(\boldsymbol{x}(s))$ is interpolated using only nodal data of the Favre filtered mass fraction $\widetilde{Y}_i^{ijk}$,

$$\widetilde{Y}_i(\boldsymbol{x}(s)) = f(\widetilde{Y}_i^{ijk})\,, \tag{1.62}$$

where f is an interpolation function. $\widetilde{Y}_i^{ijk}$ is either directly provided by the flow solver, or it can be computed at each node using flamelet lookup tables,

$$\widetilde{Y}_i^{ijk} = \widetilde{Y}_i(\widetilde{Z}^{ijk}, \widetilde{Z''^2}^{ijk}, \widetilde{\chi}^{ijk})\,, \tag{1.63}$$

where $\widetilde{Z}^{ijk}$ is the nodal Favre filtered mixture fraction, $\widetilde{Z''^2}^{ijk}$ the mixture fraction variance, and $\widetilde{\chi}$ the scalar dissipation rate. While this approach already gives markedly improved results as compared to the iso-surface visualization, panel (a), the above approach does not make full use of the flamelet approach.

In non-sooting flames, typically only radicals emit light. Radicals, however, occur only in very thin layers close to the stoichiometric mixture fraction. These layers are typically much thinner than the available grid resolution. Thus, using only the nodal mass fraction $\widetilde{Y}_i^{ijk}$, these subgrid structures cannot be resolved. Using the flamelet approach, however, it is possible to reconstruct, at least in part, the fine scale radical structures in each cell.

Since $\widetilde{Z}^{ijk}$, $\widetilde{Z''^2}^{ijk}$, and rate $\widetilde{\chi}^{ijk}$ are typically well resolved, first interpolating these quantities to the ray location $\boldsymbol{x}(s)$ and then using the flamelet lookup tables,

$$\widetilde{Y}_i(\boldsymbol{x}(s)) = \widetilde{Y}_i\left(f(\widetilde{Z}^{ijk}), f(\widetilde{Z''^2}^{ijk}), f(\widetilde{\chi}^{ijk})\right) \tag{1.64}$$

reconstructs the subgrid radical structure, thus making it possible to render the thin radical structures, resulting in the improved results of panel (c) as compared to panel (b).

It should be noted, that for finite volume data, the above approach simply removes the effect of the volume averaging inherent in the nodal data, whereas for finite difference data, the above approach provides a consistent, higher order reconstruction of $\widetilde{Y}_i$. The spatial filtering introduced in the LES approach, however, is not removed. Thus, to visualize the instantaneous unfiltered flame structure that can be seen by high speed, high resolution photography, the underlying numerical data has to be a DNS.

Finally, panel (d), when viewed through red/cyan filter glasses, shows a three-dimensional view of the flame structure. This image is generated by superimposing two subgrid scale rendered images, one red, the other cyan, whose observers are placed an eye-distance apart. The three dimensional structure of the turbulent bluff body flame can then be directly analyzed.

Bibliography

[1] APTE, S. V., GOROKHOVSKI, M. & MOIN, P. 2003a LES of atomizing spray with stochastic modeling of secondary breakup *Int. J. Mult. Flow* **29**, 1503-1522.

[2] BIJL, H. & WESSELING, P. 1998 A unified method for computing incompressible and compressible flows in boundary-fitted coordinates *J. Comput. Phys.* **141:** 153–173.

[3] BOWMAN, C. T., HANSON, R. K., DAVIDSON, D. F., *et al.*, Gri-mech 2.11,

[4] BRANKOVIC, A., MCKINNEY, R., QUYANG, H., PORTER, L., KENNEDY, J., MADABHUSHI, R. & COLKET, M. 2000, Comparison of measurements and predictions of flow in a gas turbine engine fuel nozzle, *AIAA-2000-0331*.

[5] BRUTON, D. 2004 Color science. http://www.cox-internet.com/ast305/color.html

[6] CHOI, H.G., & JOESPH, D.D. 2001 Fluidization by lift of 300 circular particles in plane Poisuille flow by direct numerical simulation. *J. Fluid. Mech.* **438**, 101-128.

[7] CITS annual technical report-2003, *Center for Integrated Turbulent Simulations, Stanford University*.

[8] COLUCCI, P. J. , JABERI, F. A., GIVI, P. 1998 Filtered density function for large eddy simulation of turbulent reacting flows, *Phys. Fluids* 10 (2) 499–515.

[9] DALLY, B. B., MASRI, A. R., BARLOW, R. S., FIETCHNER, G. J. 1998 Instantaneous and mean compositional structure of bluff-body stabilized nonpremixed flames *Comb. Flame* 114: 119–148.

[10] DIEZ, J.A., AND KONDIC, L. 2002 Computing three-dimensional thin film flows including contact lines. *J. Comput. Phys.*, 183:274–306.

[11] HARRIS S.J.,FRENKLACH M. 1987 Aerosol dynamics modeling using the method of moments *J. Colloid Interface Sci.* 252.

[12] GAYDON, A. G. 1958 *The Spectroscopy of Flames*. John Wiley & Sons.

[13] DE GENNES., P. G.1985 Wetting: Statistics and dynamics. *Rev. Mod. Phys.*, 57:827–863

[14] GHOSAL, S., LUND, T. S., MOIN, P. & AKSELVOLL, K. 1995 A dynamic localization model for large-eddy simulation of turbulent flow. *J. Fluid Mech.* **286**, 229-255.

[15] GIDASPOW, D. 1994 Multiphase flow and fluidization continuum and kinetic descriptions. Academic Press, Boston, MA.

[16] GRAVEMEIER, V. 2004 Scale-separating operators for variational multiscale large eddy simulation of turbulent flows. *Preprint*.

[17] GRINSTEIN, F.F., YOUNG, T.R., GUTMARK, E.J., LI, G., HSIAO, G. & MONGIA, H.C. 2002, Flow dynamics in a swirl combustor, *Journal of Turbulence*, **3**, article 30.

[18] VAN DER HEUL, D.R., VUIK, C. & WESSELING, P. 2002, A conservative pressure-correction method for the Euler and ideal MHD equations at all speed. *Intnl. J. Num. Methods Fluids*, **40:** 521–529.

[19] IHME, M., CHA, C. M., PITSCH, H. 2004 Prediction of Local Extinction and Re-ignition Effects in Non-premixed Turbulent Combustion Using a Flamelet/Progress Variable Approach *Proc. Combust. Inst. 30* (in press)

[20] JABERI, F. A., COLUCCI,P. J., JAMES, S., GIVI, P., POPE, S. B. 1999 Filtered mass density function for large-eddy simulation of turbulent reacting flows, *J. Fluid Mech.* 401:85–121.

[21] JOSEPH, D.D., & LUNDGREN, T., 1990, Ensemble averaged and mixture theory equations for incompressible fluid-particle suspensions. *Int. J. Mult. Flow* **16**, 35-42.

[22] 1998 PITSCH, H. A C++ computer program for 0-d and 1-d laminar flame calculations, rWTH Aachen

[23] MAHESH, K., CONSTANTINESCU, G. & MOIN, P. 2004, A numerical method for large eddy simulation in complex geometries, *J. Comp. Phys.*, **197**, 215-240.

[24] APTE, S. V., MAHESH, K., MOIN, P., & OEFELEIN, J.C., 2003, Large-eddy simulation of swirling particle-laden flows in a coaxial-jet combustor. *Int. J. Mult. Flow* **29**, 1311-1331.

[25] MOSER, R. D., KIM, J. & MANSOUR, N. N. 1999 Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$. *Phys. Fluids* **11**, 943-945.

[26] MAUSS F. 1997 Entwicklung eines kinetischen Modells der Russbildung mit schneller Polymerisation,PhD thesis

[27] MENEVEAU, C., LUND, T. S. & CABOT, W. H. 1996 A Lagrangian dynamic subgrid scale model of turbulence *J. Fluid Mech.* **319**, 353-385.

[28] MOIN, P., SQUIRES, K., CABOT, W., AND LEE, S. 1991 A dynamic subgrid-scale model for compressible turbulence and scalar transport. *Phys. Fluids A* **3**, 2746-2757.

[29] MURADOGLU, M., JENNY, P., POPE, S. B., CAUGHEY, D. A. 1999 A consistent hybrid finite-volume/particle method for the PDF equations of turbulent reactive flows, *J. of Comp. Phys.* 154:342–371.

[30] NAGARAJAN, S., LELE, S.K. & FERZIGER, J. H. 2003, A robust high-order compact method for large eddy simulation. *J. Comput. Phys.* **191:** 392–419.

[31] NGUYEN, D., FEDKIW, R. & JENSEN, H. 2002 Physically based modeling and animation of fire. In *SIGGRAPH 2002*, volume ACM TOG 21, pages 721–728

[32] ORON, A., DAVIS, S. H., AND BANKOFF, S. G. 1997 Long-scale evolution of thin liquid films. *Rev. Modern Phys.*, 69(3):931–980

[33] PETERS, N. 1984 Laminar diffusion flamelet models in non-premixed turbulent combustion. *Prog. Energy Combust. Sci.* **10**, 319-339.

[34] PETERS N. LEUSDEN C.P. 2000 Experimental and numerical analysis of the influence of oxygen on soot formation in laminar counterflow flames of acetylene, Proc. Combust. Inst. 2619-2625.

[35] PETERS, N. 2000 Turbulent Combustion, Cambridge University Press

[36] PIERCE, C. D., MOIN, P. 1998 A dynamic model for subgrid-scale variance and dissipation rate of a conserved scalar. *Phys. Fluids* 10 (12): 3041-3044

[37] PIERCE, C. D. 2001 Progress-variable approach for large-eddy simulation of turbulence combustion, Ph.D. thesis, Stanford University

[38] PIERCE, C. D., MOIN, P. 2004 Progress-variable Approach for Large-Eddy Simulation of Non-Premixed Turbulent Combustion. *J. Fluid Mech.* 504: 72-97

[39] PIERCE, C. & MOIN, P. 2001 The progress variable approach for large eddy simulation of turbulent combustion. *TF–Report 80*, Flow Physics Division, Mechanical Engineering Department, Stanford University, Stanford, California.

[40] PIERCE, C. D., & MOIN, P. 1998 Large Eddy Simulation of a Confined Coaxial Jet with Swirl and Heat Release. *AIAA paper 98-2892*.

[41] PIERCE, C.D. & MOIN, P. 1998, Method for generating equilibrium swirling inflow conditions. *AIAA J.*, **36**, 1325-1327.

[42] PARMETIS http://www-users.cs.umn.edu/ karypis/metis/parmetis/

[43] POPE, S. B. 2000 Turbulent Flows, Cambridge University Press

[44] RAMAN, V., PITSCH, H., FOX, R. O. 2004 A consistent hybrid LES-FDF scheme for the simulation of turbulent reactive flows, *to be submitted to Comb. Flame*

[45] POZORSKI, J., APTE, S.V., & RAMAN, V. 2004 Filtered particle tracking for dispersed two-phase turbulent flows. *Proceedings of Summer Program*, Center for Turbulence Research, Stanford University, Stanford.

[46] POZORSKI, J. & MINIER, J. P. 1999 PDF modeling of dispersed two-phase turbulent flows. *Phys. Rev. E* **59**, 855–863.

[47] READE, W. C. & COLLINS, L. R. 2000 Effect of preferential concentration on turbulent collision rates. *Phys. Fluids* **12**, 2530–2540.

[48] ROGALLO, R. 1981, Numerical experiments in homogeneous turbulence. *NASA Technical Memorandum TM–81315*.

[49] ROUX, S., LARTIGUE, G., POINSOT, T., MEIER, U. & BERAT, C. 2004, Studies of mean and unsteady flow in a swirled combustor using experiments, acoustic analysis and large eddy simulations, submitted for publication.

[50] SNIDER, D. M., 2001, An incompressible three-dimensional multiphase particle-in-cell model for dense particulate flows. *J. Comput. Phys.* **170**, 523-549.

[51] SPADACCINI, L.J., OWEN, F.K., AND BOWMAN, C.T. 1976 Influence of aerodynamic phenomena on pollutant formation in combustion (Phase I. Gaseous Fuels). *U.S. Environmental Protection Agency Rep.* EPA-600/2-76-247a.

[52] SANKARAN, V. & MENON, S. 2002 LES of spray combustion in swirling flows. *J. Turbulence* **3**, art. no. 011.

[53] THOMAS, D.G. 1965 Transport characteristics of suspension: VIII. A note on the viscosity of Newtonian suspensions of uniform spherical particles. *J. Colliod. Sco.* **20**, 267-277.

[54] TROIAN, S. M., HERBOLZHEIMER, E., SAFRAN, S. A., AND JOANNY, J. F. 1989 Fingering instabilities of driven spreading films. *Europhys. Lett.*, 10:25–30

[55] THOMPSON, P.A. 1988, Compressible-fluid dynamics.

[56] VILLERMAUX, J. 1986 Micromixing phenomena in stirred reactors, in: Encyclopedia of Fluid Mechanics, Gulf, Houston, TX, Ch. 27.

[57] WALL, C., PIERCE, C.D. & MOIN P. 2002, A semi-implicit method for resolution of acoustic waves in low Mach number flows. *J. Comput. Phys.*, **181:** 545–563.

[58] WANG, Q. & SQUIRES, K. D. 1996 Large eddy simulation of particle-laden turbulent channel flow. *Phys. Fluids* **8**, 1207–1223.

[59] WYSZECKI, G. AND STILES, W. S. 1982 *Color Science: Concepts and Methods, Quantitative Data and Formula*. John Wiley & Sons

Chapter 2

Turbomachinery Simulations

The objective of the Turbomachinery Group is to develop state-of-the-art capabilities for the simulation of flows in the rotating components of jet engines (compressor, turbine, secondary flow system) using the Unsteady Reynolds-Averaged Navier-Stokes (URANS) equations as a flow model. Our measures of success are fundamentally based on the ability to decrease the solution times by significant factors (10 X to 20 X) in comparison with other existing software and to increase the modeling accuracy via the introduction of more advanced turbulence and transition models. Although our target application is in turbomachinery flows, the software that we have been developing is completely generic and applicable to a wide range of other unsteady flow applications. We try to make this point in this Chapter by showing a number of validation test cases in external aerodynamics and throughout a very wide flow speed range (from incompressible flow to supersonic flow).

The following sections detail the work conducted during the period of this past year in both algorithmic improvements and in the modeling work. We have chosen to highlight the time-spectral implementation and the use of preconditioning techniques in TFLO2000. Additional algorithmic work continues and the results will be discussed at a later time. In addition, this year we present a very detailed account of the efforts of our group in improved modeling for turbulence, adaptive wall functions, transition, and in studies which use LES to improve the modeling accuracy of URANS codes.

A large portion of the effort this year has gone to ensure that all of the studies that are being conducted in parallel eventually end up having an impact of the new implementation of our massively parallel flow solver, TFLO, which we now call TFLO2000. TFLO2000 has matured considerably over the course of the last year, to the point that we have decided to phase out the old TFLO solver and everyone in our group is now using TFLO2000. Substantial enhancements are planned for the coming year as well.

2.1 Algorithmic Improvements

This section presents our efforts in algorithms for the solution of time-accurate RANS flows. A major accomplishment of this year is the development and implementation in TFLO2000 of a time-spectral solution approach for the simulation of time-periodic flows. This is the first implementation of its kind in a complex, three-dimensional flow solver and has already shown the potential of 5 X speedups in comparison with the dual-time stepping time integration method that we have used in the past. Testing continues with compressors and turbines with much higher rpm; higher rpm leads to much longer time integrations to reach a

periodic steady state, and therefore allow even larger advantages for the time-spectral approach. In addition, we have completed the implementation and testing of the preconditioning squared approach (low-speed + block Jacobi preconditioning) in TFLO2000. This implementation is undergoing detailed testing and will enable TFLO2000 to span the range of Mach numbers from zero to supersonic within the same calculation. Work in our LU-SGS and Hybrid-ADI algorithms has also continued during the past year, but we have chosen not to explicitly discuss it in this report.

2.1.1 Time-Spectral Approach

Time dependent periodic calculations find a wide variety of applications including flutter analysis, analysis of flow around helicopter blades in forward flight and rotor-stator combinations in turbomachinery, where the last application is the goal of the turbomachinery group of the Stanford ASC project.

The typical way to solve periodic flow problems is to integrate the governing equations in time until the periodic state is reached. The time integration scheme normally used is the implicit second-order Backward Difference Formulation (BDF), in which the set of nonlinear equations for the new state is solved using the dual time stepping approach in combination with standard convergence acceleration techniques such as multigrid and local time stepping, see [1]. Typically 50 multigrid cycles are needed for every time step for an accurate solution of the URANS equations for turbomachinery problems.

For high-RPM compressors it takes at least a couple of revolutions before the periodic state is reached. Combined with the 50 multigrid cycles per time step, at least 50 time steps per blade passage and at least 40 blade per blade row this leads to more than 100,000 multigrid cycles and thus to excessive wall clock times even on the largest supercomputers available.

In this section the Time Spectral Method is presented to solve time-periodic unsteady problems. The Fourier method is used for time discretization, leading to spectral accuracy in time with increasing number of modes. When the formulas are transformed back to the physical domain, the time derivative appears as a high-order central difference formula coupling all the time levels in the period. Again dual time stepping and the standard convergence acceleration techniques are used as an iterative method to obtain the solution. The advantage of solving the equations in the time domain rather than the frequency domain is that the method can be implemented rather easily in an existing CFD code, whereas when the problem is solved in the frequency domain a completely new code must be developed involving complex arithmetic.

Firstly the general formulation is described, which is extended to problems with sector periodicity where the periodicity is less than than the entire revolution. Results will be shown for a pitching airfoil and the Aachen turbine case. Conclusions and future work will conclude this section.

General formulation of the Time-Spectral Method

The periodic nature of turbomachinery problems is not used at all when the governing equations are integrated in time until the periodic state is reached. If this information is taken into account, a Fourier representation in time can make it possible to achieve spectral accuracy.

The Navier-Stokes equations in integral form are given by

$$\int_{\Omega} \frac{\partial w}{\partial t}\, dV + \oint_{\partial\Omega} \vec{F} \cdot \vec{N}\, ds = 0, \tag{2.1}$$

where w are the conservative variables and $\vec{F}$ is the corresponding flux vector, which includes the effect of a moving control volume. The semi-discrete form of equation (2.1) is obtained by only discretizing the spatial part, which can be expressed as

$$V\frac{\partial w}{\partial t} + R(w) = 0. \tag{2.2}$$

Here the volume of the control volume is assumed to be constant in time. If this is not the case, e.g. an aero-elastic problem, the derivation below is still valid; only w must be replaced by Vw.

Suppose the time period is given by T and is divided into N time steps, $\Delta t = T/N$, and N is even. The discrete fourier transform of w is then given by

$$\hat{w}_k = \frac{1}{N}\sum_{n=0}^{N-1} w^n e^{-ikn\Delta t} \tag{2.3}$$

and its inverse transform by

$$w^n = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{w}_k e^{ikn\Delta t}. \tag{2.4}$$

Discretize the governing equations as a pseudo-spectral scheme,

$$VD_t w^n + R(w^n) = 0. \tag{2.5}$$

McMullen et.al. [2, 3]solved the time accurate equations (2.5) by transforming them into frequency domain. Alternatively, the Time Spectral Algorithm proposes to solve the governing equations in the time-domain, considerably gaining on the computational time required to transform back and forth to the frequency domain.

From equation (2.4), the time discretization operator D_t can be written as

$$D_t w^n = \frac{2\pi}{T}\sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} ik\hat{w}_k e^{ikn\Delta t}.$$

This summation, involving the fourier modes $\hat{w}_k$, can be rewritten in terms of the variables w in the time domain as

$$D_t w^n = \frac{2\pi}{T}\sum_{m=-\frac{N}{2}+1}^{\frac{N}{2}-1} d_m w^{n+m}, \tag{2.6}$$

where the coefficients are given by, see also [4]

$$d_m = \begin{cases} \frac{1}{2}(-1)^{m+1} cot(\frac{\pi m}{N}) & : \quad m \neq 0 \\ 0 & : \quad m = 0 \end{cases} \tag{2.7}$$

Note that $d_{-m} = -d_m$. Hence the operator D_t is a central difference operator connecting all the time levels, yielding an integrated space-time formulation which requires the simultaneous solution of the equations for all time levels.

A pseudo-time t^* is introduced like in the dual time stepping case, and the equations are time marched to a periodic steady-state

$$V\frac{\partial w^n}{\partial t^*} + VD_t w^n + R(w^n) = 0. \tag{2.8}$$

Stability

Equation (2.8) is typically solved with explicit multi-stage Runge-Kutta schemes. The addition of the time spectral time derivative must be taken into account in the definition of the pseudo-timestep Δt^*.

For simplicity, consider the one-dimensional quasi-linear version of equation (2.1)

$$V\frac{\partial w^n}{\partial t^*} + V\frac{2\pi}{T}\sum_{m=-\frac{N}{2}+1}^{\frac{N}{2}-1} d_m w^{n+m} + A\frac{\partial w^n}{\partial x} = 0,$$

where A is the Jacobian of the spatial flux operator.

A Von-Neuman analysis of these equations using a single spatial fourier mode and a single time fourier mode gives

$$V\frac{\partial \hat{w}}{\partial t^*} + V\sum_{m=1}^{\frac{N}{2}-1} P_m \hat{w} + \frac{Q}{\Delta x}\hat{w} = 0$$

where

$$P_m = i2d_m sin(km\Delta t)\frac{2\pi}{T}$$

is the spectral matrix corresponding to the physical time discretization, and with central differences for the spatial discretization, $Q = iAsin(w\Delta x)$.

Let $\overline{\lambda}$ denote the maximum eigenvalue of Q, and $\overline{\lambda}_r$, $\overline{\lambda}_i$ its real and imaginary parts. Then,

$$V\frac{\partial \check{w}}{\partial t^*} \leq -[\frac{\overline{\lambda}_r}{\Delta x} + (\frac{\overline{\lambda}_i}{\Delta x} + V\sum_{m=1}^{\frac{N}{2}-1}\frac{4\pi\,|d_m|}{T})i]\check{w}, \tag{2.9}$$

where the coefficients d_m are given by equation (2.7). Note that the source term due to the time-spectral term affects only the imaginary part of the eigenvalue associated with the time averaged solution.

Sector periodicity

In turbomachinery problems often only a part of the entire wheel is simulated to keep the problem size within acceptable limits. Consequently sector periodic boundary conditions must be used to obtain a well-posed problem. If the time spectral method is used to discretize the time derivatives for such a problem, Equations (2.6) and (2.7) are not valid anymore, because the momentum vectors are rotated between periodic boundary conditions. Below we show that it is still possible to use the time spectral method on only a part of the wheel if the coefficients of the time derivatives for vector quantities are modified.

Suppose the entire wheel is modeled using M time steps and the sector periodic subproblem with N time steps (N is even), where $M = PN$, see Figure 2.1. Here P is the number of periodic sections present in the entire wheel.

For the entire wheel the solution is certainly periodic, i.e.

$$w^{j+M} = w^j. \tag{2.10}$$

Here j is an arbitrary time index. Relation (2.10) holds for both scalar and vector quantities and therefore equations (2.6) and (2.7) are valid when the entire wheel is considered.

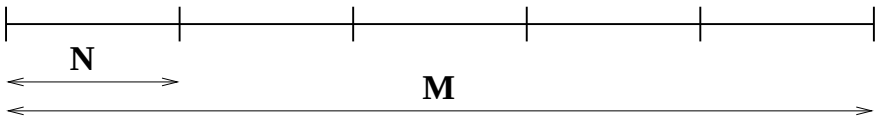


Figure 2.1: Sector periodic subproblem with N time steps and entire wheel with M time steps, where $M = PN$, P is an integer; for the case shown $P = 5$.

Splitting the sum in equation (2.6) in a part with positive and in a part with negative indices and introducing the superscript M to indicate that these coefficients are valid for the time interval T_M, the time to complete one rotation, results in

$$D_t w^l = \frac{2\pi}{T_M} \left\{ \sum_{m=1-\frac{M}{2}}^{-1} d_m^M w^{l+m} + \sum_{m=1}^{\frac{M}{2}-1} d_m^M w^{l+m} \right\}. \tag{2.11}$$

Using the periodicity of the solution, equation (2.10), and the periodicity of the coefficients d_m^M, $d_m^M = d_{m+M}^M$, the running index in the first sum in equation (2.11) can be incremented by M, i.e.

$$\sum_{m=1-\frac{M}{2}}^{-1} d_m^M w^{l+m} = \sum_{m=\frac{M}{2}+1}^{M-1} d_m^M w^{l+m}. \tag{2.12}$$

Combining Equations (2.11) and (2.12) and the fact that $d_{\frac{M}{2}}^M = 0$, this gives the following expression for the time derivative at point l

$$D_t w^l = \frac{2\pi}{T_M} \sum_{m=1}^{M-1} d_m^M w^{l+m}. \tag{2.13}$$

An alternative way of writing equation (2.13) is

$$D_t w^l = \frac{2\pi}{T_M} \left[\sum_{p=0}^{P-1} \sum_{n=1}^{N-1} d_{n+pN}^M w^{l+n+pN} \right] + \frac{2\pi}{T_M} \left[\sum_{p=1}^{P-1} d_{pN}^M w^{l+pN} \right]. \tag{2.14}$$

The second sum of Equation (2.14) can be incorporated in the double sum if the index n starts at 0 instead of 1. However, it is kept separate, because this term vanishes for scalar quantities.

Now assume that w is a vector and furthermore assume that point l belongs to a rotating part of the grid, i.e. point l moves in time. Then the following relation holds

$$w^{j+pN} = R^p w^j, \tag{2.15}$$

where R is the rotation matrix between the periodic sections. Note the the superscript p in R^p is an actual power, i.e. $R^p = R * R * \cdots * R$. Combining equations (2.14) and (2.15) results in

$$D_t w^l = \frac{2\pi}{T_M} \left[\sum_{p=0}^{P-1} \sum_{n=1}^{N-1} d_{n+pN}^M R^p w^{l+n} \right] + \frac{2\pi}{T_M} \left[\sum_{p=1}^{P-1} d_{pN}^M R^p w^l \right]. \tag{2.16}$$

It is clear that the superscript of the solution w does not depend anymore on the summation index p. Therefore the matrices d_n^N and e^N can be defined as

$$\begin{aligned} d_n^N &\overset{def}{=} \frac{1}{P}\sum_{p=0}^{P-1} d_{n+pN}^M R^p \\ e^N &\overset{def}{=} \frac{1}{P}\sum_{p=1}^{P-1} d_{pN}^M R^p \end{aligned} \tag{2.17}$$

Combining equations (2.16) and (2.17) and making use of the fact that $T_M = PT_N$, the time derivative for point l becomes

$$D_t w^l = \frac{2\pi}{T_N}\sum_{n=1}^{N-1} d_n^N w^{l+n} + \frac{2\pi}{T_N} e^N w^l. \tag{2.18}$$

Several things that can be noted. First $d_n^N \neq d_{n+N}^N$, second $d_{\frac{N}{2}}^N \neq 0$, third the time derivative at point l now depends on the state at point l via e^N and fourth the derivative of e.g. the x-component of the vector depends on the y and z components.

For completeness consider the scalar (or non-rotating) case, i.e. $R = I$. From the definition of d_m^M, equation (2.7), and e^N, equation (2.17) it is clear that for the scalar case $e^N = 0$. The definition of d_n^N simplifies to

$$\begin{aligned} \left(d_n^N\right)^{\text{scal}} &= \frac{1}{P}\sum_{p=0}^{P-1} d_{n+pN}^M \\ &= \frac{1}{P}\sum_{p=0}^{P-1} \frac{1}{2}\left(-1\right)^{n+pN+1} \cot\left(\frac{\pi(n+pN)}{M}\right) \end{aligned} \tag{2.19}$$

One would expect that this would be identical to the expression one would get if equation (2.13) is applied directly to the period N, i.e.

$$\left(d_n^N\right)^{\text{scal}} = \frac{1}{2}\left(-1\right)^n \cot\left(\frac{\pi n}{N}\right). \tag{2.20}$$

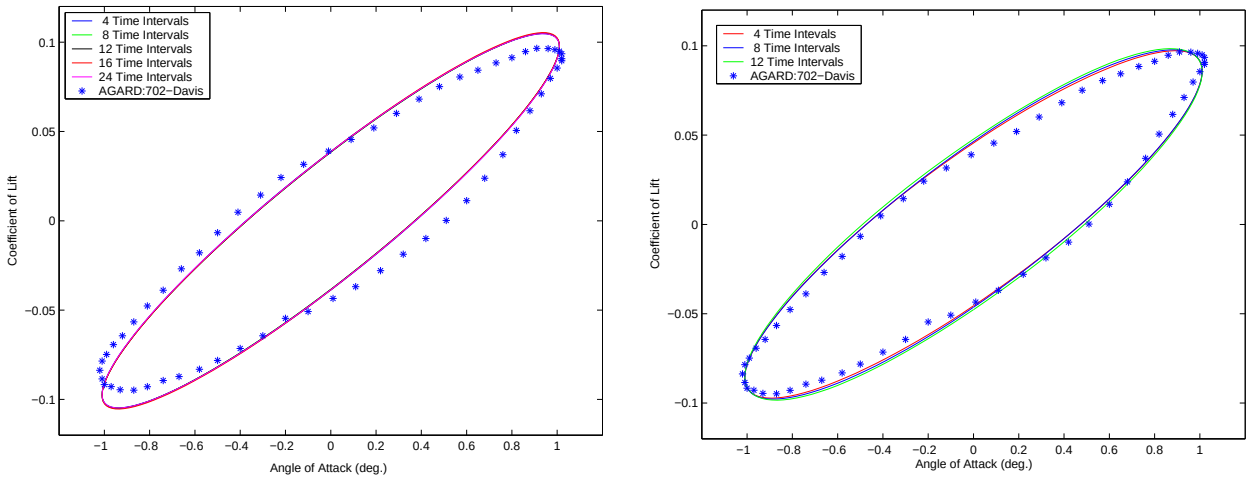
This is the case if

$$\sum_{p=0}^{P-1} \cot\left(\frac{\pi(n+pN)}{PN}\right) \overset{?}{=} P\cot\left(\frac{\pi n}{N}\right), \tag{2.21}$$

where use is made of $(-1)^n = (-1)^{n+pN}$, which is valid because N is even. It can be proved that expression (2.21) indeed holds.

M_∞	α_m, $deg.$	Re_c, $(million)$	α_0, $deg.$	k_c (red. freq.)
0.796	0	12.56	1.01	0.202

Table 2.1: Conditions for AGARD CT6 (pitching NACA 64A010) test case.



(a) Euler Calculations

(b) Navier-Stokes calculations

Figure 2.2: Comparison of C_l data with experimental results for the AGARD CT6 test case.

NACA 64A010 pitching airfoil

The oscillatory pitching airfoil AGARD CT6 (NACA 64A010) case, see [5], has been simulated in both inviscid and viscous modes. The flow conditions are summarized in table 2.1.

Euler calculations are carried out on a 160x32 O-mesh with 4,8,12,16 and 24 time intervals per pitching period of the airfoil. The coefficient of lift(C_l) as a function of angle of attack is provided in figure 2.2(a). A Navier-Stokes calculation on a 256x64 C-mesh with 4,8 and 12 time intervals shows a much better approximation to the experimental data as shown in figure 2.2(b). Note that for both cases little deviation is achieved with additional temporal resolution.

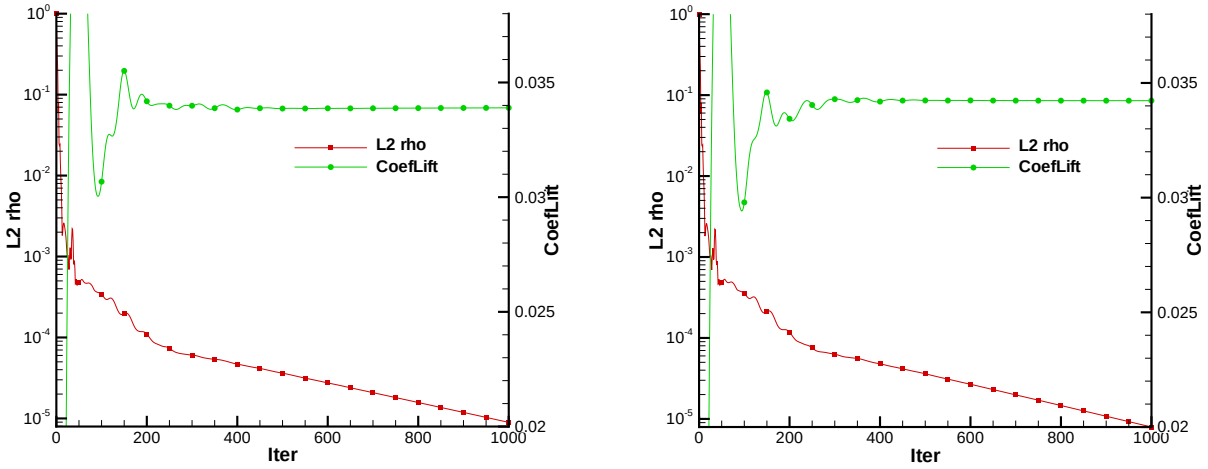
A typical convergence history, L2 norm of the density residual and the lift coefficient, for one of the spectral solutions for the RANS computation is shown in figure 2.3. After approximately 250 multigrid cycles (4w cycle) the convergence start to deteriorate due to the high aspect ratio cells needed to capture the boundary layer and the wake. Note that the convergence and the lift coefficient are relatively independent of the turbulence model.

Aachen Axial Turbine

The subsonic Aachen turbine consists of three blade rows with 36 stators, 41 rotors and 36 stators respectively. The rotor section rotates at 3500 RPM. To reduce to size of the problem the number of rotors have been changed to 36, such that a 1-1-1 configuration is obtained. Due to this quite drastic geometrical change the results of the Time Spectral Method are only compared with the results from a BDF computation; comparison with experimental data is omitted, although the impact of this kind of approximation has been rigorously tested in the past.

The surface grid of the 58 block grid is shown in figure 2.4. The total number of cells in the corresponding volume grid is approximately 2 million.

For comparison purposes, this configuration has been computed with the Time Spectral Method using 4



(a) Spalart Allmaras

(b) k-ω

Figure 2.3: Convergence histories for the RANS computation of AGARD CT6 test case

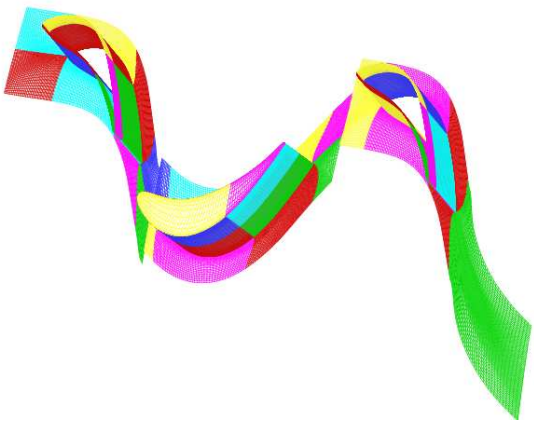


Figure 2.4: Surface grid for the 1-1-1 scaled Aachen turbine. Total number of cells in the volume grid is 2,070,528

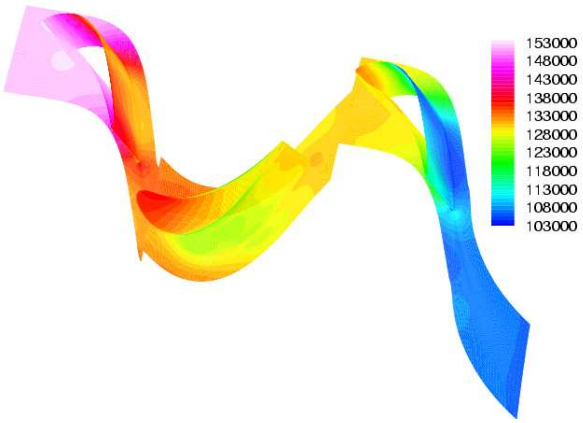


Figure 2.5: Surface pressure distribution for the 1-1-1 scaled Aachen turbine.

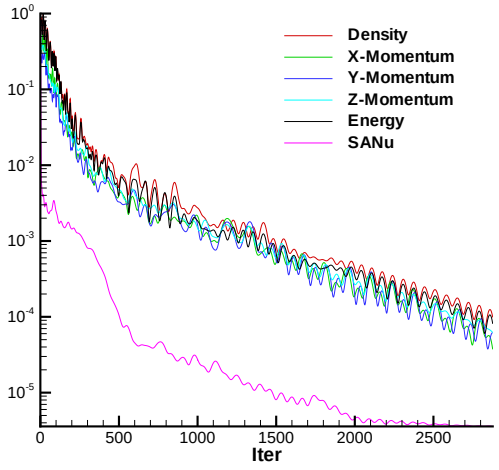


Figure 2.6: Convergence histories for the Time Spectral Method with 4 time intervals for the 1-1-1 scaled Aachen turbine; 3W cycle combined with a 5 stage Runge-Kutta pseudo-time integration scheme.

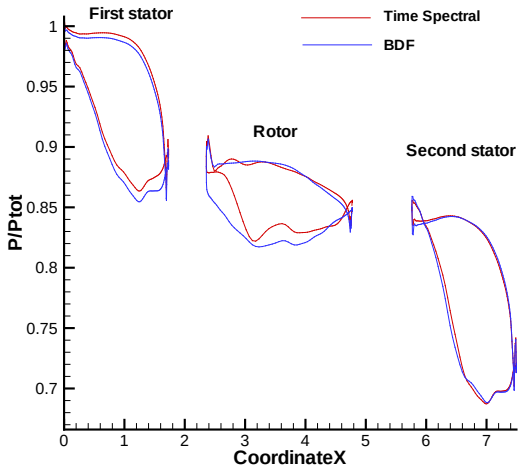


Figure 2.7: Pressure distribution on the blades at a radial cut of 10.6 inch for the Time Spectral and the second order BDF solution

modes and with a second order BDF in combination with 50 time steps per blade passage and 100 multigrid cycles per time step. For all computations the Spalart-Allmaras turbulence model was used. Due to the relatively low RPM the the solution was periodic after 5 blade passages for the BDF. The surface pressure distribution is shown in figure 2.5.

The convergence histories for the Time Spectral Method are shown in figure 2.6. All residuals converge about 4 orders of magnitude in approximately 2,500 multigrid cycles, which is typical for an internal RANS computation.

Figure 2.7 shows the pressure distribution at a radial cut located at 10.6 in on the 3 blades for the Time Spectral solution and the solution obtained with the second order BDF. The agreement on the stator blades is quite good, but on the rotor blade there is a significant difference between both solutions. This is most likely caused by the lack of temporal resolution in the Time Spectral Method and more time intervals should be used for an accurate resolution.

Even for this very low RPM case, where the time-domain solution converges rather rapidly (in situations where the machine turns at speeds nearly five times that of this test case many more than 5 blade passings may be necessary to reach a periodic steady state) the computational advantage of the time spectral approach is approximately a factor of 4. For the higher RPM case, the method is expected to yield improvements in computational time of about a factor of 10 with low numbers of modes.

Conclusions and Future Work

The formulation for the Time Spectral Method has been presented as well as a correction needed to treat sector periodicity. Numerical results for the pitching airfoil confirm the accuracy as well as the fact that engineering accuracy can be obtained with just a few modes.

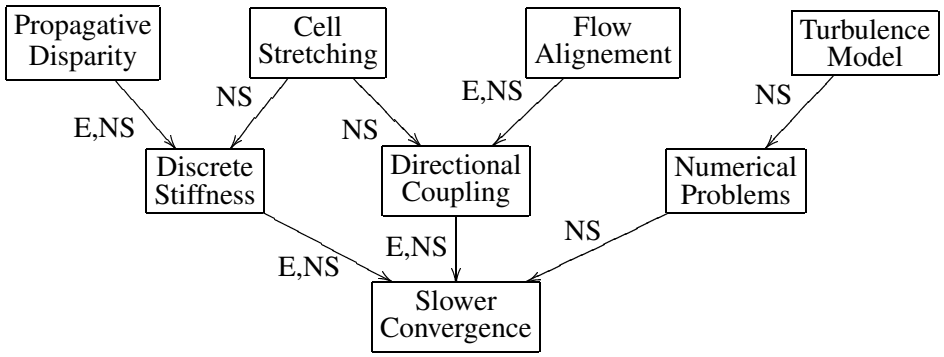


Figure 2.8: Sources of Stiffness for Numerical Solution of High Reynolds Number, Viscous, Compressible Flow. E - Euler Equations, NS - RANS Equations

For the 1-1-1 scaled Aachen turbine the temporal resolution obtained with only four time intervals per blade passage seems inadequate and further investigation is needed to find out if the Time Spectral Method could be a viable alternative for the BDF for this type of application.

The explicit treatment of the time-spectral source term appears to work well for pitching airfoils and low RPM turbines, but it fails for high RPM compressors. Currently an implicit treatment of the time-spectral source term is being investigated. This approach leads to a linear system for the updates in each cell for every pseudo time step. The linear system can be solved efficiently when use is made of the properties of the matrix corresponding to discretization of the source term.

The current implementation is very memory intensive, because all the arrays must be allocated for all solutions at the same time. A successful extension to a high number of modes, $O(100)$, may therefore not be possible unless a fairly large machine is available. However, the current approach can still be used with a reduced number of modes to provide a very good initial guess for the standard BDF such that a high quality periodic solution can be obtained much faster.

At the moment the parallelization only occurs at the spatial level and not at the time level. Massively parallel computers could be used much more efficiently if also the time could be parallelized. In principle this could be done rather easily for the Time-Spectral Method, because all the solutions are computed simultaneously. However, the stencil for the time derivative, equation (2.6) is not compact and therefore it is not well-suited for a parallel approach. Nevertheless the option should be investigated since the computation of this term does not dominate the complete procedure.

2.1.2 Squared Preconditioning

Multigrid multistage compressible flow solvers such as TFLO2000 encounter convergence problems due to numerical stiffness, directional decoupling, etc. Figure 2.8 (due to Pierce [6]) illustrates various sources of multigrid breakdown. Block-Jacobi preconditioning [7, 6] addresses some of these problems to a great extent.

In the limit of incompressible flows, however, compressible flow solvers show degraded convergence speed and loss of accuracy. It can be shown that the condition number of the continuous system of Euler or Navier-Stokes equations goes to infinity as the Mach number goes to zero. In addition, accuracy is lost due to improper scaling of the artificial dissipation terms. These problems cannot be addressed by the Block-

Jacobi preconditioner but can be successfully treated by using low-speed preconditioners [8, 9, 10, 11, 12, 13, 14, 15, 16].

There exists a growing need for compressible flow solvers that can tackle incompressible flow problems without loss of convergence speed or accuracy. Such problems occur, for example, in the secondary flow system and the turbine of a jet engine where Mach numbers as low as 0.01 and as high as 1.2 can be encountered in the same flow. An example of this kind of flow computed using the old version of TFLO can be found in Figure 2.9.

It is possible to combine Block-Jacobi preconditioning with low-speed preconditioning in an approach we refer to as squared preconditioning [11, 10, 15] that combines the advantages of both types of preconditioners.

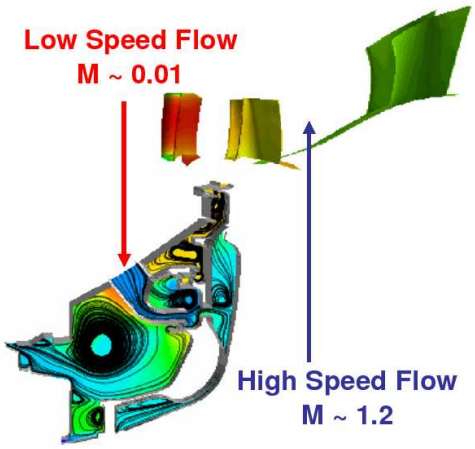


Figure 2.9: Main Flow Path and Secondary Flow System Example

In order to modify an existing flow solver to include squared preconditioning, let us consider the case of the Euler equations using scalar local time-stepping and high-order matrix artificial dissipation:

$$W^{n+1} - W^n + \left[\frac{1}{\sigma}\left(\frac{\rho\left(A_x\right)}{\Delta x} + \frac{\rho\left(A_y\right)}{\Delta y}\right)\right]^{-1}\left[A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \mu_a\left(\Delta x^3\left|A_x\right|\frac{\partial^4 W}{\partial x^4} + \Delta y^3\left|A_y\right|\frac{\partial^4 W}{\partial y^4}\right)\right] = 0. \tag{2.22}$$

Using squared preconditioning we now have a matrix time-step that includes a low-speed preconditioner P into the structure of a Block-Jacobi preconditioner while using re-scaled dissipation terms:

$$W^{n+1} - W^n + \left[\frac{P^{-1}}{\sigma}\left(\frac{\left|PA_x\right|}{\Delta x} + \frac{\left|PA_y\right|}{\Delta y}\right)\right]^{-1}\left[A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \mu_a\left(\Delta x^3 P^{-1}\left|PA_x\right|\frac{\partial^4 W}{\partial x^4} + \Delta y^3 P^{-1}\left|PA_y\right|\frac{\partial^4 W}{\partial y^4}\right)\right] = 0. \tag{2.23}$$

We have shown [17, 18] that the full potential of squared preconditioning can only be achieved through a proper application of entropy fix and the use of optimized values for modified Runge-Kutta multistage coefficients.

Furthermore, a cost-effective and robust implementation of squared preconditioning is not straightforward. We have chosen our low-speed preconditioner to be the Weiss-Smith preconditioner based on Darmofal's approach to the squared preconditioning problem [15] and we have developed an ultra-compact and symmetric form of the matrix $P^{-1}|PA|$ in entropy variables that includes an entropy fix in its analytical form. We use transformations between conservative variables and entropy variables in the implementation that result in simple preconditioned Jacobians such as

$$P^{e-1}\left|P^e A^e\right|^* = \begin{pmatrix} \frac{c(s+|\lambda_2|^*)-q_n r(1-\epsilon)}{c\epsilon} & rn_x & rn_y & 0 \\ rn_x & sn_x^2+|\lambda_2|^* & sn_x n_y & 0 \\ rn_y & sn_x n_y & sn_y^2+|\lambda_2|^* & 0 \\ 0 & 0 & 0 & |\lambda_2|^* \end{pmatrix}. \tag{2.24}$$

with

$$\epsilon = \left\{ \begin{array}{cc} \frac{M^2}{1-3M^2} & M<0.5 \\ 1 & M\geq 0.5 \end{array} \right\}, \qquad q_n = un_x + vn_y, \tag{2.25}$$

$$\tau = \sqrt{(1-\epsilon)^2 q_n^2 + 4\epsilon c^2}, \qquad \lambda_2 = \lambda_3 = q_n, \tag{2.26}$$

$$\lambda_1 = \frac{1}{2}\left[(1+\epsilon)q_n - \tau\right], \qquad \lambda_4 = \tfrac{1}{2}\left[(1+\epsilon)q_n + \tau\right], \tag{2.27}$$

$$r = \frac{c\left(|\lambda_4|^* - |\lambda_1|^*\right)}{\tau}, \quad s = \tfrac{c(|\lambda_1|^* - 2|\lambda_2|^* + |\lambda_4|^*)}{2} + \tfrac{q_n(1-\epsilon)r}{c}. \tag{2.28}$$

We observed substantial stability and convergence improvements using the following entropy fix formulation:

$$|\lambda_i|^* = \left\{ \begin{array}{cc} \frac{1}{2}(\delta c + \frac{\lambda_i^2}{\delta c}), & |\lambda_i| < \delta \max(|\lambda_i|) \\ |\lambda_i|, & |\lambda_i| \geq \delta \max(|\lambda_i|) \end{array} \right\}, \delta = \frac{1}{2}. \tag{2.29}$$

Figure 2.10 shows the convergence improvements achieved in TFLO2000 using squared-preconditioning for Euler calculations. The test case was a 48-block $192 \times 32 \times 48$ ONERA M6 wing at 3.06°. The convergence acceleration and accuracy improvement were particularly noticeable for the low Mach numbers 0.01 and 0.1, although it can be noted that the preconditioned scheme yields faster convergence across the range of Mach numbers.

Modifications to the current implementation to include the required terms for viscous flows are being pursued at the moment within the context of TFLO2000 and will be tested in both two- and three-dimensional configurations soon as part of a doctoral thesis in the Center.

2.2 Modeling and Fundamental Studies on Turbulence and Transition

Several projects on computational modeling of turbulence and transition have been pursued during this year. In this section we report on the progress on several fronts.

We first start with a description of several turbulence modeling efforts that have been or are being implemented in our main flow solver, TFLO2000. This section includes a discussion of the current status of the v2-f model implementation including improvements to the numerical scheme, a discussion of recent efforts

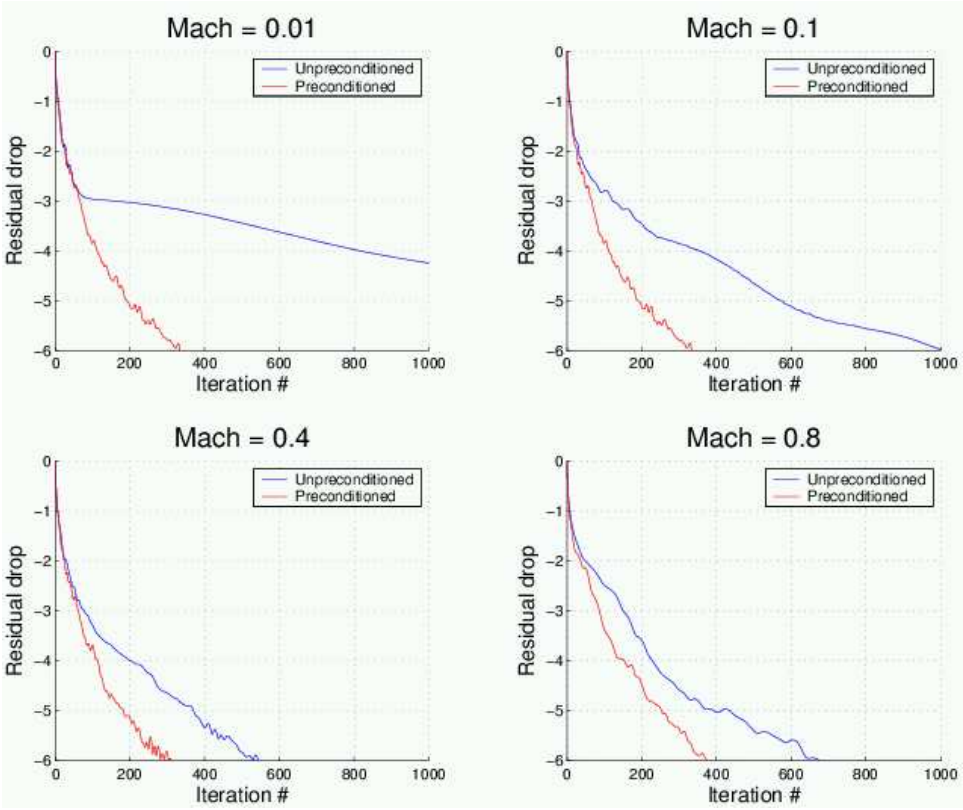


Figure 2.10: Convergence Histories of Squared Preconditioning for TFLO2000

to incorporate algebraic Reynolds stress ideas to conventional turbulence models and the incorporation of the effects of rotation and buoyancy into existing turbulence models in TFLO2000.

We then focus on a new method to develop and implement wall function boundary conditions in RANS codes and the application of a compressible LES code to transition simulation. The first is likely to enhance the capability of TFLO2000 to treat large, complex problems. The method is a general and can be used in any applied RANS code. It has also been implemented into our IB-RANS code. The second topic deals with simulations of bypass transition using LES. Our previous transition simulations have been conducted using DNS. It is known that standard LES methods are inadequate for transition. The process is intermittent and the smallest scales in the transition region play a critical role. Both of these argue against the philosophy of simulating the large scales and adding sub-grid dissipation. The present simulation shows that by combining a high order scheme with a fine boundary layer grid, results similar to DNS can be obtained. We conclude this section by detailing some of our recent efforts in simulating film cooling using LES. Although these efforts are at their beginning stages, they are already starting to produce significant results.

2.2.1 TFLO2000 v2-f Model Implementation

In the last year, work in turbulence modeling concentrated on three main topics: improvements in the numerical scheme for v2-f, development of turbulence models for rotating wall bounded flows and further development of adaptive wall functions for RANS.

Improvement of Numerical Scheme for v^2-f

In segregated RANS flow solvers the temporal advancement of turbulence equations is often realized using local time steps. In the old TFLO, the time step was determined by scaling the local time step from the mean flow with a constant factor. This factor was model specific and usually obtained by numerical experimentation. Another way of advancing the turbulence equations is by means of implicit under-relaxation. As shown in [20], implicit under-relaxation is equivalent to local time stepping with the time step:

$$\Delta t = \frac{\alpha}{b(1-\alpha)} \tag{2.30}$$

where α is the relaxation parameter ($0 < \alpha < 1$) and b is the diagonal element of the implicit matrix. The advantage here is that the time step for the turbulence equations is determined from the numerical properties of the discretized turbulence equations rather than through mean flow criteria and numerical experimentation for the scaling factor. However, there are some disadvantages when implicit under-relaxation is applied to the v^2-f equations.

The v^2-f turbulence model [21] consists of, in addition to the usual transport equation for the turbulent kinetic energy k and dissipation ϵ, a transport equation for the scalar $\overline{v^2}$ and an elliptic relaxation equation for f. The $\overline{v^2}$ and f-equation are:

$$\partial_t \overline{v^2} = \frac{1}{\rho}\nabla \cdot \left[(\mu + \mu_t)\nabla \overline{v^2}\right] - U \cdot \nabla \overline{v^2} + kf - \frac{\epsilon}{k}\overline{v^2} \tag{2.31}$$

$$f = L^2 \nabla^2 f + \frac{C_1}{T}\left[\frac{2}{3} - \frac{\overline{v^2}}{k}\right] + C_2 \frac{P_k}{k} \tag{2.32}$$

The stiffness of the f wall boundary condition, $f_w = -20\nu^2\overline{v_1^2}/(\epsilon_w y_1^4)$, requires an implicit coupled solution of the $\overline{v^2}$ and f-equations. Implicit under-relaxation introduces an unsteady term in the f-equation changing its elliptic nature. In addition, local time stepping has been found to worsen the convergence for certain flows with strong shock-boundary layer interaction.

A way to avoid the introduction of an unsteady term in the f-equation is to use explicit under-relaxation. Here, the solution of the discretized equations (2.31) and (2.32) in delta-form is weighted and added to the previous iterate:

$$\phi^{n+1} = \phi^n + \alpha\Delta\phi \tag{2.33}$$

where α is again the relaxation parameter with $0 < \alpha < 1$.

Explicit under-relaxation seems to work well even when a DDADI factorization scheme is used. The DDADI scheme [26] used in TFLO2000 for v^2-f has been shown to be very robust convergence for a number of flow field computations when explicit under-relaxation is used. Current work assesses the impact of the factorization error on the solution.

The numerical scheme developed for v^2-f allows full 3-D computations of the flow field computations in a very robust manner. The scheme does not utilize any numerical limiters to stabilize the solution algorithm. Most computations with v^2-f converge to the steady state in the same number of multigrid cycles as when Spalart-Allmaras or k-ω are used. We are currently applying the model to unsteady computations.

2.2.2 Improvements in Modeling Rotating Wall-Bounded Flows

Linear eddy-viscosity models are known to be inaccurate in predicting the effect of strong streamline curvature and frame rotation. There is no shortage of modifications and adjustments proposed in the literature to correct their behavior. For example, in [34] the k-ϵ model is modified by introducing a coefficient in the ϵ-equations that depends on the shear rate and frame rotation. A more consistent redesign of the ϵ equation for flows with rotational effects has been proposed in [25]. Another recent attempt in [22] consists of the modification of the eddy-viscosity coefficient. Although these modifications have shown to provide encouraging predictions for simple flows with rotation (namely channel flows), their accuracy for more complex situations remains unclear. Differential Reynolds stress models, on the other hand, possess the obvious advantage that the turbulence production terms and the stress anisotropy are automatically accounted for. Unfortunately, the difficulties in modeling the stress redistribution terms and their inherent numerical stiffness make them not amenable to mainstream use in engineering calculations.

Algebraic Reynolds stress models have received a substantial amount of attention given the potential benefit of introducing stress anisotropy in the *controlled* environment of an eddy-viscosity closure. Several models have been devised with various degree of success ([23], [35]). The basic idea behind these models is to express the Reynolds stress tensor as a function of one or more (up to ten) different tensors. This is not different from what is used to derived the so-called non-linear eddy-viscosity models where additional (high-order) terms are added to the Boussinesq relationship between mean strain and Reynolds stresses. Reynolds and co-workers (e.g. [33], [28], [29]) have repeatedly argued that for adequate modeling and description of rotating turbulence, information about the turbulence structure is crucial. The Reynolds stresses only characterize the componentality of turbulence, *i.e.*, which velocity components are more energetic. The turbulent field has much more information than that contained in the Reynolds stresses, which is important in presence of rotation, and which is described by the turbulence structure. For instance, the dimensionality of the flow is important. This carries information about which directions are favored by the more energetic

turbulent eddies: if the turbulent eddies are preferentially aligned with a given direction, then the dimensionality is smaller along that direction. Here, hypothetical turbulent eddies are used to bring awareness of turbulence structure into the turbulence model. Averaging over an ensemble of eddies produces a set of one-point statistics, representative of the eddy field, and a set of equations of state relating the Reynolds stresses to these statistics.

The structure-based approach to build the Reynolds stress closure has lead in [30] to the development of the Algebraic Structure-Based Model (ASBM) in conjunction with a *novel* two-equation model based on the transport equation for turbulent kinetic energy, k, and large scale enstrophy ω^2. The model has been calibrated for channel flow simulations and the results have shown excellent agreement with available DNS data.

The primary objective of this research was to combine of the ASBM Reynolds stress evaluation with conventional turbulence model, namely the k-ω [36] and $v^2 - f$ [31] models. Results are presented for channel flow with and without spanwise rotation.

Rotating Channel Flow

The steady RANS equations governing the motion of an incompressible viscous fluid in a Cartesian rotating frame of reference are given by conservation of mass and momentum as [24]:

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{2.34}$$

$$u_j \frac{\partial u_i}{\partial x_j} + 2\epsilon ijk\Omega_j^f u_k = -\frac{\partial P}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_k \partial x_k} + \frac{\partial}{\partial x_j}\left(-\overline{u_i' u_j'}\right). \tag{2.35}$$

where u_i is the mean velocity measured in the coordinate system rotating with constant angular velocity Ω_j^f, and x_j, P, ρ, and ν represent respectively the position vector, reduced pressure, density and kinematic viscosity. The reduced pressure is given by

$$P = \frac{p}{\rho} + \mathcal{U} - \frac{1}{2}(\epsilon ljk\Omega_j^f x_k)(\epsilon lpq\Omega_p^f x_q), \tag{2.36}$$

where p is the thermodynamic pressure, $\mathcal{U}$ is such that a conservative body force per unit mass (*e.g.* gravity) is given by $\partial\mathcal{U}/\partial x_i$, and the last term represents the centripetal acceleration.

For fully-developed channel flow in a spanwise rotating frame the mean velocity is given by $u_j = u(y)$ where y is the wall normal direction. The frame rotation rate vector is given by $\Omega_j^f = \Omega_z^f$ with z being the spanwise direction. The wall-normal mean velocity component vanishes by continuity for a fully developed channel flow with zero velocity at the walls. This simplifies the momentum equation; only the momentum equation in streamwise direction x needs to be solved and the term containing the angular velocity Ω^f is zero.

The Reynolds stress $\tau_{ij} = -\overline{u_i' u_j'}$ in equation (2.35) is obtained with the algebraic structure-based procedure described [30]. The complete ASBM model includes two scalar transport equations for the turbulent kinetic energy k and the large-scale turbulent enstrophy ω^2. The purpose of these two quantities is to provide the field distribution of k and of the turbulence time scale τ. The latter has the following relation to k and ϵ:

$$\tau^2 = \left(\frac{k}{\varepsilon}\right)^2 + \left(2.0\sqrt{\frac{\nu}{\varepsilon}}\right)^2. \tag{2.37}$$

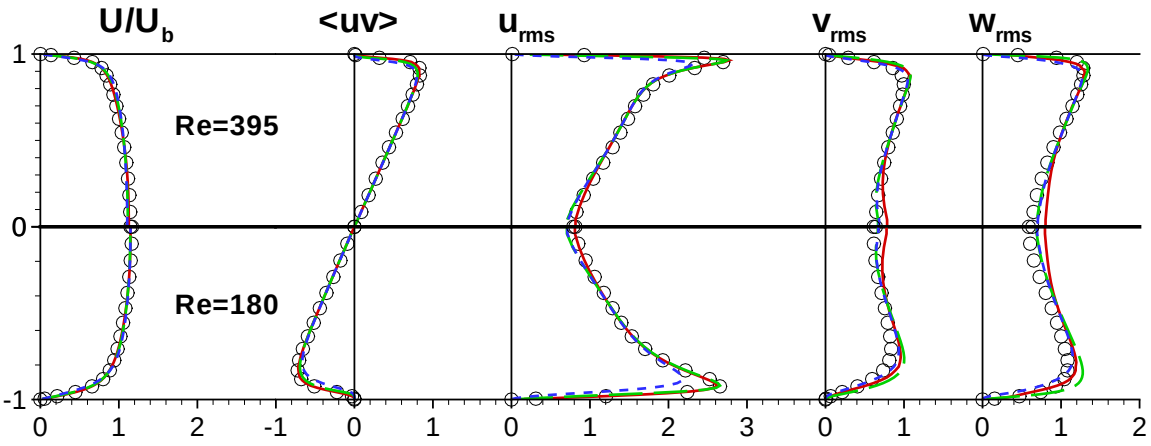


Figure 2.11: Channel flow at $Re = 395$ (upper half) and $Re = 180$ (lower half). ——: ASB v^2-f, — —: ASB low-Re k-ω, - - - - : ASB high-Re k-ω, o: DNS by [32].

The ASB Reynolds stress is evaluated from the tensors τS_{ij}, $\tau\Omega_{ij}$ and B_{ij} as well as k and τ and the frame rotation vector $\tau\Omega_j^f$. The vorticity Ω_{ij} and strain S_{ij} are obtained from the mean flow velocity distribution. The blockage tensor B_{ij} is obtained from an elliptic equation.

In this work, field distributions of k and τ have been obtained from the k-ω model and v^2-f model. In addition, low-Re modifications given in [36] for the k-ω model have been considered. For the k-ω model, the time scale is computed as $\tau^2 = 1/(\beta^*\omega)^2 + 4.0\nu/(k\omega\beta^*)$ while equation (2.37) is used directly for v^2-f which includes transport equations for k and ϵ.

In the momentum equation (2.35) the Reynolds stress enters only in the diffusion term. In the transport equations of the turbulence models the Reynolds stress enters the production term $P_k = \tau_{ij}\partial u_i/\partial x_j$. The diffusion term in the turbulence models retains the eddy-viscosity. No additional modifications of the turbulence equations have been performed.

The channel flow computations have been performed with the incompressible Cartesian code IBRANS [27]. The computational domain consists of two cells in streamwise direction. The flow in this direction is treated as periodic flow; the pressure and velocity components at the outflow have been copied to the inflow and a source term has been added to the u-momentum equation to account for the pressure loss.

Numerical results

Channel flow simulations in orthogonal mode rotation have been performed for a variety of Reynolds and Rossby numbers. Fig. 2.11 shows the effect of the Reynolds number for the flow without rotation. In this case, the results with the high- and low-Re ASB k-ω, as well as the ASB v^2-f models are reported. Here we added the ASB prefix to the models to indicate that the Reynolds stresses are evaluated with the ASB procedure. The latter two produce results that are satisfactory for both Reynolds numbers whereas the high-Re ASB k-ω under predicts the peak of the u_{rms} in particular for $Re = 180$.

Fig. 2.12 and Fig. 2.13 report results for a channel flow at two different Rossby numbers. The Rossby number is defined here as $Ro = \frac{\Omega^f 2h}{u_b}$ where Ω^f is the magnitude of the frame rotation rate, h is the half-height of the channel and u_b is the bulk velocity in the channel. In these plots DNS data and the original

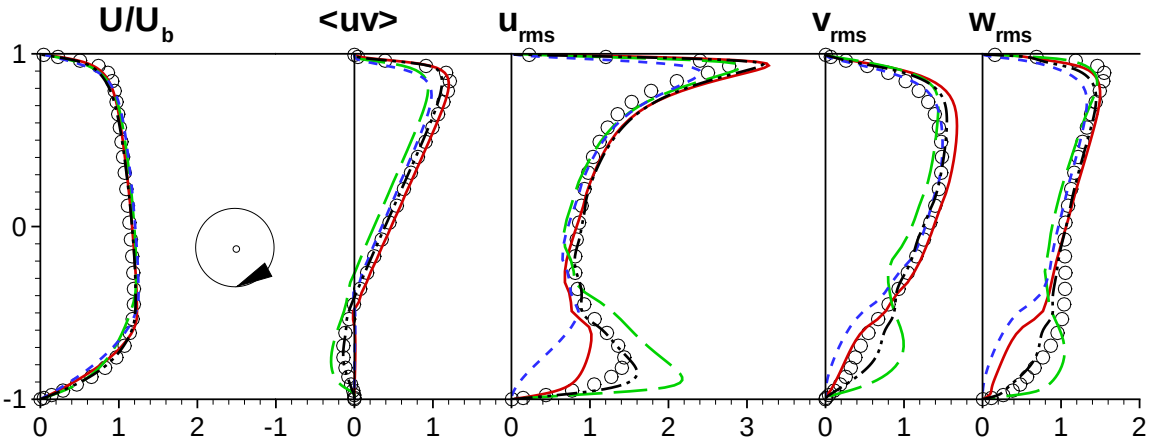


Figure 2.12: Rotating channel flow at $Re = 180$ and $Ro = 0.22$. ——: ASB v^2-f, – – –: ASB low-Re k-ω, - - - - : ASB high-Re k-ω, —··—: ASBM, o: DNS by [19].

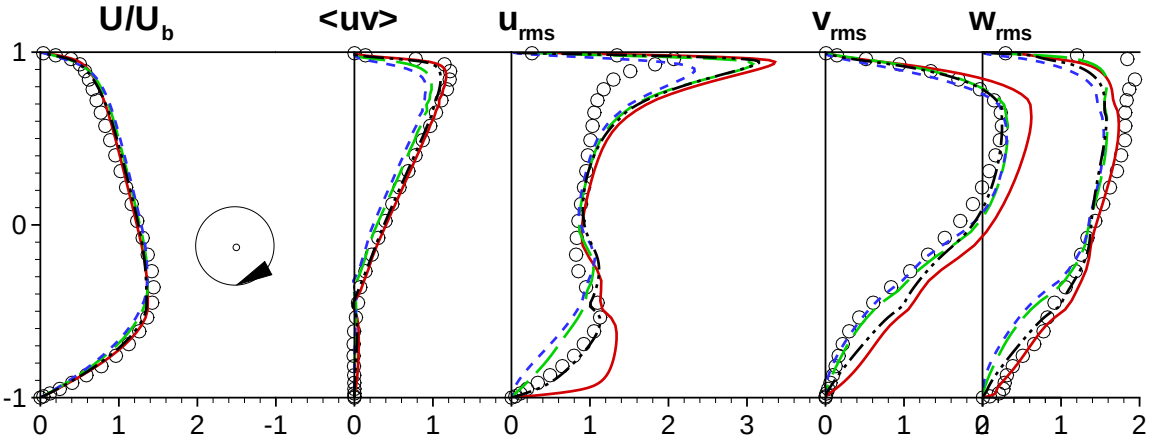


Figure 2.13: Rotating channel flow at $Re = 180$ and $Ro = 0.77$. ——: ASB v^2-f, – – –: ASB low-Re k-ω, - - - - : ASB high-Re k-ω, —··—: ASBM, o: DNS by [19].

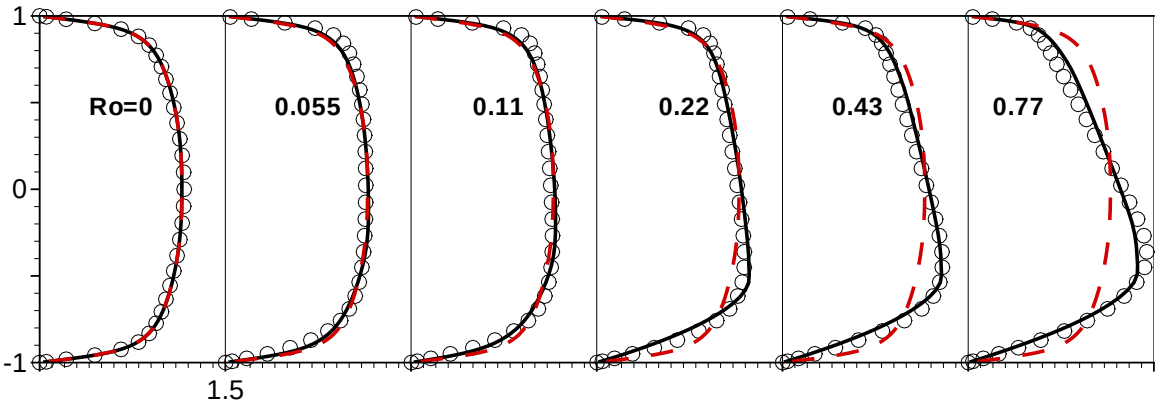


Figure 2.14: Velocity profiles for channel flow at $Re = 180$ and $Ro = 0., 0.055, 0.11, 0.22, 0.43$ and 0.77; - - - - : v^2-f, —— : ASB v^2-f, o: DNS by [19]

.

ASBM are compared to ASB k-ω and ASB v^2-f predictions. The asymmetry in the mean velocity profile is properly captured even for the high Rossby case. In addition, the Reynolds stress anisotropy is remarkably close to the DNS results at the turbulence-enhanced side of the channel. Notice that at the pressure-side of the channel (lower side in the Fig. 2.12 and Fig. 2.13) turbulence intensity is reduced and, eventually, the turbulent stresses are negligible with respect to the viscous stresses. The correspondence between mean flow predictions and correct level of anisotropy is very encouraging. The difference between the full ASBM approach and the current combined approach is also very small especially when v^2-f is used.

Further simulations have been performed at a variety of Rossby numbers in the $[0 - 0.77]$ interval. The results obtained using the v^2-f and the ASB v^2-f are presented. The mean velocity profile is reported in Fig. 2.14. The current model and the DNS agree remarkably well.

The algebraic structure-based model has been combined with conventional linear eddy-viscosity models to evaluate the Reynolds stress in the RANS equations. This approach has proven to be very accurate in predicting the mean flow and the stress anisotropy in rotating channel flow as opposed to the baseline eddy-viscosity predictions that are typically insensitive to frame rotation.

The current combination of the ASB Reynolds stress evaluation with the v^2-f and k-ω models is carried out in a full three-dimensional flow solver. However, only channel flow simulations were performed. Preliminary computations of flows in square-ducts appeared encouraging. Convergence for this flow was poor indicating that further work on the numerical scheme and the ASB algorithm is needed. Current work focuses on the clarification of this issue.

2.2.3 Modeling Rotation and Buoyancy Effects on Turbulence

The scalar turbulence models (e.g. $k-\omega$) currently used in TFLO2000 do not embody the effects of rotation and buoyancy on turbulence. In as much as it is true that mean flow predictions will be sometimes acceptable, evidence indicates that this will not be so inside turbine blade internal cooling passages. In the Second Moment Closure (SMC) turbulence model, body forces due to rotation and buoyancy are accounted for in

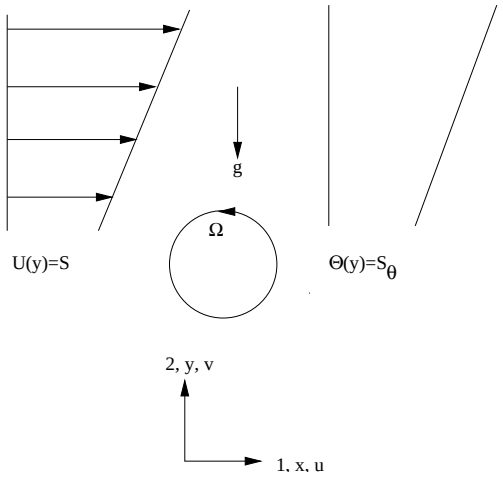


Figure 2.15: Schematic of flow configuration.

a systematic manner; therefore, it is a good place to start modeling the effects of rotation and buoyancy on turbulence. A properly formulated SMC model can lead to the modification of the scalar turbulence model to include body force effects.

Modeling efforts begin with homogeneous turbulence (Figure 2.15); inhomogeneous terms such as those due to walls are added at a later stage. The equations for Reynolds stresses and fluxes for homogeneous turbulence are

$$d_t\overline{u_iu_j} = P_{ij} + R_{ij} + B_{ij} - 2/3\varepsilon\delta_{ij} + \phi_{ij}, \tag{2.38}$$

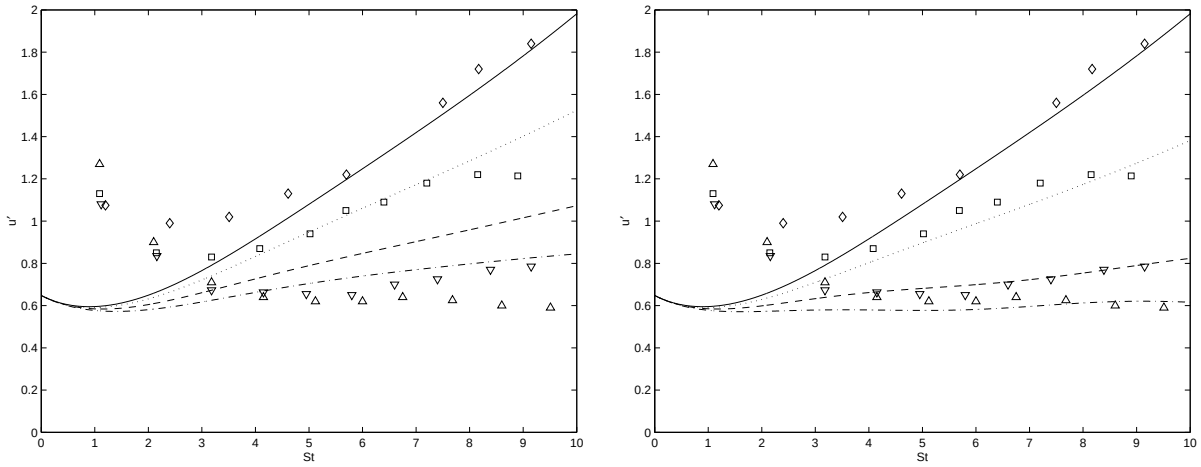
$$d_t\overline{u_i\theta} = P_{i\theta} + R_{i\theta} + B_{i\theta} + \phi_{i\theta}. \tag{2.39}$$

The pressure-strain, ϕ_{ij}, and pressure-temperature gradient terms, $\phi_{i\theta}$, are closed using the General Linear Model (Durbin and Pettersson Reif [22])

Equilibirum and bifurcation analysis is a powerful tool in developing SMC models and is used here. We obtained a closed form solution of the General Linear Model, which is expressed as a function of rotation (Ω/S) and buoyancy (Ri_g). With the use of the closed form solution, it is found that the IP model, one of the most widely used SMC models, does not have a real and stable solution when certain magnitudes of rotation and buoyancy are combined. We have found a set of coefficients that is consistent with experimental data and ensures a real and stable solution for all combinations of rotation and buoyancy. A more detailed description of this work can be found in Ji and Durbin [37]. Figure 2.16 shows the root-mean-square streamwise velocity fluctuations compared with experimental data. Figure 2.17 shows heat flux ratio as a function of Ri_g. Currently, the simulation of buoyant, rotating channel flow using the SMC model is in progress and the application to the existing models in TFLO2000 is being considered.

2.2.4 Near-Wall Behavior of RANS Turbulence Models and Implications for Wall Functions

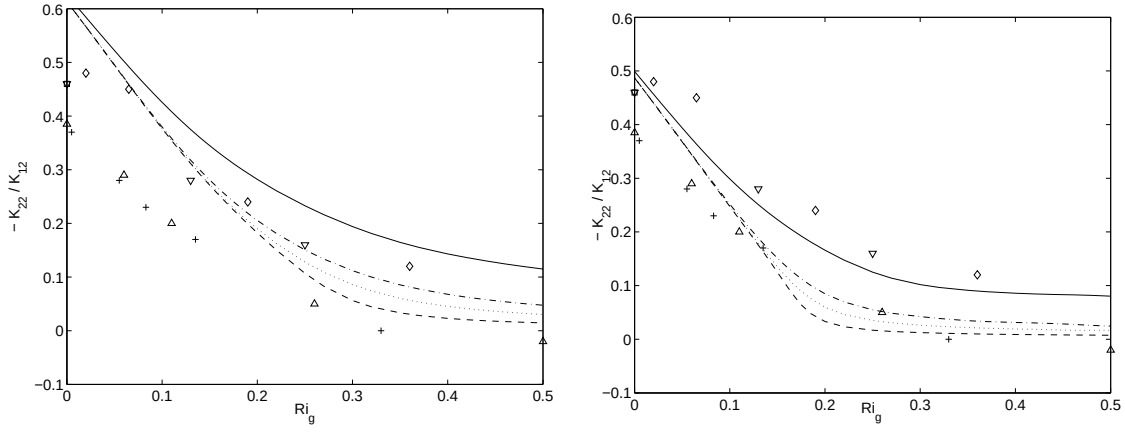
The size of industrial CFD problems has grown considerably in recent years and, despite a rapid increase in computational resources, there are still many applications for which the grid resolution is insufficient for



(a) IP model

(b) New model

Figure 2.16: u' vs. St for several gradient Richardson numbers. Lines are model predictions: solid, 0; dotted, 0.075; dashed, 0.21; dashdot, 0.37. Symbols are experimental results: $\diamond$, 0; $\square$, 0.075; $\triangledown$, 0.21; $\triangle$, 0.37.



(a) IP model

(b) New model

Figure 2.17: Heat flux ratio $\overline{v\theta}/\overline{u\theta}$ vs. gradient Richardson number Ri_g. Lines are model predictions: solid, $\tau = 15$; dashdot, $\tau = 45$; dotted, $\tau = 75$; dash, $\tau = 165$. Symbols are experimental and numerical simulation results.

wall integration and accurate wall functions are a necessity. For example, computation of rotating stall in turbomachinery requires large scale, detailed flow field predictions in the entire compressor [20]; assuming that a high quality grid is employed in each blade passage, the computational grid would easily exceed 50 million cells. Wall functions are also an essential ingredient for the development of Cartesian Immersed Boundary RANS methods [27]. Shortcomings of existing wall functions motivate further research in this field.

The theory that lies behind wall functions is the universal character of the law-of-the-wall. This asserts that under many flow conditions the form of the solution between the wall and the outer edge of the logarithmic layer is invariant when appropriate scaling is used. From early on, with the development of the first successful RANS turbulence model, the universal character of the logarithmic layer has been invoked to design off-wall boundary conditions. The earliest wall functions required the first point above the wall to lie in the logarithmic layer. That is a too severe constraint, which will often be violated by the grid. The present formulation circumvents some of the shortcomings of the existing wall functions and aims to be an efficient, and robust approach that is applicable to various turbulence models implemented into TFLO2000.

The turbulent flow velocity profile can be split in three distinguished regions: the viscous sublayer, the logarithmic layer and the defect layer. In a quasi-equilibrium boundary layer, the region between the wall and the outer edge of the logarithmic layer is universal; i.e. the profiles of the flow variables collapse when scaled with the friction velocity u_τ and molecular viscosity ν. This universality allows the derivation of wall functions.

Near the wall, derivatives in streamwise direction can be neglected and the flow and turbulence variables depend only on the coordinate y which is directed normal to the wall. Essentially, this is a turbulent Couette flow approximation. To derive solutions for the viscous sublayer and logarithmic layer, the equations are recast in non-dimensional form. The velocity and various turbulence variables in plus units are:

$$U^+ = \frac{U}{u_\tau},\ y^+ = \frac{y u_\tau}{\nu},\ \nu_t^+ = \frac{\nu_t}{\nu},\ \tilde{\nu}^+ = \frac{\tilde{\nu}}{\nu},\ k^+ = \frac{k}{u_\tau^2},\ \omega^+ = \frac{\omega \nu}{u_\tau^2}$$

$$g^+ = \frac{g u_\tau}{\sqrt{\nu}},\ \epsilon^+ = \frac{\epsilon \nu}{u_\tau^4},\ \overline{v^2}^+ = \frac{\overline{v^2}}{u_\tau^2},\ f^+ = \frac{f \nu}{u_\tau^2} \tag{2.40}$$

For the flow over a flat plate at zero pressure gradient the RANS equations simplify in the region between the wall and the outer edge of the logarithmic layer to:

$$\frac{d}{dy}\left((\mu + \mu_t) \frac{dU}{dy} \right) = 0 \tag{2.41}$$

Integration along the wall normal coordinate y yields:

$$(\mu + \mu_t) \frac{dU}{dy} = \rho u_\tau^2 \tag{2.42}$$

This equation states that the sum of the viscous and turbulent shear stress is constant and equal to the wall shear stress $\tau_w = \mu \left(\frac{\partial u}{\partial y} \right)_w = \rho u_\tau^2$. Introducing the non-dimensionalization (2.40), equation (2.42) becomes

$$(1 + \nu_t^+) \frac{dU^+}{dy^+} = 1 \tag{2.43}$$

Suppose ν_t^+ were known. Then (2.43) could be integrated to find the universal function $U^+(y^+)$. An early approach was to assume a form for that function, such as Spalding's profile [38]. However, it is more consistent to develop the function that is appropriate to each model by solving the wall layer equations numerically. Now, knowing the universal function, the friction velocity can be solved from the computed velocity at the first grid point. Say that is U_1, evaluated at y_1. If this lies in the wall layer then

$$Re_y \equiv y_1 U_1/\nu = y_1^+ U^+(y_1^+) = F(y_1^+) \tag{2.44}$$

Re_y is the Reynolds number obtained from the first cell. The right is a universal function. Inverting this function gives

$$y_1 u_\tau/\nu = F^{-1}(Re_y)$$

Hence, u_τ can be found. Given $F(y_1^+)$, the inversion can be done iteratively by Newton's method. However, this can be done once and for all, and the inverse function stored in a table.

For the turbulence equations, analytical solutions can often be derived for the viscous sublayer and for logarithmic layer. However, the intermediate region is problematic. This is addressed by generating a look-up table for each non-dimensionalized turbulence variable from a well-resolved numerical solution. This table takes the form of spline coefficients. Succinctly, this method has been applied to the models used in TFLO2000 to generate the requisite tables. This provides a wall function formulation that is correct for each particular model — $k-\omega$, $k-g$, v^2-f and Spalart-Allmaras.

Numerical Implementation

The initial testing of the new wall function formulation was in our Cartesian, incompressible RANS flow solver, IB-RANS.

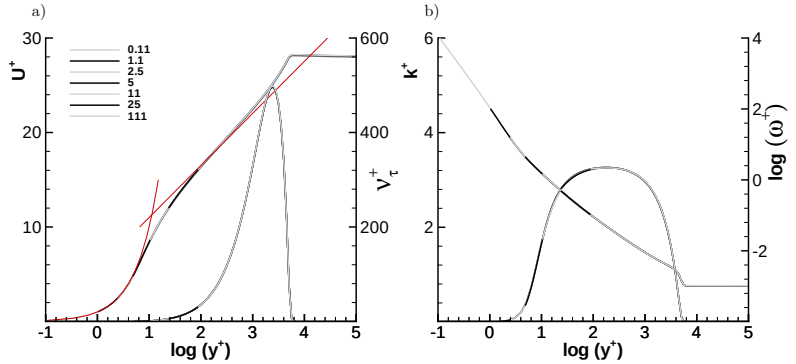
There are many ways to implement wall functions into a Navier-Stokes code. However, the main ingredients are as follows. The u-velocity component requires a condition that ensures the correct shear stress at the wall (equation (2.42)). Since the diffusion term is treated implicitly here, the correct wall flux is included as an explicit correction on the right hand side. To compute the correction, u_τ is needed. Using the values y_1 and U_1 from the first cell center a local Reynolds number Re_y is computed. A value of u_τ is obtained from the corresponding look-up table.

An off-wall boundary condition is adopted for the turbulence variables by enforcing the first cell center values. These values can be obtained from the near-wall behavior of turbulence models via the look-up table that was created for each variable.

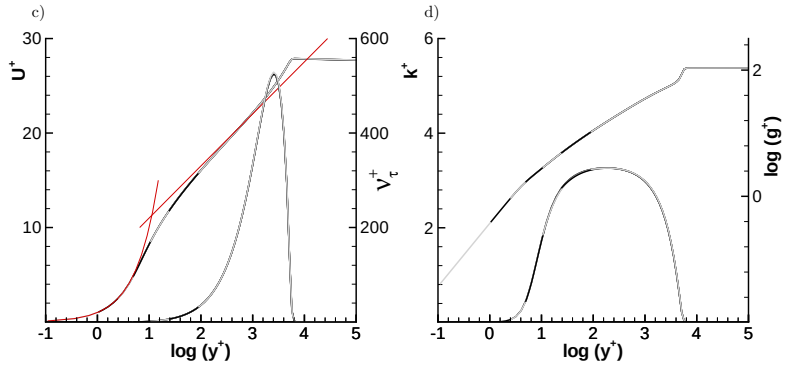
Flow over flat plate

Flow over a flat plate was solved to the downstream location defined by momentum thickness based Reynolds number of $Re_\theta = 7700$. All turbulence models first have been solved on grids with grid that is shifted away from the wall. These are called δ-grids. They allow the wall function to be tested without contamination by poor resolution in the first cell. After considering these, solutions will be given on standard coarsened grids.

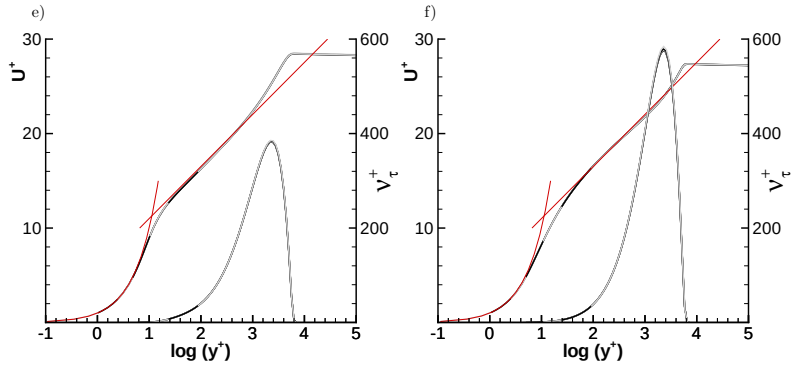
When δ-grids are used with the correct boundary conditions, the computations collapse onto the wall integration profile. This is true for all the models considered, as shown in Figure 2.18. Since the results for all the grids are practically coinciding, the colors of the curves are chosen to alternate between grey and black to visualize the first grid point for each grid.



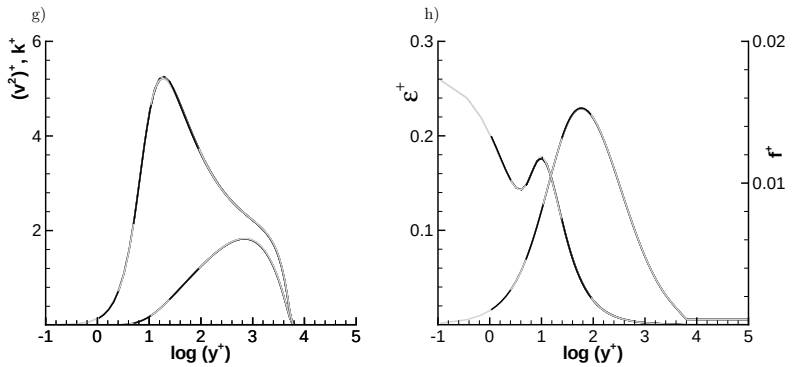
Velocity and eddy-viscosity profiles (a) and k and ω profiles (b), k-ω model



Velocity and eddy-viscosity profiles (c) and k and g profiles (d), k-g model



Velocity and eddy-viscosity profiles Spalart-Allmaras (e) and v^2-f model (f)



k and v^2 profiles (g) and ε and f profiles (h), v^2-f model

Figure 2.18: Application of adaptive wall functions using δ-grids

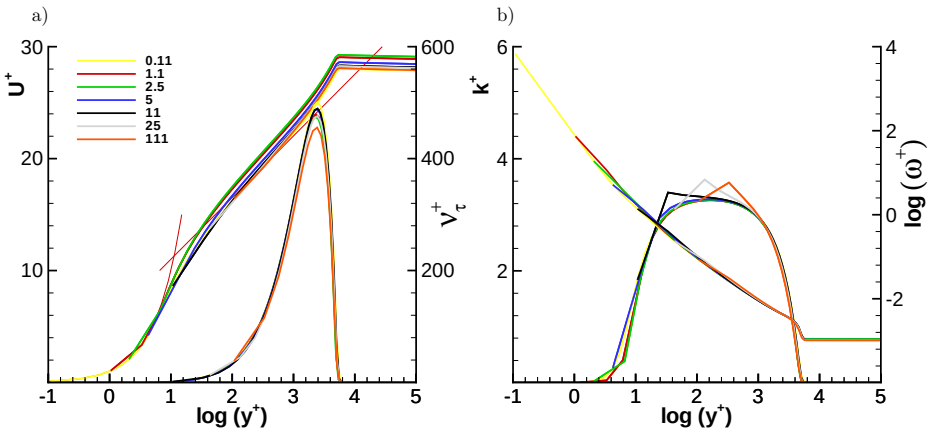


Figure 2.19: Standard application of adaptive wall functions using coarse grids. k-ω model.

The results obtained on standard coarse grids are presented in Figure 2.19. For brevity, on the $k-\omega$ result is shown. The spread in the curves is the consequence of numerical errors. Our formulation suggests a way to improve the accuracy.

Improving numerical results on coarse grids

An investigation has shown that the numerical error is mainly of local character. Increased accuracy of the momentum flux at face 12 has a significant impact on the results. The eddy-viscosity is a highly non-linear function of wall distance in the viscous sublayer and the buffer region. The numerical scheme, however, uses a linear interpolation to compute the eddy-viscosity at face 12 (Figure 2.20) when computing the diffusive flux. A remedy is to enforce the correct value of ν_{t12}^{+} using the corresponding y_{12}^{+}. This value can be computed from the appropriate analytical expression or look-up table. This can be seen as having a more accurate interpolation scheme at our disposal.

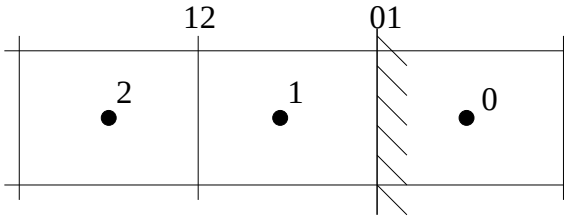


Figure 2.20: Near-wall grid structure.

Numerical results obtained with this methodology are presented in Figure 2.21 . The velocity profiles for Spalart-Allmaras and v^2-f models are significantly improved. It is interesting to note, that even though the diffusive flux has also been improved in the turbulence equations the impact on the turbulence variables is negligible.

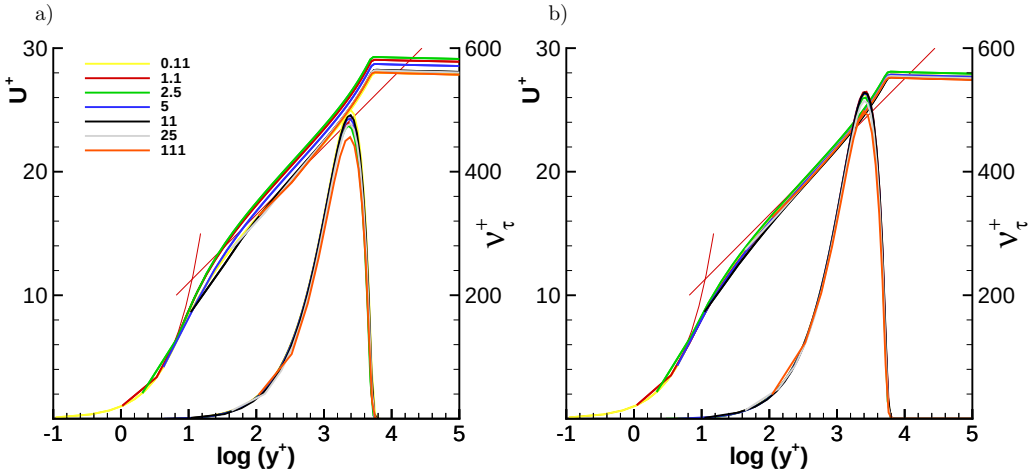
Recirculating flow

The proposed adaptive wall functions were derived for zero pressure gradient flow over a flat plate. Their ability to capture pressure gradient driven separation and reattachment is studied by simple and reproducible test problem: the boundary layer over a flat plate with an imposed streamwise pressure gradient. The pressure gradient is modulated by suction and blowing applied at a given distance from the plate as illustrated in Figure 2.22. The grid consists of 312×96 cells. A detailed grid-dependence study has shown that this resolution provides sufficient accuracy both in x and y directions.

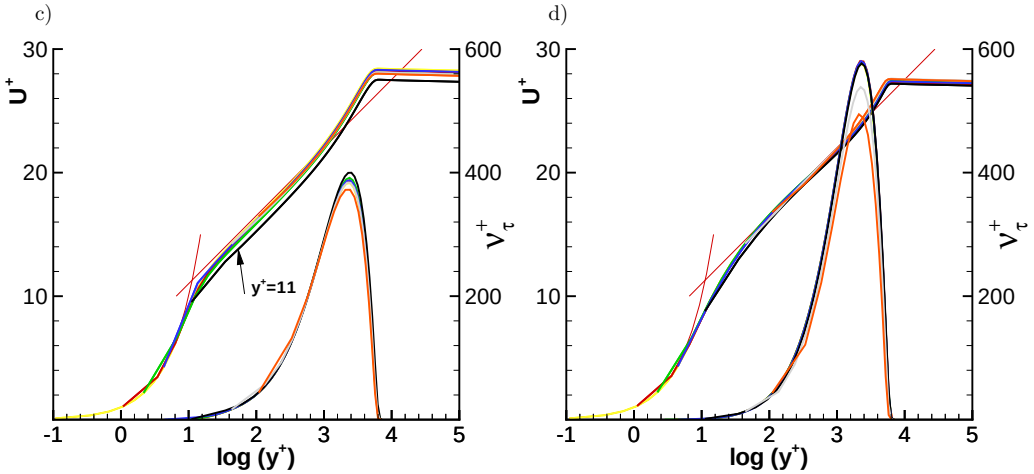
Streamlines and contours of eddy-viscosity computed using various turbulence models with wall integration are presented in Figure 2.23. The skin friction and pressure distribution for zero pressure gradient and for a recirculation-inducing pressure gradient are reported in Figure 2.24. The results show significant sensitivity of the solution to the model. The length of the recirculation region scaled with the displacement thickness $\delta^*_{0.1}$ at $x/L = 0.1$ is in the range of $1000 < L_{rec}/\delta^*_{0.1} < 1600$.

In order to assess the performance of the proposed adaptive wall functions, we consider grids with the first cell center at $y^+ = 0.11, 1.1, 2.5, 5, 11, 25, 111$, at the entrance. Note that in contrast to the flat plate the y^+ values vary considerably in the recirculation region.

The results presented in Figures 2.25 and 2.26 indicate that the proposed wall functions function properly for Spalart-Allmaras and k-ω models (the same behavior was observed for k-g model). The skin friction coefficients computed using wall integration and wall functions on the $y^+ = 0.11$ grid coincide.



Velocity and eddy-viscosity profiles, k-ω (a), k-g model (b)



Velocity and eddy-viscosity profiles, Spalart-Allmaras (c), v^2-f model (d)

Figure 2.21: Adaptive wall functions using coarse grids + imposing $\nu_{t_{12}}^{+}$

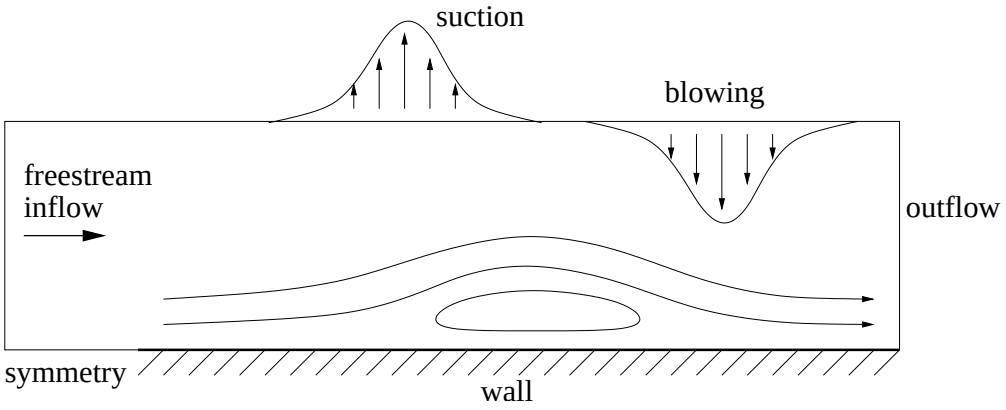


Figure 2.22: Flat plate with separation induced via suction and blowing at the upper boundary.

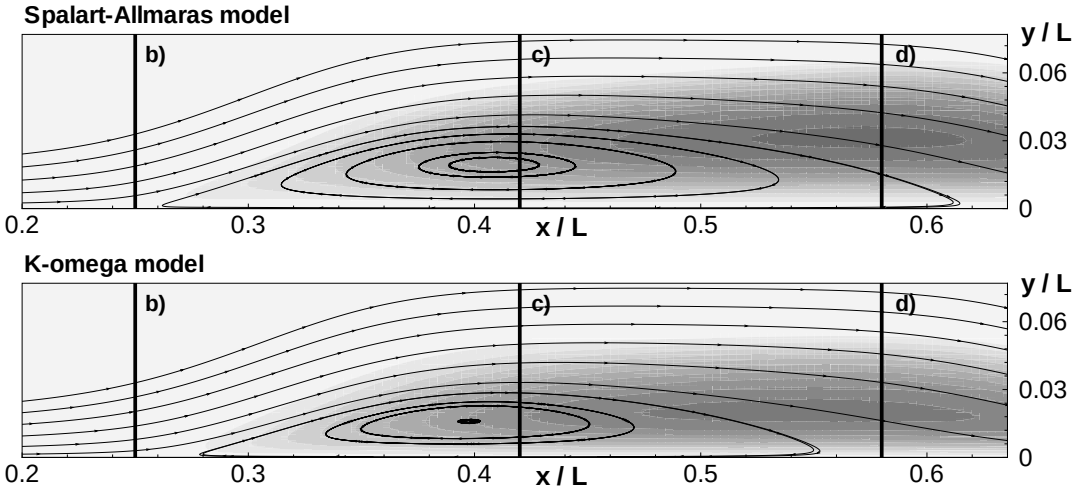


Figure 2.23: Streamlines showing the recirculation and contours of eddy-viscosity.

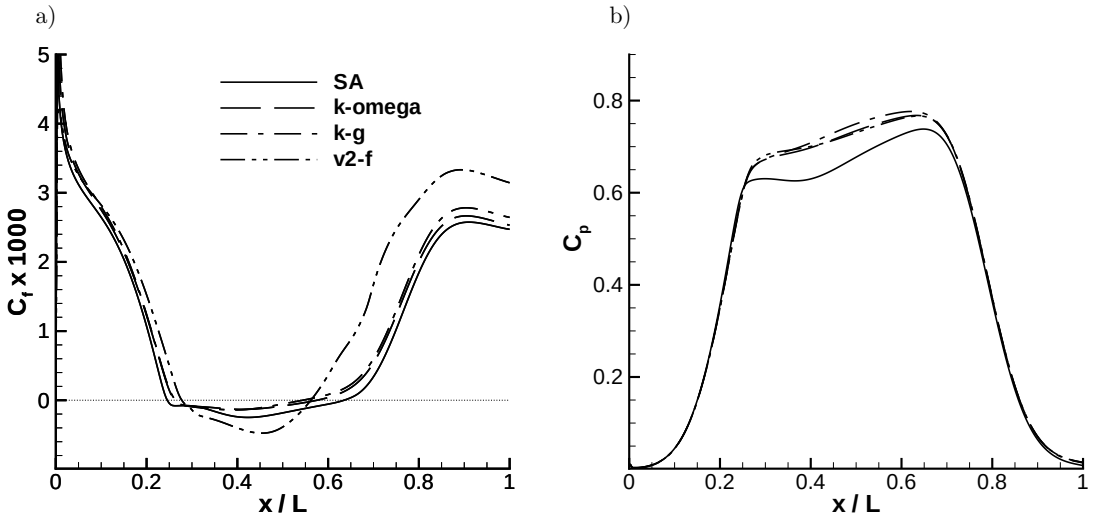


Figure 2.24: Skin friction (a) and pressure distribution (b) for the flow with recirculation with v^2-f, k-ω, k-g and Spalart-Allmaras models.

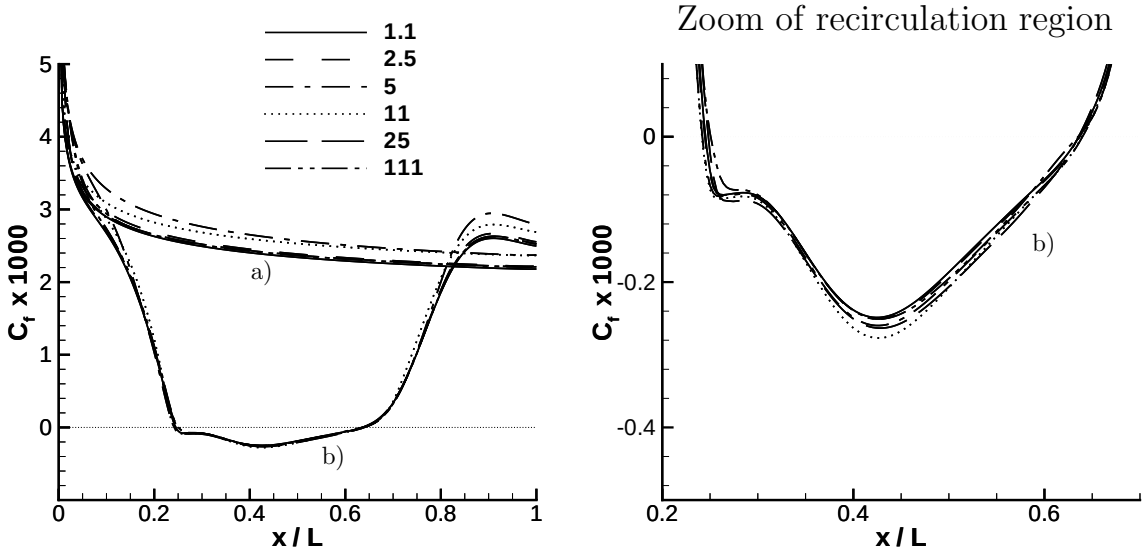


Figure 2.25: Skin friction for zero pressure gradient (a) and adverse/favorable pressure gradient (b) using Spalart-Allmaras model: comparison of wall functions results.

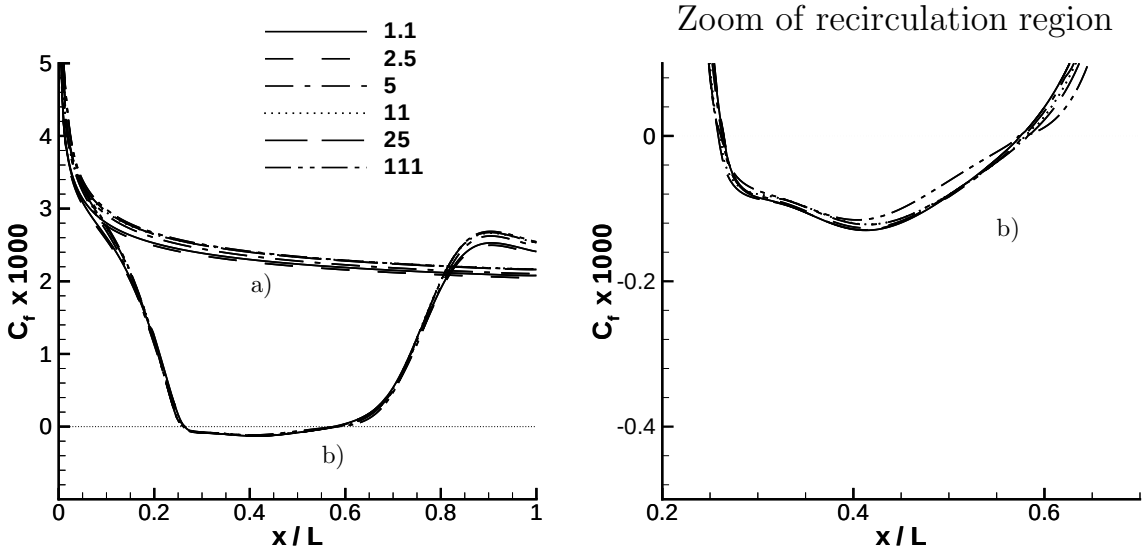


Figure 2.26: Skin friction for zero pressure gradient (a) and adverse/favorable pressure gradient (b) using k-ω model: comparison of wall functions results.

2.2.5 Transition simulation with compressible LES.

The problem of transition to turbulence remains poorly understood. Renewed interest in laminar-turbulent transition stems from the associated increase in shear stress and heat loads on the overall system. For example, transition of the boundary layer on the surface of a turbine blade is an important factor affecting the efficiency of low pressure turbines (LPT), which in turn, has a significant impact on overall engine performance. In fact, a 1% improvement in LPT efficiency reduces specific fuel consumption by 0.96%. Therefore, transition has important economic and environmental consequences that necessitate a better understanding of the transition process.

In the presence of free-stream turbulence (abbreviated hereinafter as FST), boundary layer transition does not follow the mechanism of growth of Tollmien-Schlichting (TS) waves followed by 3-D modulation and breakdown to turbulence, a process that is commonly referred to as *natural transition*. Instead, transition occurs sooner, via mechanisms not yet fully understood. [39] called this *bypass transition*, a term indicative of the fact that the aforementioned natural transition path to turbulence is bypassed. This process is characterized by growth over time scales much shorter than the viscous scale of TS waves. As a result, boundary layer transition can occur at momentum thickness Reynolds numbers (Re_θ) of a few hundred instead of a few thousand. FST levels are relatively high in low pressure turbines. The boundary layer therefore transitions via the bypass mechanism; natural transition occurs rarely. Even though there is broad agreement that the conditions at the leading edge have a significant impact on transition location, not much effort has been put into studying this effect. Such an investigation is also hampered by the fact that bypass transition remains poorly understood by itself and the additional complexity introduced by the leading edge is therefore avoided. Based on some results available in the literature two possible leading-edge mechanisms emerge a) earlier transition due to the increased amplitude of TS waves growing in a boundary layer buffeted by FST, and b) quicker growth of Klebanoff modes due to anisotropy introduced by stretching of vortices around the leading edge. One of the goals of this study is to assess the relative importance of these mechanisms.

Towards this end, a suitable methodology for accurate computation of transitional flows over turbine-like geometries is developed and applied to the canonical problem of bypass transition of the boundary layer on a blunt-nosed flat plate. A direct numerical simulation of this problem was carried out by [40] earlier in this ASCI program. However, direct simulation is still very expensive. To overcome this cost constraint, a mixed direct- and large-eddy simulation approach is utilized here. In the boundary layer, the grid needs to be fine enough to resolve all scales. On the other hand, the free-stream turbulence can be handled less accurately with LES. The method utilizes sixth-order compact schemes on a staggered mesh for spatial differencing while zonal explicit-implicit time marching is utilized for optimum performance. Details of the method are presented in [41].

Simulation Conditions

The body simulated consists of a flat plate fitted with a super-ellipse leading edges of aspect ratios 10 and 6. All simulations are conducted at a Mach number of 0.5 and a Reynolds number of $Re = U_\infty L/\nu = 50000$. The plates are at zero angle-of-attack and nominally in a zero pressure gradient flow.

A laminar flow is obtained and is used to initialize a 3-D turbulent flow simulation in which free-stream turbulence (generated as described in [41]) is imposed at the inflow boundary. In addition to the leading edge aspect ratio, the characteristics of the free-stream turbulence are also varied. The simulations conducted are summarized in table 2.2.

	Case A	Case B	Case C	Case D	Case E
Leading Edge	AR10 LE	AR10 LE	AR10 LE	AR6 LE	AR6 LE
Turbulence Intensity Tu	3.5	4.5	4.5	4.5	3.5
Integral Scale $\mathcal{L}$	0.03	0.03	0.045	0.045	0.03
$Re_{\mathcal{L}}$	90	117	156	156	90

Table 2.2: Summary of simulations conducted

Averaged Results

Though the simulation conditions were not chosen to match any particular experiment, the conditions for simulation case A results are comparable to the ERCOFTAC transition case T3A [42]. The variation of skin-friction coefficient C_f with Re_θ is also in good agreement with experimental observation, as shown in figure 2.27. In the simulations, as in the experiments, the skin friction coefficient indicates that transition onset around $Re_\theta \sim 300$ and is complete by $Re_\theta \sim 600$. However, the simulated skin-friction coefficient in the pre-transitional *buffeted boundary layer* is slightly higher than the experiment (which remains fairly close to the Blasius value). This trend, also seen in the results of [43], may be attributed to the smaller turbulence length scale in the simulations. Due to the smaller length scale, the free-stream turbulence in the simulations has more energy in structures with characteristic spanwise length scales comparable to the boundary layer thickness. These structures affect the pre-transitional region as they penetrate into the boundary layer far more effectively. Profiles of RMS velocity fluctuation at $Re_\theta = 456$ are shown in figure 2.28. The computation slightly under-predicts levels in the RMS velocity but is in very good qualitative agreement with experimental observations.

The skin-friction plot in figure 2.27 also reproduce the trends observed in experiments. With an increase in the turbulence intensity from case A to case B, onset of transition moves upstream. This effect is well known and is included in correlations such as those of [44]. Furthermore, as observed in experiments, the increase in the integral length scale from case B to C, while holding the intensity fixed, also moves the transition location upstream.

Effect of the leading-edge

In case A, as in the simulations of [43], spots are formed by instabilities of *backward jets* which are essentially regions of fluid moving slower than the mean. It may therefore be concluded that when leading edge effects are unimportant, as in case A, formation of turbulent spots occurs through the interaction of a backward jet with a free-stream eddy, as observed by [43]. However, when leading edge effects are dominant, other mechanisms take over. In figure 2.29, a sequence of snapshots tracks the formation of turbulent spots in case B. The plan-view shows contours of spanwise velocity fluctuation at a distance of $y \sim \delta_{99}/4$ from the wall, where δ_{99} is the boundary layer thickness at the junction of the leading-edge and the flat plate. Wave-packet like precursors to spots are clearly visible in these snapshots. They appear to grow over the background disturbance inside the boundary layer, and eventually dominate the motion. They then continue to grow in size and increase in amplitude before breaking down into a patch of turbulence.

A magnified view of w velocity fluctuations in a precursor to a spot from case B is shown in figure 2.30. The structure is wave-packet-like in the streamwise direction, *i.e.*, it oscillates with an amplitude that decreases away from its center. This structure of the spot precursor is shown in figure 2.31 where

the spanwise velocity is shown versus the streamwise coordinate. There is a clear dominant wavelength corresponding to the oscillations inside the wave-packet, and an amplitude envelope that decays rapidly towards the edges of the structure. The wave-packets move at around $0.55U_\infty$.

The disturbances that lead to bypass transition in the absence of leading-edge effects usually occur at the outer reaches of the boundary layer in regions of intense backward flow relative to the mean. On the other hand, the wave-packet disturbances do not seem to require such strong backward jets, and they occur in the lower half of the boundary layer. This difference is shown in figures 2.32 and 2.33 that show snapshots of spanwise velocity fluctuations in the xy-plane along with fluctuating velocity vectors in the plane. Figure 2.32 corresponds to the backward jet mechanism in the initial stages. The slower moving fluid (left pointing vectors) and the initial stages of spot development in the outer half of the boundary layer are evident. On the other hand, 2.33 corresponds to the wave-packets. This picture, typical of all spot precursors formed under leading edge influence, shows the disturbance to be concentrated in the lower half of the boundary layer. This is in marked contrast with the backward jet mechanism of [43]. Furthermore, the spot precursors are not limited to regions of slower moving fluid, as is evident from the right-pointing vectors of figure 2.33 in the region of the young spot.

2.2.6 DNS of an Infinite Stationary and Rotating Serpentine Passage

The effects of rotation and curvature on fluid flow have been topics of interest for a long time (i.e. refer to Hopfinger and Lindens (1990) review article which summarizes the findings of 65 participants of the 245th Euromech Colloquium focusing on rotating, curved and stratified flows). Fluid flow through the complex turbine blade serpentine cooling passages are complicated by curvature and rotational effects. Additionally, the surfaces are typically ribbed to facilitate turbulent mixing thus enhancing heat transfer. Unfortunately, the addition of ribs in serpentine passages presents considerable difficulty for detailed experimental and numerical investigations, especially DNS. The focus of this investigation is a smooth walled serpentine passage. Additionally, the serpentine passages often found in turbine blade cooling passages alternate between round-ended and square-ended passages. The focus of this investigation is a round-ended serpentine passage. Another point of concern for serpentine passages is secondary flow effects. In order to conduct DNS simulations, endwall effects are negated by assuming periodicity, thus allowing for a constant grid spacing in the spanwise direction.

The purpose of this investigation is to gain a better understanding of the effect of curvature and rotation in a geometry that is fairly well representative of the passages typically found in turbine blades for cooling purposes. The knowledge gathered will provide an opportunity to better model the effect of turbulence, and hence improve heat transfer predictions in turbine blade cooling passages.

Methodology

A test matrix for two different Reynolds numbers and two different Rotation numbers for a smooth-wall round-ended serpentine passage was developed to conduct a thorough DNS study. The geometry was based loosely on channel flow simulations and U-bend passages found in the literature as well as those found in turbine blade cooling passages. The flow conditions were based on channel flow simulations commonly found in the literature. Initially two Reynolds numbers Re_b = 2800 and 4200 (based on δ corresponding to Re_τ = 180 and 250) were simulated for two different Rotation numbers Ro_b = 0 and 5 (corresponding to Ro_τ = 0 and 0.32). The geometry and computational grid of the serpentine passage is depicted in Figures 2.34

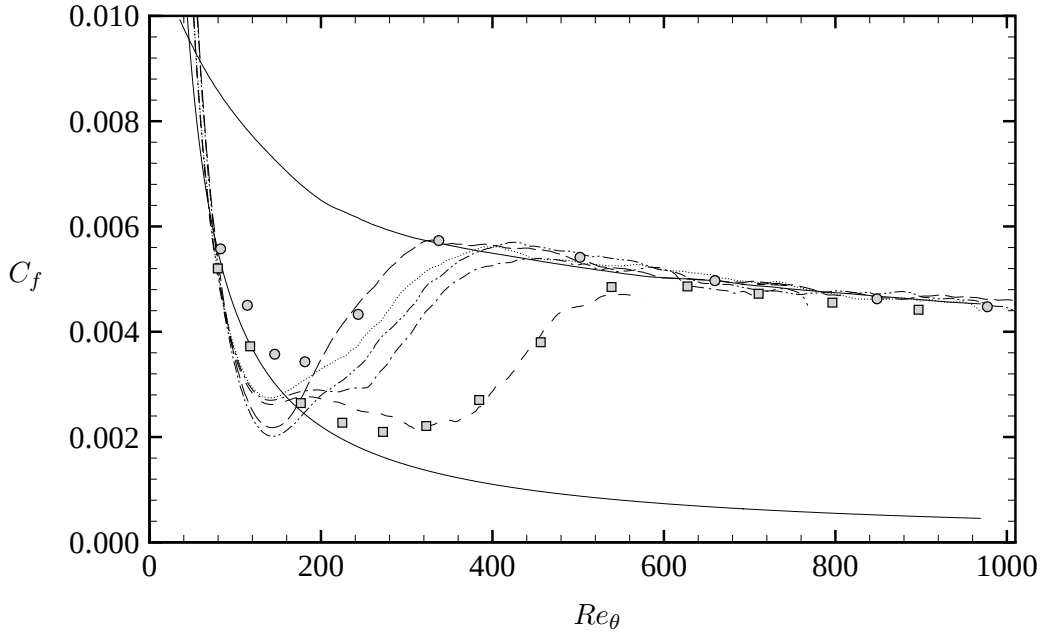


Figure 2.27: Skin-friction C_f as a function of momentum thickness Reynolds number Re_θ. ----, Case A; —·—, Case B; · · · ·, Case C; ——, Case D; —··—, Case E; ——, Blasius boundary layer and turbulent boundary layer correlation.

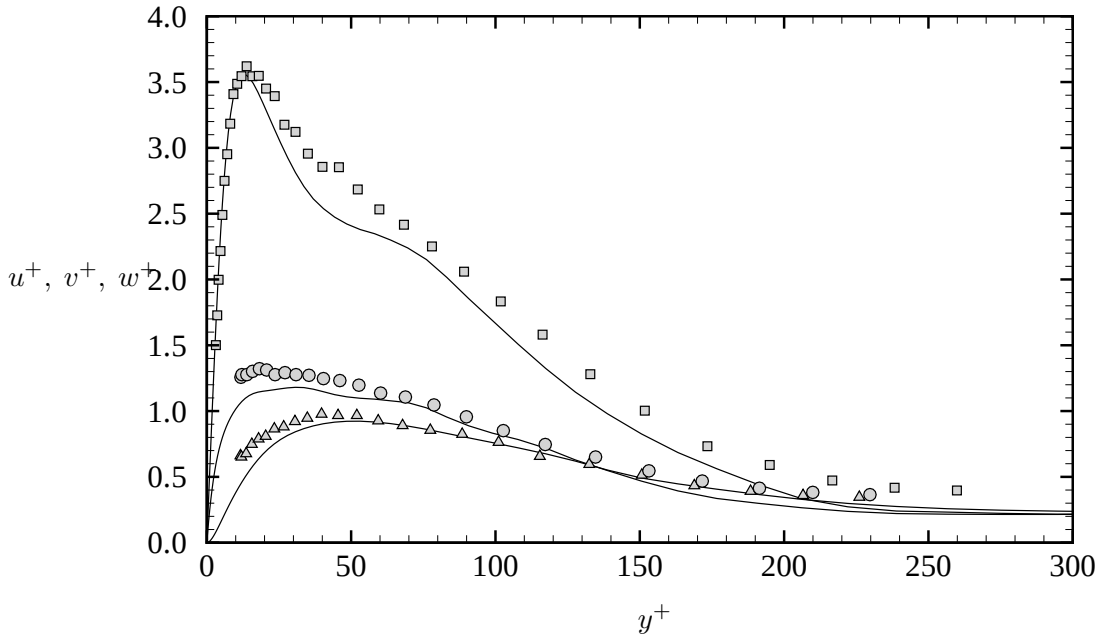


Figure 2.28: Profiles of RMS velocity fluctuation at $Re_\theta = 456$ compared to the experiments of [42]. Symbols are experimental data points □, u^+; △, v^+; ○, w^+.

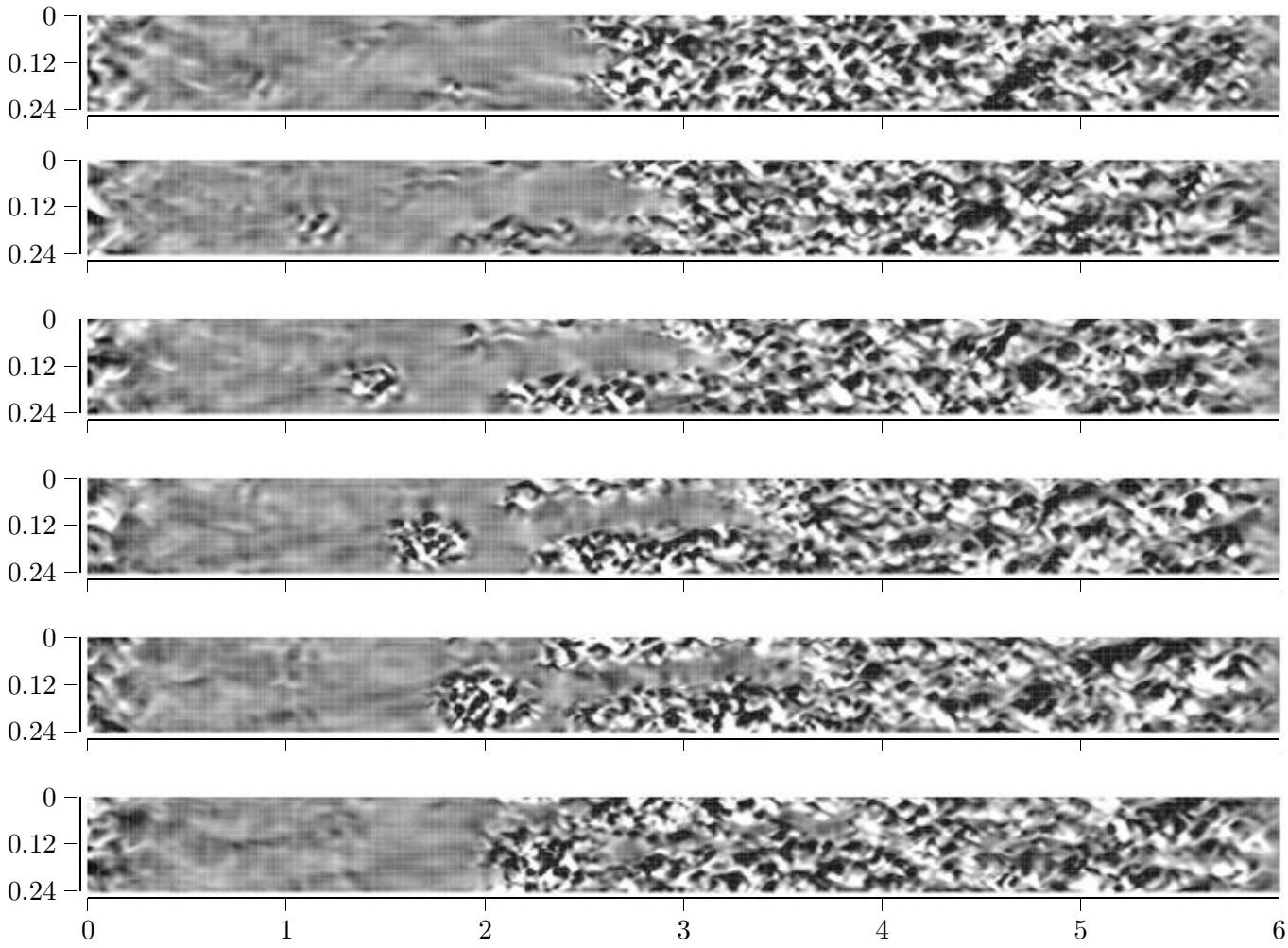


Figure 2.29: Plan-view (xz-plane) of the flat plate from case B showing incipient spots, their growth and eventual merger with the fully turbulent boundary layer. Snapshots of w-contours ($-0.08 < w/U_\infty < 0.08$) separated by $\Delta t = 0.75$. z-coordinate is magnified by a factor of 2.

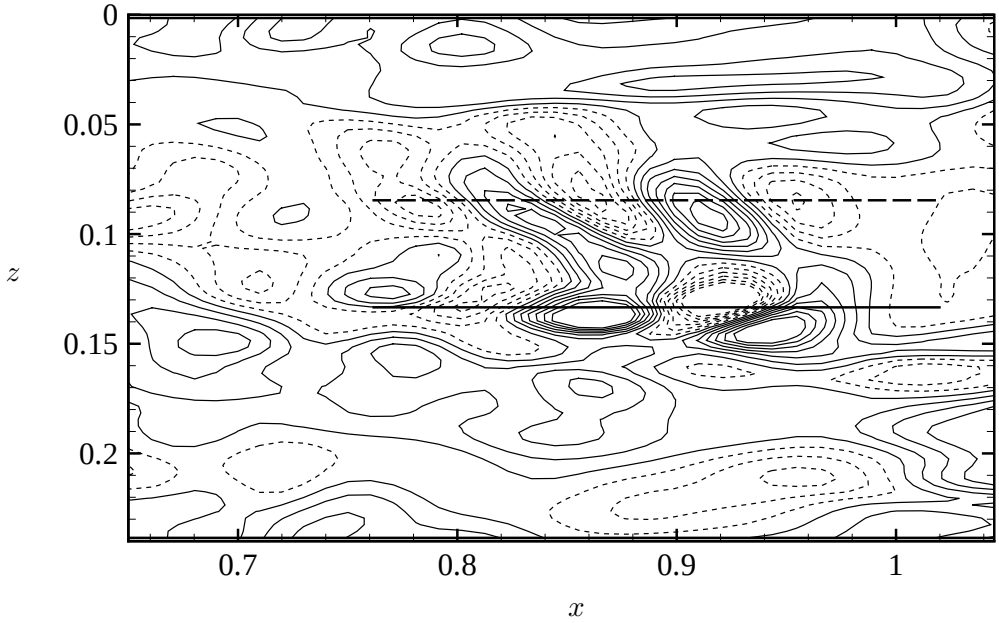


Figure 2.30: Contours of spanwise velocity fluctuations in a spot precursor in the xz-plane. From case B at $y = \delta_{99}/4$ with $-0.06 < w/U_\infty < 0.06$. Contour increment: 0.01. Dashed lines : Negative contours.

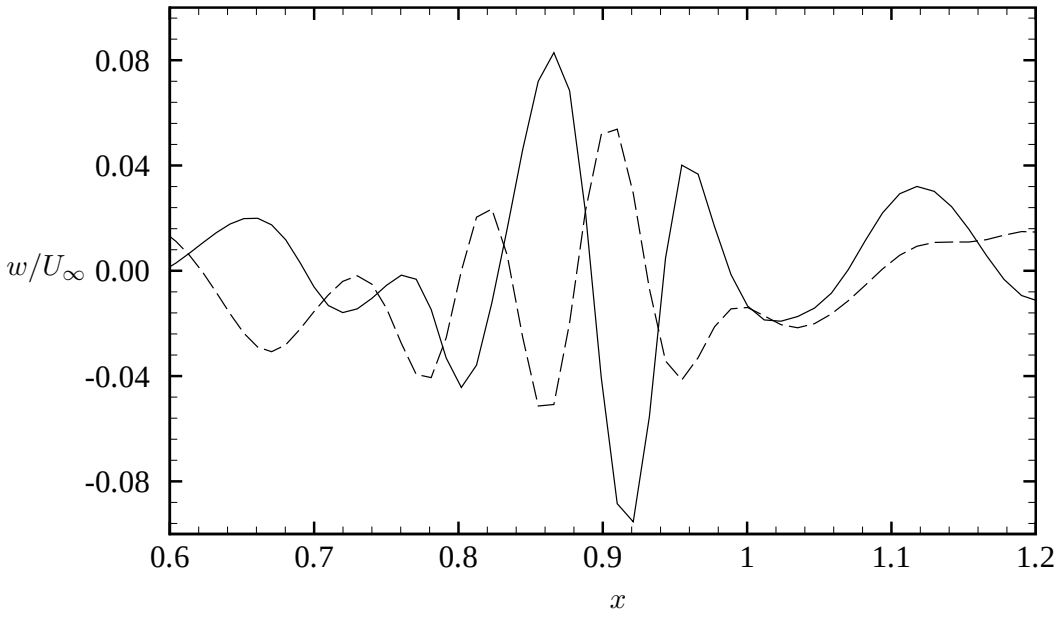


Figure 2.31: Spanwise velocity fluctuation versus streamwise coordinate along —— and — — lines from figure 2.30.

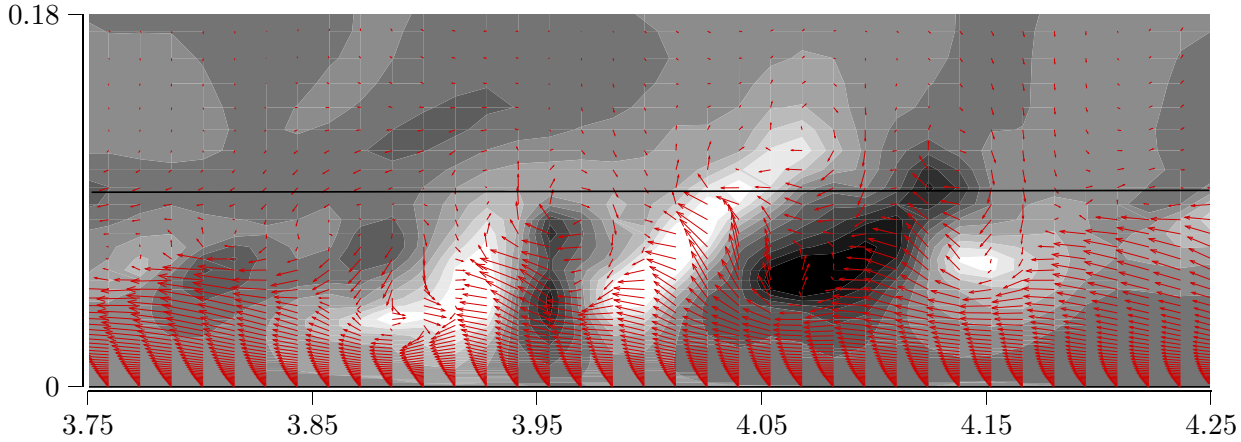


Figure 2.32: Side-view (xy-plane) of contours of spanwise velocity (levels: $-0.1 < w/U_\infty < 0.1$) and vectors of velocity fluctuation in plane. Early stages of the backward jet mechanism. The black line denotes the edge of the boundary layer.

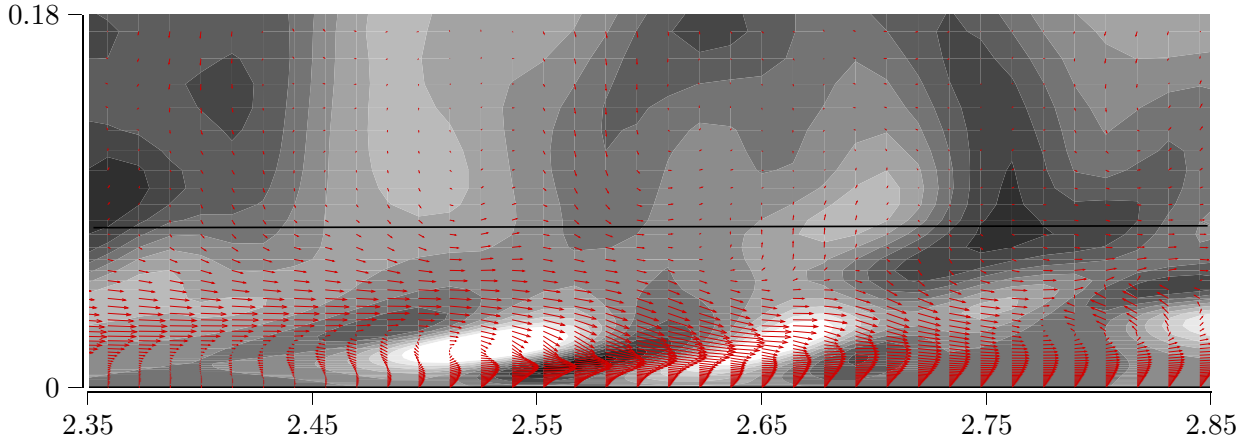


Figure 2.33: Side-view (xy-plane) of contours of spanwise velocity (levels: $-0.06 < w/U_\infty < 0.06$) and vectors of velocity fluctuation in plane. A wave-packet-like spot precursor. The black line denotes the edge of the boundary layer.

and 2.35 respectively. The span length was set to 3π in order to adequately resolve secondary flow features and to allow the statistical quantities to fully decorolate in the spanwise direction.

The computational grid shown in Figure 2.35 is for a coarse grid simulation of the Re_τ = 180 case. Four different grid levels for the Re_τ = 180 case were investigated, as shown in Table (1). The grids used for the Re_τ = 250 case are included, but will not be discussed in detail as it was demonstrated that the effect of simulating a higher Reynolds number did not warrant the additional computational cost. It was decided that the limited computational resources would be better served to ensure and prove grid independence for a case which does not exist in the literature.

Re_τ = 180 (Ro_z = 0 and 5)	Re_τ = 250 (Ro_z = 0 and 5)
C1: 192 x 32 x 64	C1: N/A
M1: 384 x 64 x 128	M1: 576 x 96 x 192
F1: 768 x 128 x 256	F1: 1152 x 192 x 384
F2: 1536 x 128 x 256	F2: N/A
F3: 768 x 256 x 256	F3: N/A

Table 2.3: Computational grids for Re_τ = 180 and Re_τ = 250.

The geometry shown in Figure 2.34 has a curvature ratio r_i/r_o = 0.33 or δ/r_c = 0.5. This ratio was selected to minimize the disparity that arises for the circumferential distribution of grid points near the inner and outer walls for a fixed channel width as the ratio ri/ro decreases. Case F1 receives the most attention. For the F1 case, 128 grid points were evenly distributed in the streamwise direction for the straight section from x = $0 \rightarrow 2\pi$. Thus, the streamwise distribution is twice as fine as the channel validation case presented in the previous chapter to account for streamwise variation of the flow properties (and hence the mean wall shear stress). Another 128 grid points were evenly distributed in the circumferential direction in the bend section. For the geometry presented in Figure 2.34, this corresponds to a streamwise spacing at the channel midspan that is coincident with the streamwise distribution in the straight section. Obviously, the streamwise distribution near the inner wall is slightly finer than the midspan distribution and, conversely, the distribution near the outer wall is slightly coarser than the midspan. Finally, another 128 grid points were evenly distributed in the straight section after the bend to the symmetry point. In the wall normal direction, 128 grid points were distributed based on the stretching function described in the channel validation section of the previous chapter. In the spanwise direction, 256 grid points were evenly distributed. The second U-bend which describes the serpentine passage has the same grid point distribution as the first U-bend.

Simulations were initially conducted for M1-180-0 which were initialized based on a converged RANS simulation for a mass flow determined from the channel flow validation simulations conducted and discussed in the previous chapter. A divergent free random perturbation field was imposed on the mean flow. The flow was then allowed to develop until a statistical steady state was reached at which point statistics were gathered for grid independence study purposes. The flowfield was then interpolated onto the C1-180-0 and C1-180-5 grids, and the F1-180-0 and F1-180-5 grids and the flow was allowed to develop further to wash out any effect of interpolation error. Once the F1 case reached a statistical steady state, the flowfield was once again interpolated onto the F2 grid to ensure streamwise resolution was of a high enough quality.

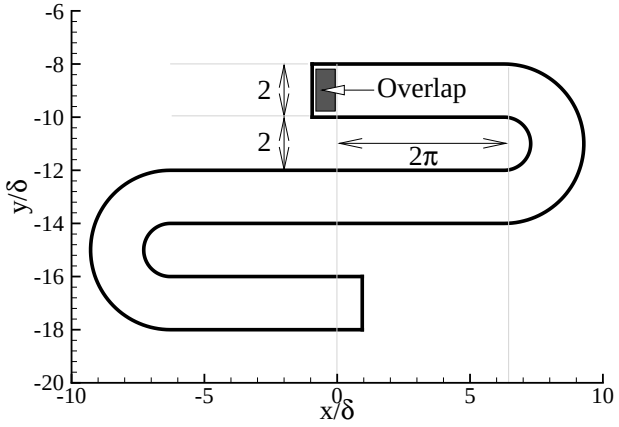


Figure 2.34: Geometry of serpentine passage.

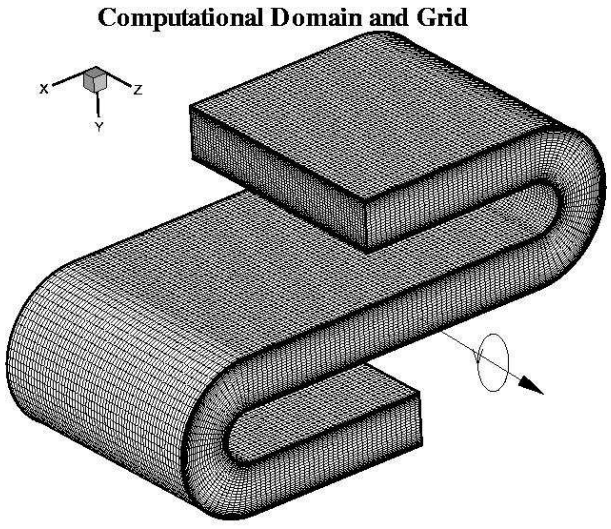


Figure 2.35: Computational grid for coarse Re_τ = 180 serpentine passage.

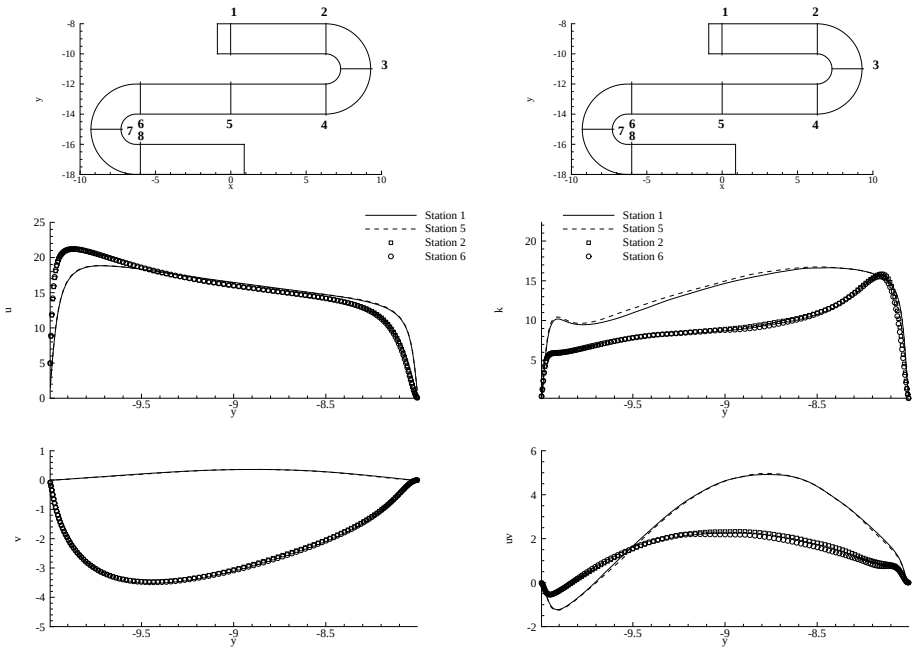


Figure 2.36: (a) Velocity component profile and (b) Turbulent kinetic energy and Reynold's stress plots demonstrating convergence for stationary case.

Some Results

The simulations for F1-180-0 and F1-180-5 were conducted for a time step of Δt = 1.25 x 10-4. Statistics were gathered over 75,000 iterations, which corresponds to 3.75 flow through times. Two different averaging procedures were utilized to determine the effect of secondary flow on the turbulent statistical quantities. Convergence was monitored based on the variation and evolution of certain statistical quantities. Figure 2.36 demonstrates the level of convergence by plotting statistical quantities at 2 station locations for the stationary case which should be identical due to symmetry.

An extensive database for the stationary and rotating serpentine passages has been developed. In order to demonstrate the complexity of the flows that were simulated the averaged and instantaneous vorticity contours for the stationary and rotating cases respectively are presented in Figure 2.37. Figure 2.38 presents the velocity vectors demonstrating the effect of spanwise rotation on the separated region. Numerous interesting flow features have been identified, the results of which can be found in the thesis of Laskowski (2005).

Hopfinger, E. J. and Linden, P. F. (1990). The effect of background rotation on fluid motion: a report on Euromech 245. Journal of Fluid Mechanics, 211:417-435.

Laskowski, G. M. (2005). Inverse Design of a Turbine Cascade Passage and DNS of a Stationary and Rotating Serpentine Passage, PhD Thesis, Stanford University.

2.2.7 Towards Large Eddy Simulation of Film-Cooling

The present study focuses on the computational investigation of the process of film-cooling which is used in turbine blades of the modern engines. The necessity for using film-cooling is dictated by the very high

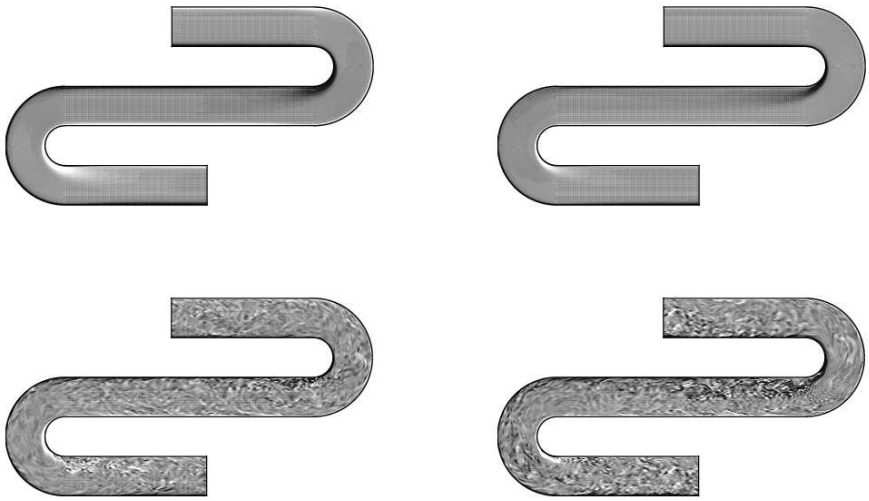


Figure 2.37: Mean and instantaneous spanwise vorticity for (a) stationary case and (b) rotating case. (White +50, Black -50).

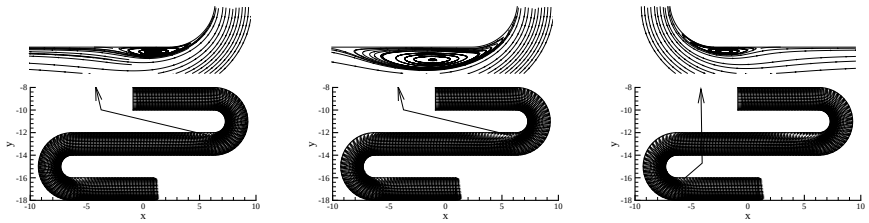


Figure 2.38: Averaged velocity vectors demonstrating the extent of separation in the vicinity of the convex curvature for (a) stationary case, U-bend 1; (b) rotating case, U-bend 1; (c) rotating case, U-bend 2.

temperature of the gases impinging on the leading edge of the turbine blades. Film cooling is one of the measures taken in order to alleviate the blade heat load. It consists of supplying cold gas from the compressor through the film-holes to the surface of the blade.

The goal of this study is to perform Large Eddy Simulation (LES) of the flow around a model turbine blade with applied film cooling in order to understand the physics of the process. The understanding of the basic phenomena will be helpful for more efficient film cooling designs.

It has been shown by many authors [45, 46, 47] that the details of the flow in the vicinity of the film-cooling hole exit depend greatly on the flow inside the hole. These details in turn affect the overall features of the flow including film-cooling performance. Therefore in order to simulate the problem accurately one needs to include the film-cooling hole as well as the plenum into the simulation.

In order to perform this task a special computational tool has been developed. It consists of two different numerical codes.

1. The first code is the fully compressible code written in curvilinear coordinates by Z.Xiong [48]. It is used for the simulation of the exterior flow around the turbine blade.

2. The second code is the low Mach number code written in cylindrical coordinates by C.Pierce [49]. It is well suited for the simulation of the flow inside the plenum and film-hole, especially due to its higher efficiency compared to compressible code in the region of the slow moving plenum flow.

The coupling of the two codes has been performed using MPI so that the codes run synchronously in time exchanging the boundary conditions at every time step. The boundary conditions supplied from the compressible to low Mach number code are three components of velocity, temperature and gradients of pressure on the interface boundaries, as well as the value of the background thermodynamic pressure. For the compressible code, the three components of velocity and temperature are supplied from the low Mach number code in the region of the film-hole exit, while density is evaluated from the continuity equation. The solid parts of the blade surface are treated with the usual no-slip boundary conditions.

The flow in the overlap region is calculated by both codes. The values for the parameters of the external flow in the region of overlap are taken from the more accurate compressible code. However, an assurance that the solutions of two codes don't drift too much from each other is needed for the convergence of the coupled solution. The closeness of solutions has been thoroughly tested on the model problem of a zero-circulation vortex moving from the compressible code domain into the low Mach number code domain. It was found that the parameters distribution of the moving vortex reproduced by two codes was in a very good agreement. The error was of the order of M^2 which is inherent to the different sets of equations being solved. Since the region of the overlap for film-cooling problem falls into the blade boundary layer where effective Mach numbers are quite small it is believed that the difference in solutions will not be the main source of errors.

There are two groups of simulations which are planned in the near future.

1. The first group of simulations are a validation study which consists of the detailed comparison of the computations with experiment. In order to be sure that the numerical tool is working well it would be necessary to test it on the geometry where the features of the flow are fairly well known. Jet in a crossflow is a good candidate for such a detailed comparison, since this type of flow has been investigated thoroughly by many researchers. Experiments of Gopalan et al.[50] were chosen as the benchmark for comparison in the present study for two reasons. First, it documents very accurately

the three-dimensional velocity field as well as turbulence statistics at multiple locations. Second, it uses non-dimensional parameters close to those found in the real film-cooling environment. Thus, the blowing ratio in the experiments is set to values $B = 0.5$ and $B = 1$ which is characteristics of film-cooling applications. It is worth noting that the majority of other jet in crossflow experiments usually use much higher blowing ratios $B > 2$.

2. The second group consists of simulations of the real film-cooling configuration. These simulations will model the turbine blade with the film-cooling holes in the vicinity of the leading-edge region. The experiments chosen for comparison would be either experiments of Polanka et al.[51] or experiments of Ardey et al.[52], since they are the only experiments found in literature which document the three-dimensional structure of the flowfield and not just the averaged parameters on the surface of the blade. These simulations model the jet in a crossflow in the presence of the stagnation region and where effects related to the curvature of the surface are important. This type of flow is much less investigated and understood and this is where the present study could provide some useful contribution into the problem.

Both groups of simulations are currently underway. Sample pictures of a laminar 2D flowfield for these two types of geometries are shown at Fig. 2.39.

The top picture, Fig. 2.39, documents the vertical velocity for the jet in crossflow simulations. The jet is issued perpendicular to the flat surface with a blowing ratio $B = 1$ and Reynolds number $Re_d = 500$ based on cross-flow velocity and jet diameter. For the LES simulations of this problem we will need to reproduce the turbulent boundary layer for the initial and inflow boundary conditions since it is typical of film-cooling applications and it is a feature of the experiment [50]. Separate LES of a turbulent boundary layer will be performed for this purpose.

The bottom picture, Fig. 2.39, shows the vertical velocity flowfield obtained when the jet is inclined at 35° with respect to the main stream, $B = 1$, $Re_d = 500$. The semi-elliptical model of the turbine blade is used in this simulation. This figure also demonstrates the capability of the code to handle non-cartesian surface geometries as well as arbitrary intersection of the grid-lines of two codes in the region of overlap which will be needed for the leading edge region simulation.

Finally, Fig. 2.40 shows the comparison of profiles for a flat plate geometry calculated by the compressible code and low Mach number code taken along the line $y = 0.1$ in the region of overlap (see Fig. 2.39, top). Here U stands for streamwise velocity and V - for wall-normal velocity. It can be seen that the difference between the solutions is indeed small.

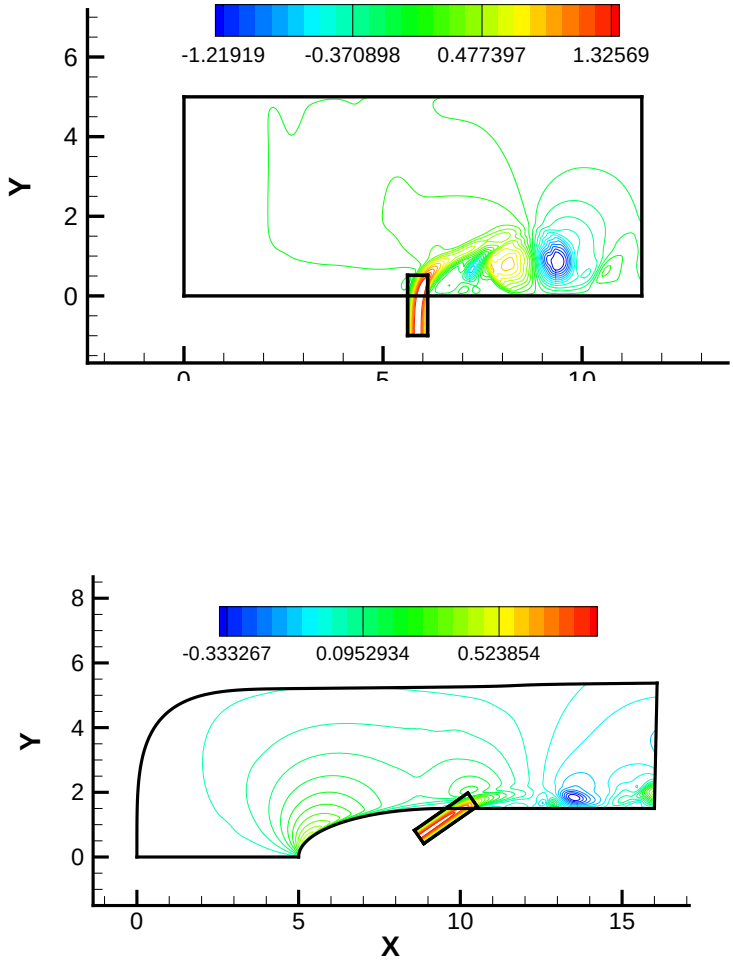


Figure 2.39: Vertical velocity. Top: flat plate geometry, bottom: turbine blade geometry

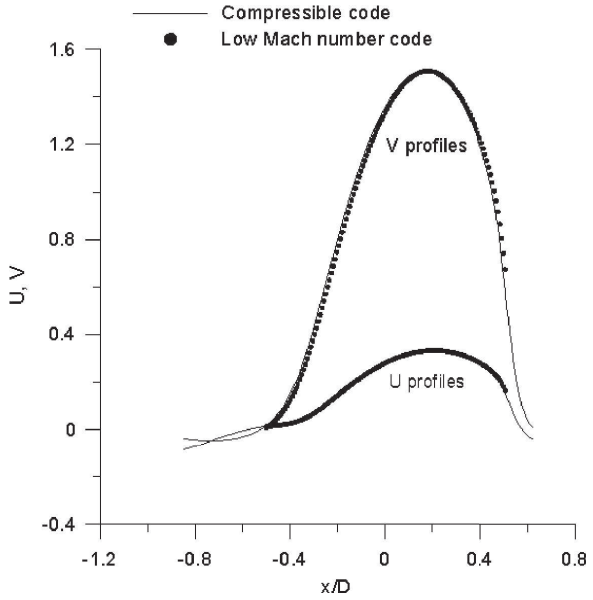


Figure 2.40: Velocity profiles for jet in crossflow calculation

2.3 Simulations in Support of Validation and Verification Efforts

A number of simulations have been conducted to assess the accuracy of the modeling in TFLO2000. This section presents a number of these test cases. This section is work in progress and will be updated in the final version of this report.

2.3.1 Selected Validation Test Cases

Fin-plate junction

In this test case a supersonic boundary layer with a free stream Mach number of 2, collides with a rounded fin. This produces a complex shock-wave boundary-layer interaction, where multiple horse-shoe vortices are present. Experimental data for four fin configurations and three flow incidence angles have been obtained by Barberis and Molton [53]. Here only the normal fin with zero degree incidence is considered.

Some uncertainty arises due to the lack of information of the upstream boundary layer. The only thing mentioned in [53] is the boundary layer thickness upstream of the interaction. The flow conditions are summarized in table 2.4

M_∞	α, $deg.$	Re/m	δ mm	P_{tot} Pa	T_{tot} K
1.97	0	$9.27 \cdot 10^6$	15	$80 \cdot 10^3$	330

Table 2.4: Conditions for the rounded fin-plate junction

To obtain a boundary layer profile these conditions are used as inflow conditions for a 2D flat plate

computation. The distribution of the velocity, density, pressure and turbulence variables at the position where the boundary layer thickness is 15 mm was taken as inflow conditions for the actual computation, 150 mm upstream of the fin leading edge. This procedure has been applied for all turbulence models used in this study.

A four block mesh with a total of 6 million cells has been used. The normal spacing of the cells adjacent to the viscous walls is such that the maximum $y+$ value of the first cell center is less than 0.5. A geometrical stretching factor of 1.1 is used, such that the boundary layer is resolved well. To avoid an excessive number of points an inviscid wall boundary condition is used for the top of the fin. This part of the flow does not have any influence on the interaction in front of the fin and is only modeled to obtain a well-posed problem.

The JST scalar dissipation scheme [54] has been used for the advective fluxes of the mean flow equations whereas the advective parts of the turbulence equations have been discretized by a first order upwind scheme. A three level W multigrid cycle in combination with a 5 stage Runge-Kutta scheme with local time stepping accelerates the convergence to the steady state. Convergence histories for the turbulence models used are shown in figures 2.41 and 2.42. From the former it is clear that all models convergence very well for this

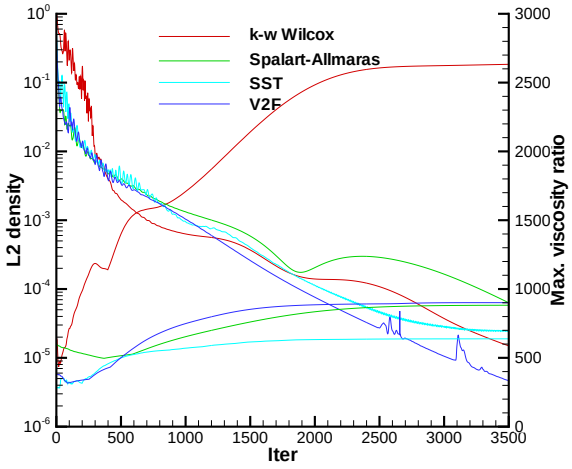


Figure 2.41: Convergence histories for the L2 norm of the density residuals and the maximum eddy viscosity ratio for the fin-plate junction

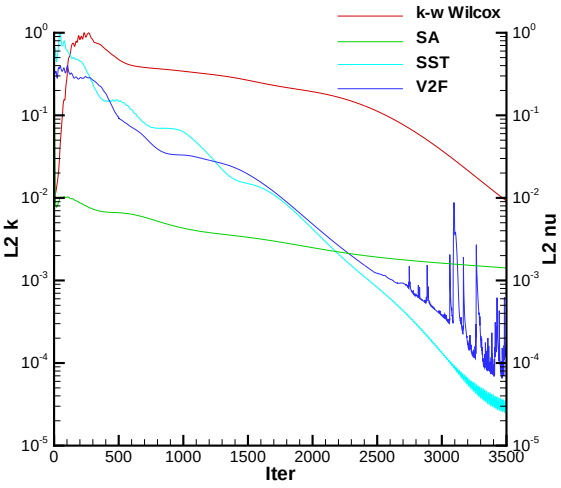


Figure 2.42: Convergence histories for the L2 norm of the $\tilde{\nu}$ transport variable for the Spalart-Allmaras model and the turbulent kinetic energy for the other models.

test case. The relatively high number of iterations needed is caused by the fine grid in combination with very high aspect ratio cells. Note that the standard $k-\omega$ model produces a much higher maximum value of the eddy viscosity than the other models. This is not only true for the maximum value; in general the $k-\omega$ model produces higher values of the eddy-viscosity than the other models, which results in a shorter separation length, see figure 2.45

Figure 2.43 shows the surface pressure distribution for the SST model and the Mach number distribution on the symmetry plane as well as streamlines can be found in figure 2.44. Although the solutions for the other turbulence models are significantly different, they show a similar behavior and are therefore not shown.

The skin friction pattern on the plate just in front of the fin are given in the figures 2.45 to 2.48. The

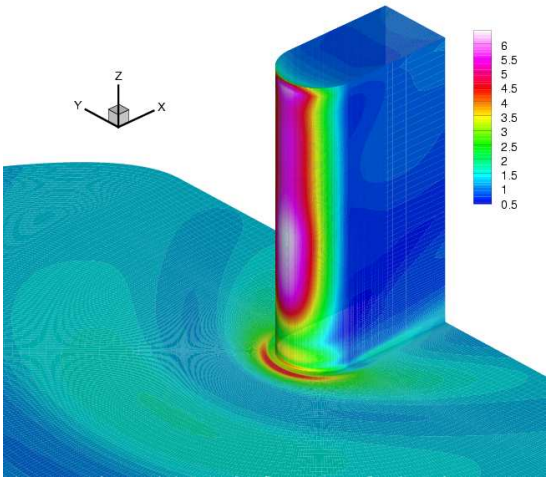


Figure 2.43: Nondimensional surface pressure distribution, nondimensionalized by the free stream static pressure, for the SST turbulence model.

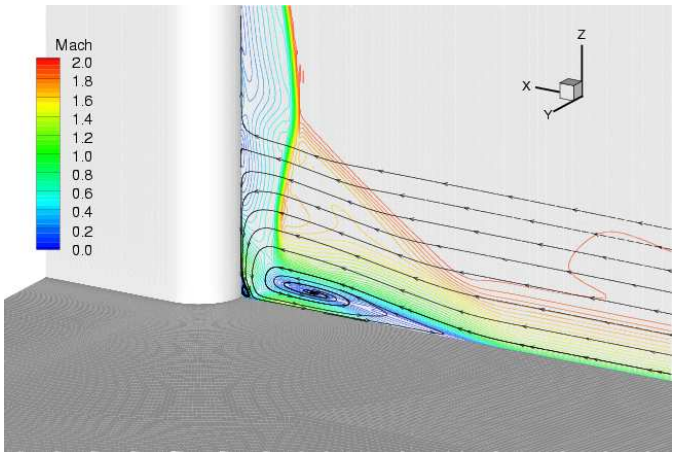


Figure 2.44: Mach number distribution and streamlines in the symmetry plane for the SST turbulence model.

corresponding experimentally measured pattern is shown on the bottom part of these figures. The objective of the Turbomachinery Group is to develop state-of-the-art capabilities for the simulation of flows in the rotating components of jet engines (compressor, turbine, secondary flow system) using the Unsteady Reynolds-Averaged Navier-Stokes (URANS) equations as a flow model. Our measures of success are fundamentally based on the ability to decrease the solution times by significant factors (10 X to 20 X) in comparison with other existing software and to increase the modeling accuracy via the introduction of more advanced turbulence and transition models. Although our target application is in turbomachinery flows, the software that we have been developing is completely generic and applicable to a wide range of other unsteady flow applications. We try to make this point in this Chapter by showing a number of validation test cases in external aerodynamics and throughout a very wide flow speed range (from incompressible flow to supersonic flow). It is clear that none of the turbulence models considered predict the correct separation pattern; the experimental results show four vortices while the computations give only two. This observation was also made by Leschziner [55], in which is concluded that only second order closure models are able to capture the correct separation/reattachment pattern.

Finally, figures 2.49 and 2.50 show the wall pressure distribution along the lines $y = 0$ mm and $y = 15$ mm, where y is the coordinate normal to the symmetry plane. The separation length itself is severely underpredicted by the standard $k-\omega$ model, Spalart-Allmaras and V2F give very similar results and slightly underpredict the separation length, whereas SST shows an overprediction. It must be noticed that both Spalart-Allmaras and SST are carefully tuned for aerodynamic flows and they may perform worse for other type of flows. The V2F model, and also $k-\omega$ is a much more general model with a much wider range of applications.

The conclusion for this test case is that none of the turbulence models considered is able to predict the flow pattern of this very complicated shock-wave boundary layer interaction correctly. The most likely cause is that all the models here linear eddy-viscosity models, V2F included, and only anisotropic models

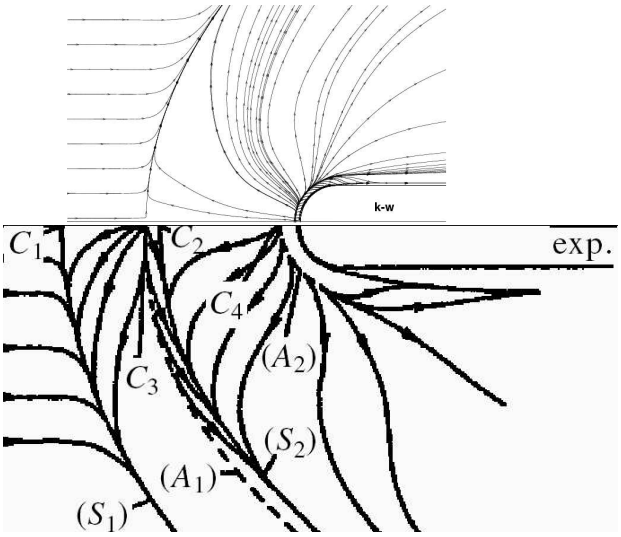


Figure 2.45: Computed and experimental skin friction pattern, standard $k-\omega$ model

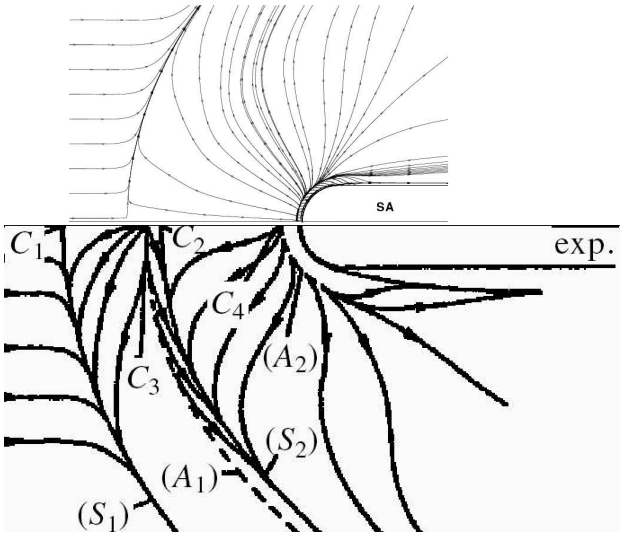


Figure 2.46: Computed and experimental skin friction pattern, Spalart-Allmaras model

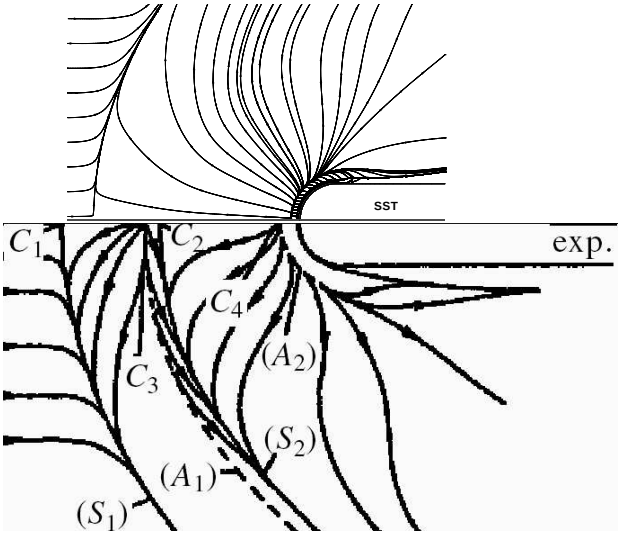


Figure 2.47: Computed and experimental skin friction pattern, SST model

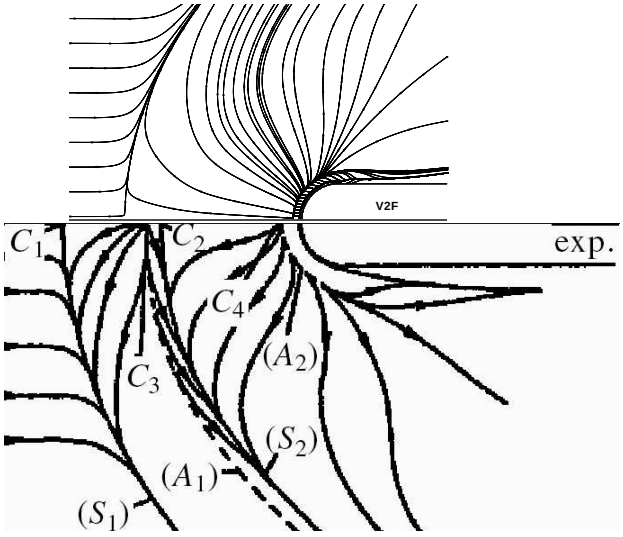


Figure 2.48: Computed and experimental skin friction pattern, V2F model

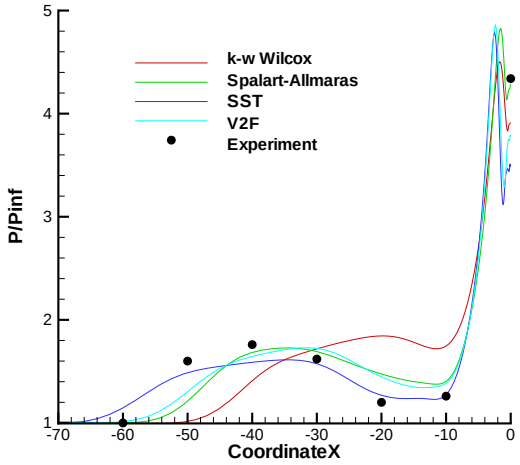


Figure 2.49: Wall pressure distribution along the line $y = 0$ mm

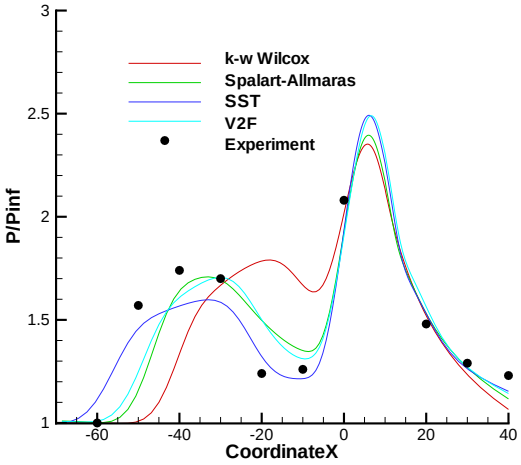


Figure 2.50: Wall pressure distribution along the line $y = 15$ mm

are able to predict the correct pattern [55]. SST, Spalart-Allmaras and V2F show a similar performance; $k - \omega$ clearly gives the worst results for this test case.

From a numerical point of view it is very encouraging that all turbulence models converge very well and are extremely stable.

Wing tip vortex

The goal of this test case is the accurate prediction of a wing tip vortex generated by a low aspect ratio wing. Extensive experimental data is available [56]. To avoid possible non-uniformities from the inflow and outflow boundary, we extended both upstream and downstream the computational domain by a court length compared to what was used in [56]. The surface grid and the computational domain is shown in figure 2.51. The normal spacing of the cells adjacent to the viscous wall boundaries is such that $y^+ < 0.5$ and therefore integration to wall is used for the turbulent quantities. To avoid an excessive number of cells the viscous effects on the wind tunnel walls are neglected, i.e. inviscid wall boundary conditions are applied there.

Preliminary results have been obtained for the SST and V2F turbulence models. To minimize the numerical dissipation Roe's approximate Riemann solver [57] has been used in combination with a second order reconstruction scheme. The Van-Albeda limiter was chosen to avoid overshoots in this reconstruction. The convergence histories for both turbulence models are shown in figure 2.52 and 2.53. Relatively many iterations are needed to reach convergence and especially the turbulent kinetic energy drops only an order of magnitude in 6,000 iterations. The reasons are the large computational domain discretized with a fine grid and the presence of a thick trailing edge. However the main flow features are present and some conclusions can be drawn.

Figure 2.54 shows the turbulent kinetic energy in planes $x =$ constant, while figure 2.55 shows the eddy viscosity distribution in the same planes. From these figures it is clear that the V2F models performs much better in the vortex region. The level of turbulence in the core is much less than for SST, which already has

a reduced eddy viscosity compared to standard $k-\omega$ because of the vorticity limiter, and consequently V2F preserves the vortex much better.

The method of Jeong & Hussain [58] has been used to visualize the tip vortex. The method is based on the eigenvalues of the symmetric tensor $S2+\Omega 3$, where S and Ω are respectively the symmetric and antisymmetric parts of the velocity gradient tensor. A vortex is defined to be the connected region with two negative eigenvalues of $S2+\Omega 2$. This definition is equivalent to the requirement that the second eigenvalue λ_2 is negative inside the vortex. The surface where λ_2 is zero is plotted in figure 2.56 for the V2F model.

2.3.2 NASA Stage 35 and Rotating Stall

It is widely known that flow instabilities of turbomachines incur significant degradation to the performance of the engine as well as actual physical damage to the engine itself. When surge develops as a result of stall, the only known remedy is to restart the engine. Consequently, predicting the inception of stall, a precursor of surge, has received much attention, but the phenomenon is poorly understood.

In past decades, there have been a number of key developments to address these problems. Among them, Moore and Greitzer suggested a simplified model of the whole turbomachine. With this simple nonlinear model, see Figure 2.57, the zero-th order effects of both stall and surge are captured.

Simplified Nonlinear Model

In the development of the model, Moore and Greitzer made use of available experimental compressor data and devised simple formula relating the mass flow to the total pressure rise, through both rotors and stators.

$$\frac{\Delta P}{\frac{1}{2}\rho U^2} = F(\phi) - \tau\frac{d\phi}{dt}$$

where $\phi(=\frac{C_x}{U})$ is the axial velocity coefficient at the compressor,
$F(\phi) =$ the axisymmetric performance characteristic,
and $\tau =$ time constant associated with the internal lags in the compressor.

Through the stator and the rotor,

$$(\frac{d\phi}{dt})_{stator} = \frac{U}{R}\frac{\partial\phi}{\partial\xi}$$

$$(\frac{d\phi}{dt})_{rotor} = \frac{U}{R}(\frac{\partial\phi}{\partial\xi} + \frac{\partial\phi}{\partial\theta})$$

For an N-stage compressor,

$$\frac{P_E - P_1}{\rho U^2} = NF(\phi) - \frac{1}{2a}(2\frac{\partial\phi}{\partial\xi} + \frac{\partial\phi}{\partial\theta})$$

where $a \equiv \frac{R}{N\tau U}$

With other simplified assumptions, the whole turbomachine can be described by three ordinary differential equations,

$$\frac{d\Psi}{d\xi} = \frac{W/H}{4B^2}[\frac{\Phi}{W} - \frac{1}{W}F_T^{-1}(\Psi)]\frac{H}{l_c}$$

$$\frac{d\phi}{d\xi} = [-\frac{\Psi - \psi_{c0}}{H} + 1 + \frac{3}{2}(\frac{\Phi}{W} - 1)(1 - \frac{1}{2}J) - \frac{1}{2}(\frac{\Phi}{W} - 1)^3]\frac{H}{l_c}$$

$$\frac{dJ}{d\xi} = J[1 - (\frac{\Phi}{W} - 1)^2 - \frac{1}{4}J]\frac{3aH}{(1 + ma)W}$$

whose results are presented in Figures 2.58 and 2.59, rotating stall and surge respectively.

TFLO2000 and Stage 35

With current advances in CFD, new tools are available for the simulation of these phenomena. Presently, TFLO2000 is capable of running an unsteady simulation on Stage 35 where stall effects are being investigated. Figure 2.60 shows the surface mesh for the NASA Stage 35 and Figure 2.61 shows the unsteady surface data after it has reached a periodic steady state.

Even after the rapid technological advancement in computer science, direct simulation of the entire turbomachine would require a sizeable amount of time. Using a simplified model but with CFD data on portions of it that need higher accuracy would allow us to conjecture the overall behavior of the turbomachine in a reasonable turnaround of computational time frame. This effort to modify existing simplified models is currently under way and will be reported on at a later time.

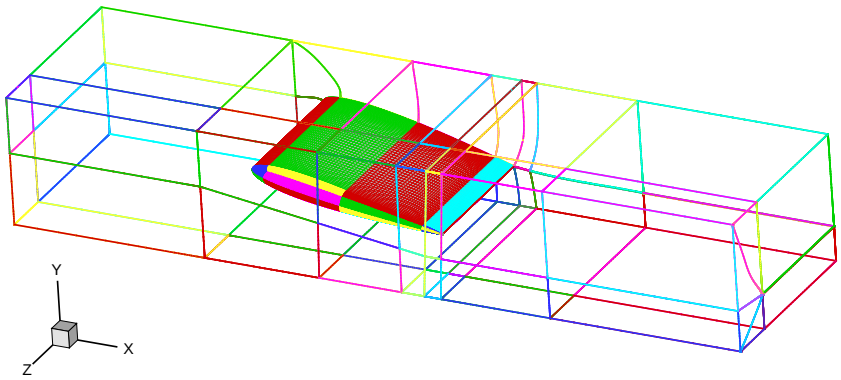


Figure 2.51: Computational domain and wing surface grid for the wing tip vortex case. Corresponding volume grid consists of 56 blocks and 7.2 million cells.

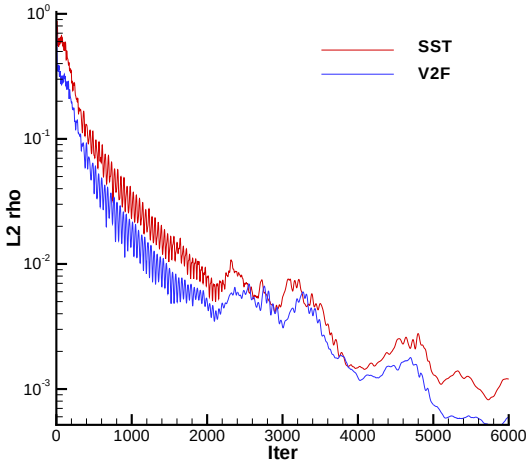


Figure 2.52: Convergence histories for the L2 norm of the density residuals; 3 level W-cycle in combination with a 5 stage Runge-Kutta time integration scheme.

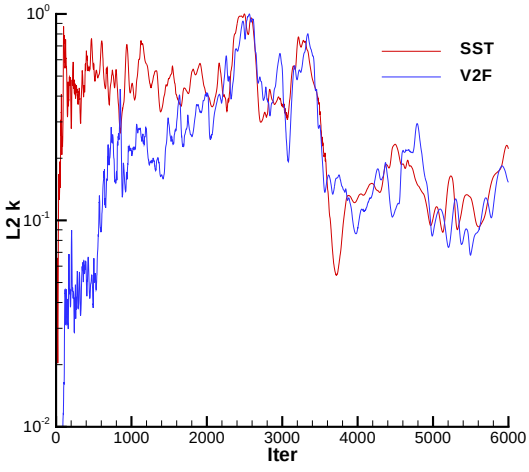


Figure 2.53: Convergence histories for the L2 norm of the turbulent kinetic energy; 3 level W-cycle in combination with a 5 stage Runge-Kutta time integration scheme

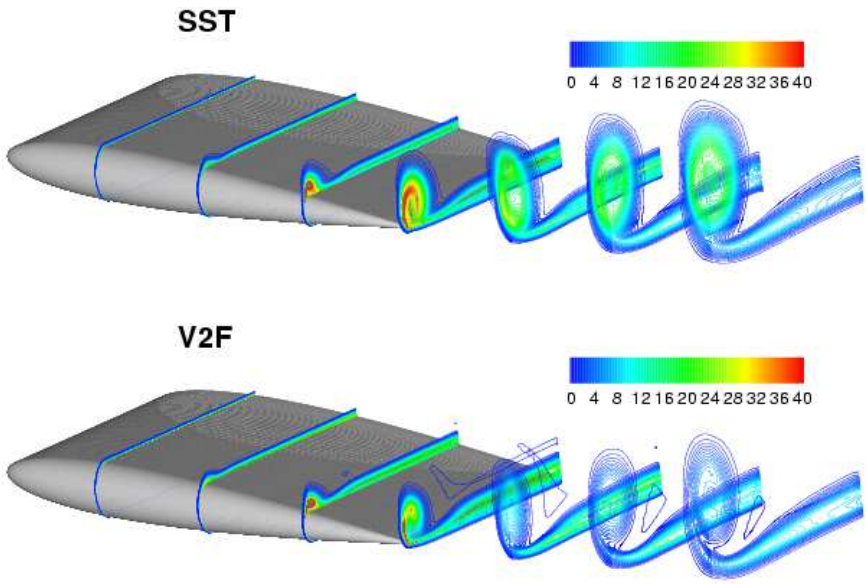


Figure 2.54: Turbulent kinetic energy isolines in planes x = constant for the SST and V2F turbulence models.

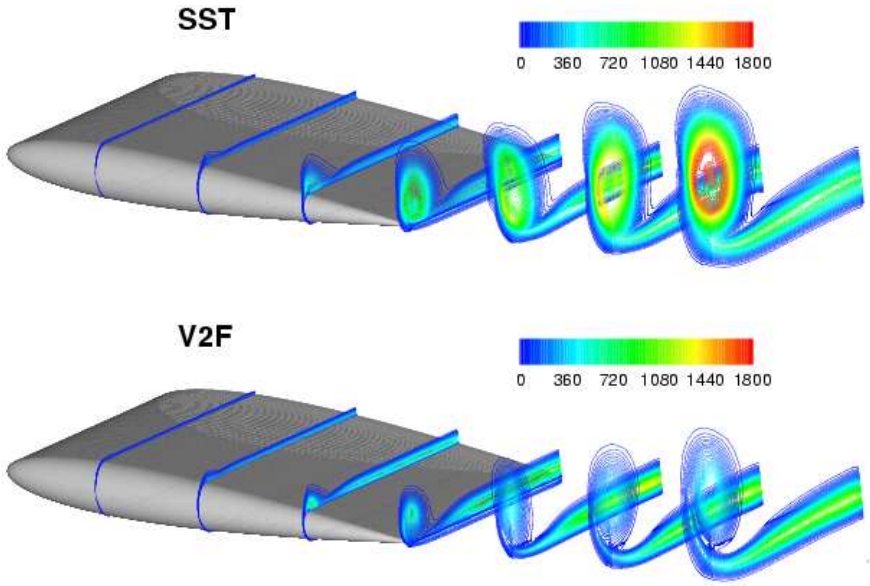


Figure 2.55: Isolines of the eddy viscosity ratio in planes x = constant for the SST and V2F turbulence models.

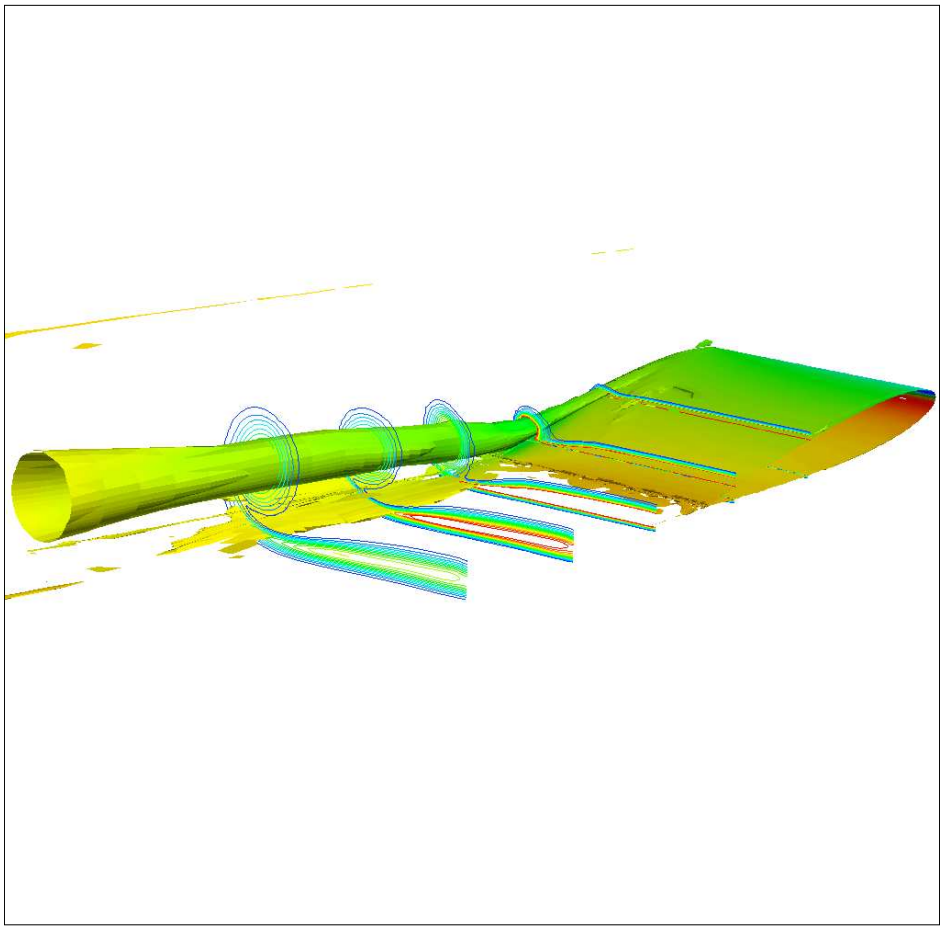


Figure 2.56: Isosurface of $\lambda_2 = 0$ for the V2F model. The color of this isosurface visualizes the pressure. The domain cuts show contourlines of the turbulent quantity $\overline{v2}$. Color red and blue identify high and low values, respectively.

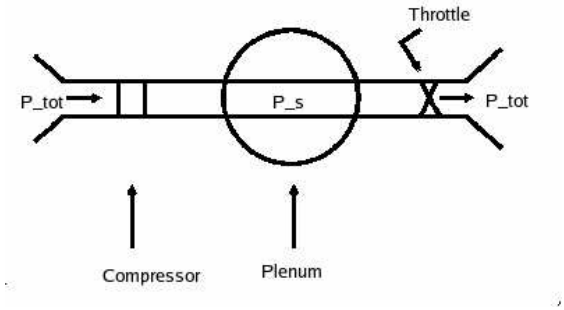


Figure 2.57: Schematics of Simplified Model

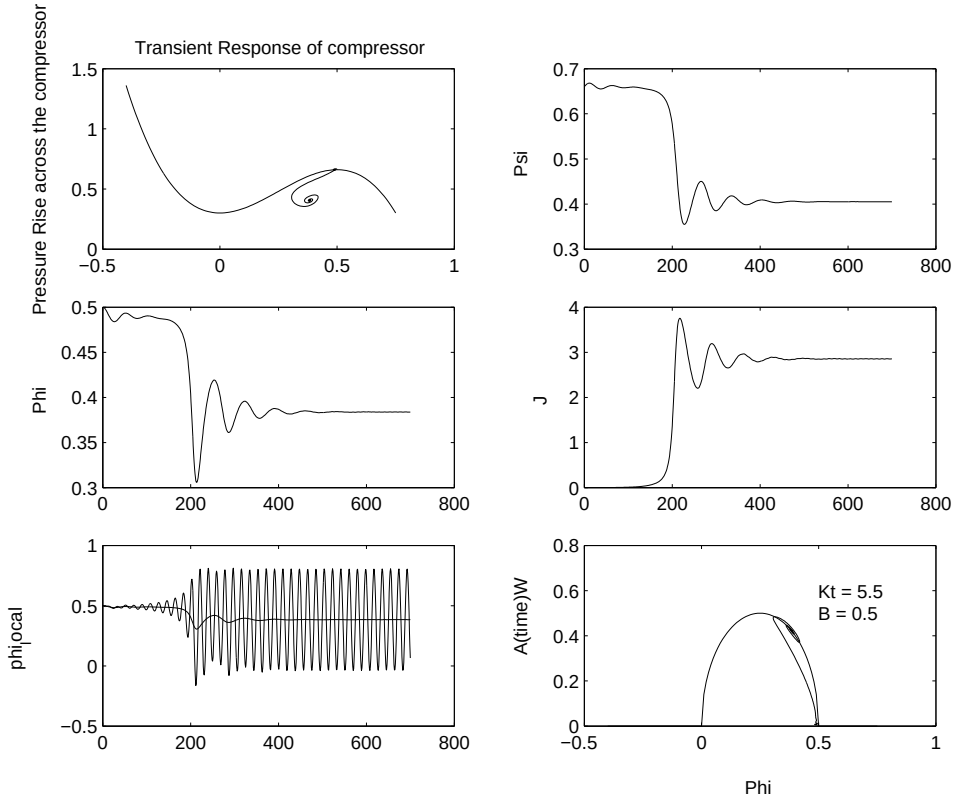


Figure 2.58: Calculation of Stall

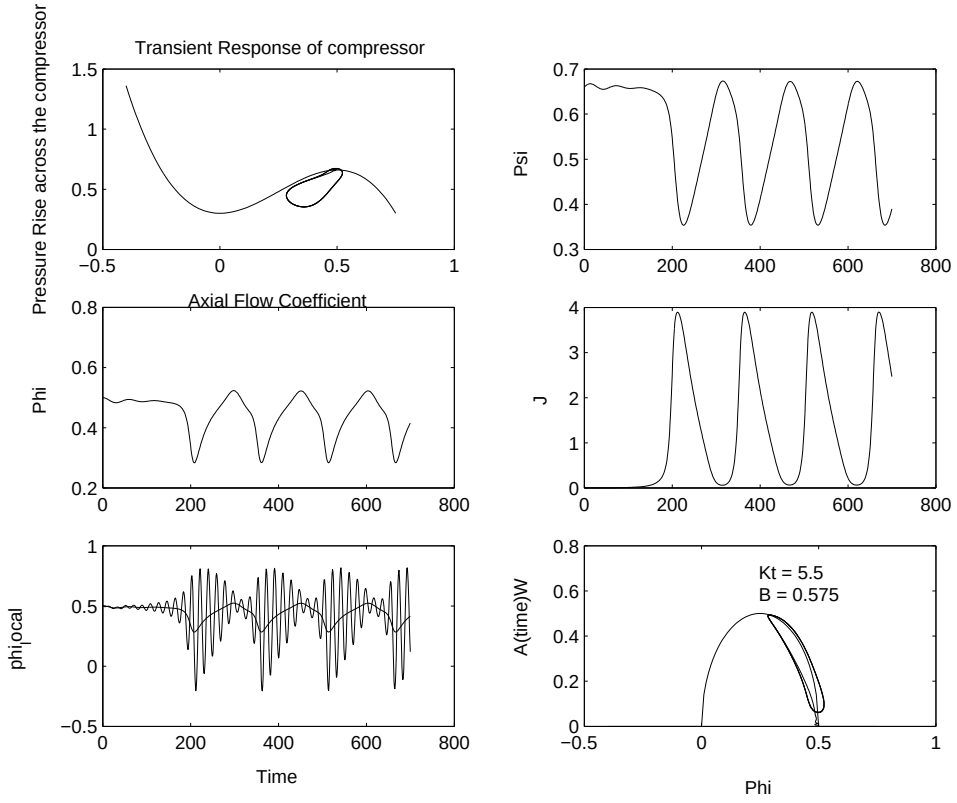


Figure 2.59: Calculation of Surge

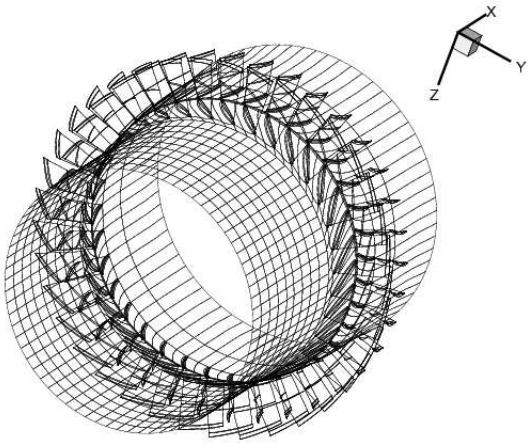


Figure 2.60: Stage 35 Mesh

2.3.3 Python Software Integration and Aeroelastic Simulations

Python Integration

In a large project such as ASC where people are working in various disciplines somehwat independently of each other the major challenge has been that of integration while maintaining their independence. The need to couple analyses of different disciplines revealed the inconvenience of "gluing" and scripting different codes that have been written in different programming languages (Fortran 77/95, C, C++, Java.) In addition to differing in the language, these codes also have unique interfaces and data structures. For example, an aeroelastic code might utilize an existing CFD code written in Fortran 90/95 for the aerodynamic analysis, Nastran for the finite-element analysis of the structure and a data visualization program written in C or C++ with its own API for displaying the results. For this interfacing, we considered a number of programming languages and selected Python [59]. The selection of Python and its advantages relative to other programming languages are discussed in the next section and in Reference [60].

Why Python?

Most of our high-fidelity analysis programs are written in Fortran. Large-scale scientific computing requires the use of a compiled language for acceptable floating-point performance, and for this reason, Fortran is the language of choice for many numerical applications. While Fortran 90/95 features such as dynamic memory allocation and data structures are welcome additions (we use them extensively in our programs), Fortran lacks some of the features of other current languages, such as object-oriented programming.

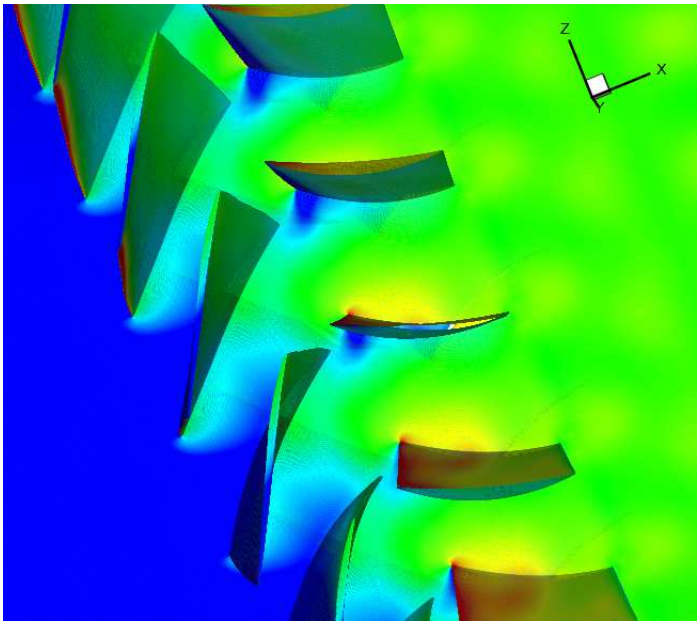


Figure 2.61: Stage 35 Surface Pressure Distribution

Python uses a concise syntax that results in programs that are easy to read and has all the features of a modern object-oriented language without the annoyances of a compiled language. Python runs on many different computers and operating systems (Windows, Mac-OS, many brands of Unix, OS/2) and provides a large standard library that supports many common programming tasks such as connecting to web servers, regular expressions, and file handling. Perhaps a more important feature of Python is good support for compiled libraries. Instead of moving the whole project to a compiled language like C++ or Fortran, one can re-write just the computationally intensive parts in compiled code. This approach results in much faster development than a compiled language alone: you get the robustness of a scripting language with the speed of a compiled program. This is also a good way to utilize existing Fortran subroutines without writing the whole program in Fortran.

Python Tools

In addition to the useful features we just described, Python also provides a number of tools that are very useful in our work. The scientific computing module, `Numeric`, the parallel computing module, `pyMPI`, and the automatic Fortran wrapper, `f2py` are the three essential tools that make Python even more attractive for our purposes.

Numeric-py

`Numeric` is a popular and powerful compiled library for Python. It can provide math operations on large matrices at speeds close to those of the compiled languages. One can do operations on large arrays quickly, and with very simple program syntax. In addition, Numeric includes a large number of common matrix

operations.

pyMPI

The ability to introduce parallel computing codes into the pyMDO framework is essential to realize the vision of a high-fidelity, integrated design environment. `pyMPI` is an MPI enabled Python interpreter that provides parallel programming support on distributed parallel machines [71].

Automatic Fortran Wrapper

Python provides good support for compiled libraries. Python itself is written in C, and extending Python to access code written in C and C++ is easy to do by hand and even easier with tools, such as SWIG [62], that automate the process. Since it is possible to wrap Fortran programs with C, we can also create libraries whose source code is written in Fortran, but that can be imported into the Python interpreter as a module. Doing this by hand is a laborious task that is prone to errors.

Fortunately, there are automatic interface generators for Fortran that require the programmer to have little or no knowledge of the coding details necessary to transfer data and to call functions between Python and Fortran. There are two such tools: *Pyfort* [64] and `f2py` [72, 73].

We decided to use `f2py` since its features are better suited to our efforts. Since `f2py` provides direct access to common block and module variables, there is no need to change the Fortran source, making the wrapping process even easier.

The way `f2py` works is summarized in Figure 2.62. Given a set of Fortran source files, `f2py` generates a signature file with extension `pyf` that contains all information about the data, functions and their arguments in the Fortran source. This file can be edited by the user for better control, and `f2py` can be instructed to just read this file, rather than generate it automatically.

Given this signature file, `f2py` then creates the C code wrappers and compiles the final module (`.so` file) that can be directly imported into the Python interpreter.

We simplified the process of wrapping Fortran programs even further by developing `pymdobuildutil`, a utility that has a similar interface as Python's `distutils`. The developer of an extension module only needs to write a file which defines various aspects of the module and how to build it: the name or a signature file, locations of source code, and which libraries are necessary to link. Some knowledge of how `f2py` works is necessary but `pymdobuildutil` automates many of the routine tasks. This utility compiles the Fortran source, calls `f2py` to create the wrappers and then compiles the Python module, all with a single command.

2.3.4 Aeroelasticity

The ASC Center at Stanford intends to develop a high fidelity simulation of an aircraft engine. Perhaps one of the most challenging task is the High-Fidelity simulation of Aeroelastic phenomena occurring in a jet engine as it involves the unsteady simulation of the flow field as well as that of the structure. The accurate simulation of these phenomena is not only of academic interest but also will help in prediction of potentially catastrophic events before the system goes into production. In the following sections a brief overview of the solvers and the methodology used to couple them is described.

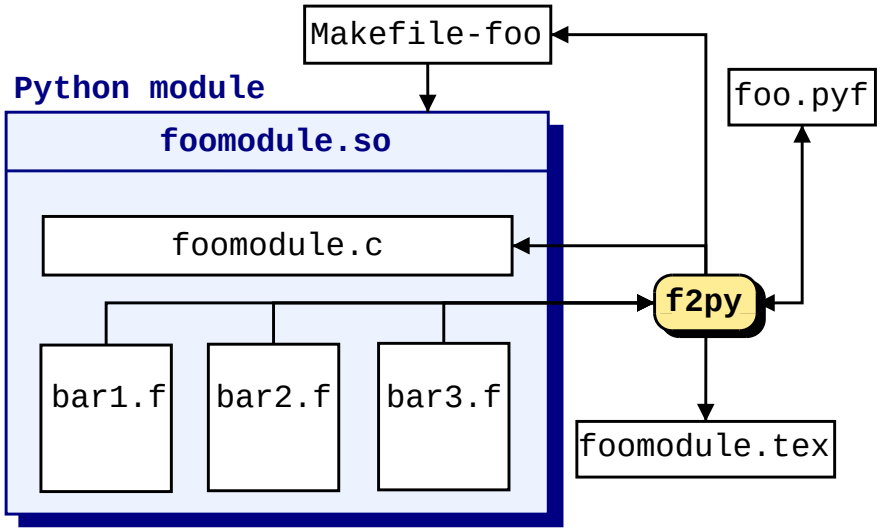


Figure 2.62: The wrapping process with `f2py`.

Aerodynamics

For aerodynamic analysis we use TFLO2000. TFLO2000 is a state-of-the-art flow solver that contains much of the functionality of our old, Fortran 77 multiblock flow solver, FLO107-MB [74]. TFLO2000 contains extensions for turbomachinery applications needed for the Stanford ASC project and includes substantial new capabilities in the pre-processor, parallel implementation, turbulence models, and baseline algorithms. It uses a finite-volume formulation, a modified Runge–Kutta scheme, a central-difference discretization augmented with artificial dissipation terms, and the multigrid solution procedure to enhance convergence to a steady state. TFLO2000 incorporates a number of turbulence models including Spalart–Allmaras, $k-\omega$, SST, and v^2-f. A DD-ADI (Diagonally-Dominant Alternating Direction Implicit) approach is used to converge the turbulence equations and no multigrid is used for these. TFLO2000 also uses the CGNS standard for I/O purposes and can be run on a variety of parallel computers with demonstrated scalability up to thousands of processors.

The Python interface to TFLO2000 is provided by a class called `FlowSolution`. The class was designed as a base class that can wrap any type of CFD solver. A new class, `TFLOSolution` inherits all the data and methods from the superclass, but redefines and extends the class in order to specialize it for the specific case of TFLO2000. The flow solver and the additional class definitions are encapsulated in a module we call *pytflo2000*.

The `FlowSolution` contains one object: the computational mesh that is an object of the `Mesh` class. A diagram representing this hierarchy, attributes and methods for this class is shown in Figure 2.63.

Methods in the flow class include those used for the initialization of all the class components (`__init__` functions, which are not shown in UML diagrams), as well as methods that write the current solution to a file. In addition, we have added methods to this class that allow us to retrieve surface loads, store deflected and perturbed surface shapes, and that can be used to advance the flow solution for a specified number of iterations. As new integrated applications are constructed, additional functionality is added to these classes (which may require modifications/enhancements to the baseline Fortran 90/95 code). At the time of this

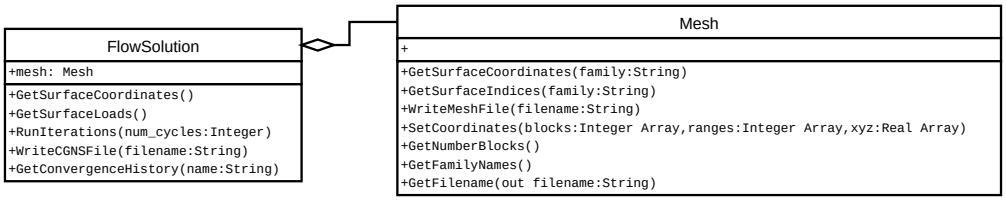


Figure 2.63: UML diagram for the `FlowSolution` class.

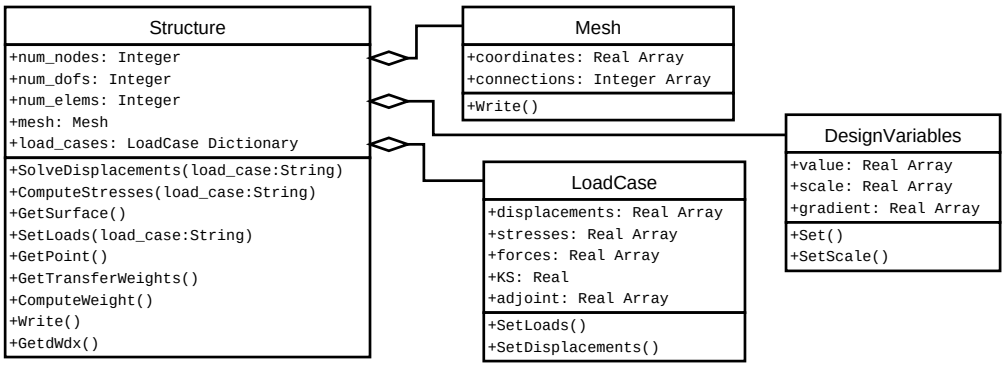


Figure 2.64: UML diagram for the `Structure` class.

writing, a number of additional methods were already available for these classes.

Structures

The structural analysis program we use is the Finite Element Analysis Program (FEAP), written by Prof. Taylor at UC Berkeley [77], and is a general purpose finite-element package for the analysis of complex structures. The program includes the capability to construct arbitrarily complex finite-element models using a library of one-, two-, and three-dimensional elements for linear and nonlinear deformations. In addition, a number of material models (such as isotropic, orthotropic, plasticity) are available to model the constitutive properties of the materials that are present in the structure. Once the model is assembled, a number of solution procedures are available for linear, nonlinear, and time-accurate problems. In addition, for very large nonlinear structural models, interfaces are available for external parallel sparse solvers that can greatly improve the calculation turnaround times. A number of advanced time-accurate integration algorithms are also included with FEAP, which are of interest in the computation of aeroelastic responses and constraints.

The general class that we created for structural analysis is `Structure`. As detailed in Figure 2.64, a `Structure` object contains a `Mesh` and a dictionary of `LoadCase` objects. As we mentioned before, Python supports dictionaries, which are lists of objects of any type, each of which is associated with a keyword. This enables us to handle multiple load conditions. For each load case, the displacement field, the stresses and the loads are stored in the `LoadCase` class. This is a capability that the underlying Fortran code does not have.

The functions provided by the `Structure` class are what one would expect from a structural analysis program: solve the displacements, compute the stresses, and set loads. There are also functions that are needed for aero-structural coupling, such as getting the surface of the finite-element mesh, and the transfer

weights.

Other data and functions are design oriented. The main class contains design variables and provides a function that computes gradients with respect to those design variables using an adjoint method. Each load case contains the corresponding adjoint vector and sensitivity of stresses with respect to the design variables.

Fluid Structure Interface

In aero-structural analysis, there is a clear interdependence between the equilibrium state of the two systems: the flow solution depends on deflections calculated by the structures, and the structural solution depends on the loads calculated by the flow solver. The procedures for transferring of displacements and loads, which are based on work by Brown [63], are discussed below.

Displacement Transfer

Each CFD surface node is associated with a point on the surface of the finite-element model in a pre-processing step, as shown in Figure 2.65. The association is performed by locating the point on the structural model surface that is closest to each surface CFD node.

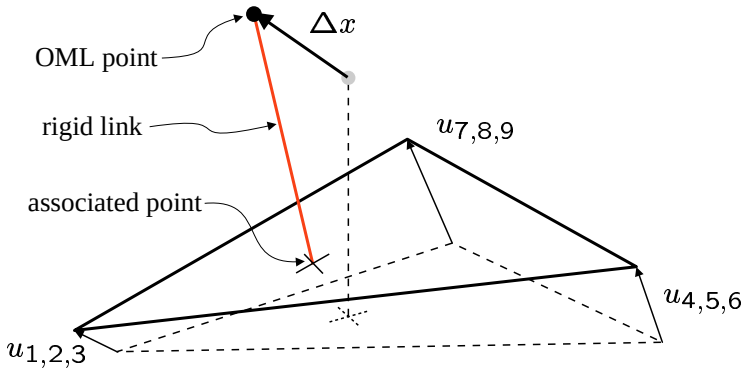


Figure 2.65: Displacement extrapolation procedure.

During aeroelastic analyses, the displacement of each associated point is computed by interpolating the node displacements of the element containing that point. The displacement is then transferred onto the surface CFD node using extrapolation functions that emulate a rigid link between it and the associated point on the surface of the structural model. For small angular deflections we use the linear relationship

$$\Delta \boldsymbol{x} = \boldsymbol{N}\boldsymbol{u}, \tag{2.45}$$

where $\boldsymbol{N}$ is a matrix of transfer weights and $\boldsymbol{u}$ is the vector of structural displacements.

Once a perturbation is applied to the surface of the CFD mesh, it must be propagated throughout the whole multiblock mesh. This volume mesh perturbation is achieved very efficiently by using WARP-MB.

Load Transfer

The load transfer procedure is responsible for transferring the pressures calculated by the CFD algorithm to the structural nodes. In order to transfer the aerodynamic loads from the CFD surface mesh to the structural

nodes, we first integrate the pressures in the CFD mesh to obtain a force vector for each CFD surface node. In transferring interpolated loads from the CFD surface into finite-element forces, it is crucial that both consistency and conservation be maintained. The property of consistency specifies that the resultant forces and moments due to the forces on the CFD surface mesh, must be equal to the sum of the nodal forces and moments applied to the structural model. There are an infinite number of structural load vectors that satisfy this requirement. However, we also require that the load transfer be conservative. Conservation stipulates that the virtual work performed by the load vector, $\boldsymbol{f}_S$, undergoing a virtual displacement of the structural model, $\delta \boldsymbol{u}_S$, must be equal to the work performed by the CFD forces, undergoing the equivalent displacement of the OML mesh, $\delta \boldsymbol{x}$. The virtual work in the finite-element model is given by the dot product

$$\delta W_{\mathrm{S}} = \boldsymbol{f}_S \delta \boldsymbol{u}, \tag{2.46}$$

while the virtual work performed by the fluid acting on the surface of the CFD surface mesh is given by

$$\delta W_{\mathrm{A}} = \boldsymbol{f}_A \delta \boldsymbol{x} \tag{2.47}$$

For a conservative scheme, $\delta W_{\mathrm{A}} = \delta W_{\mathrm{S}}$, and a consistent and conservative load vector then is given by

$$\boldsymbol{f}_A = \boldsymbol{N}^T \boldsymbol{f}_S \tag{2.48}$$

where we used the linear relationship (2.45) for the virtual displacements $\delta \boldsymbol{x}$. In Figure 2.66 we can see how the pressure field is integrated over an CFD surface patch to produce a force vector that is translated into the nodal forces of a finite element using equation (2.48). The result of the force transfer in a test case(supersonic business jet) is shown in Figure 2.67. The forces at the leading and trailing edges are particularly high due to the large non-structural surfaces that lie ahead and behind the wing. At a given spanwise location, these large forces also produce a moment in the direction parallel to the front and rear of the wing box, again due to the aerodynamic loads on the non-structural surfaces. The transfer of forces was found to be consistent and conservative.

As previously, the transfer matrix $\boldsymbol{N}$ is calculated in a pre-processing step. This matrix plays a dual role: it provides the appropriate weighting factors for both the transfer of CFD surface pressures to structural load vectors (2.48) and the transfer of the structural displacements to OML point displacement (2.45).

Both the displacement and load transfer procedures described in this section have been implemented into a Python module called *pycfd-csm*. The association between the CFD surface mesh and the CSM mesh is contained in this module.

Multi-block Mesh Perturbation

For both aero-structural analysis and design applications, the need arises to construct deformed meshes that conform to the actual shape of the OML at every point in a calculation. Given that our flow solver, TFLO2000, uses body-fitted multi-block meshes, and that these meshes have to be constructed by hand (with the aid of mesh generation software such as Pointwise's Gridgen), it becomes infeasible to regenerate these meshes from scratch every time that a surface perturbation is desired. In the past we have developed an algorithm called WARP-MB [74, 75], that is based on fast algebraic methods. WARP-MB is a multi-block wrapper on the WARPBLK program that carries out the deformation of a single block in a multi-block mesh.

WARPBLK requires as inputs the geometry of the initial block and the perturbations to the mesh points that lie on the external surfaces of the block. Some or all of these surface points can be provided to WARP-BLK. If only some of the deformed surface points are provided, WARPBLK uses quasi-1D and quasi-2D

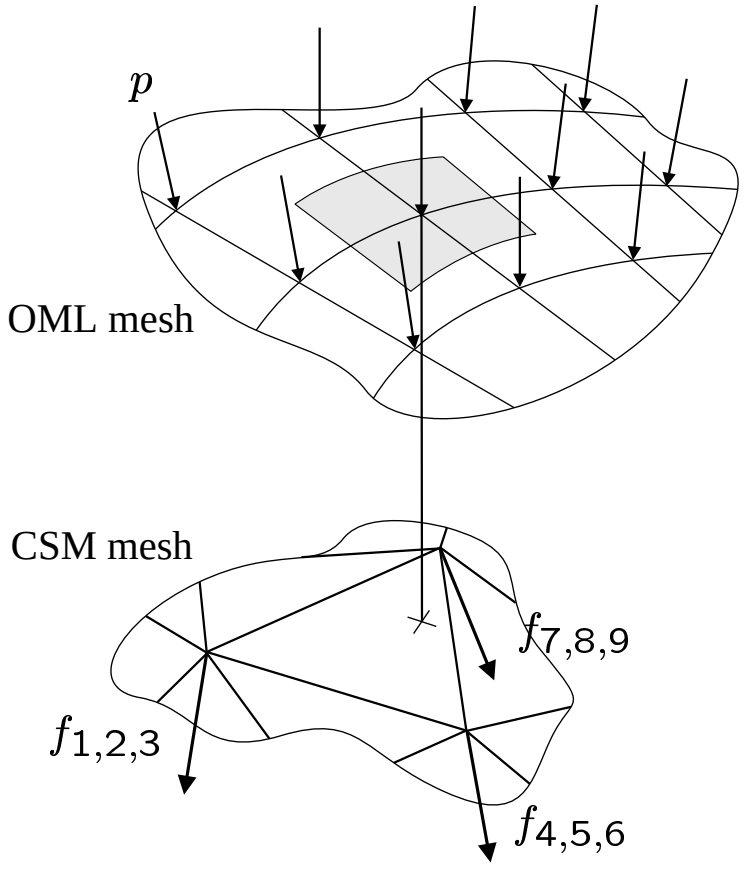


Figure 2.66: Transfer of the pressure on the surface CFD points to the finite-element nodal forces

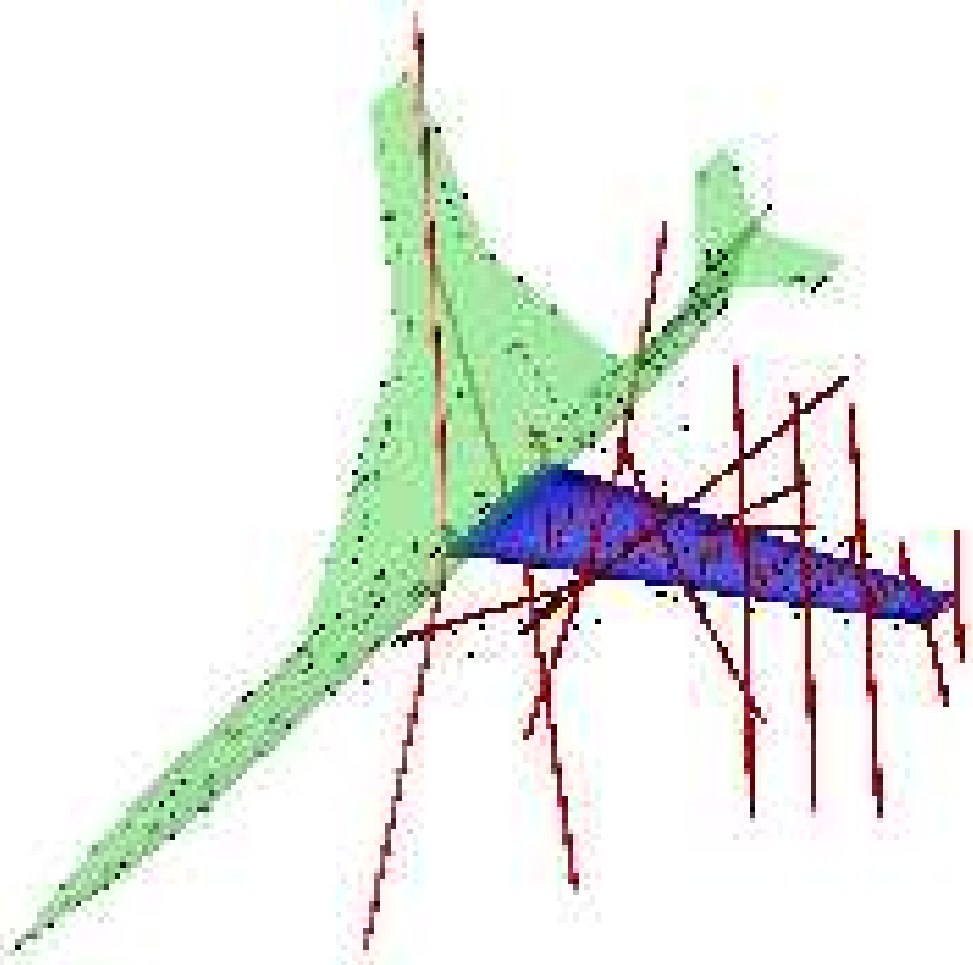


Figure 2.67: Aerodynamic loads transferred to the structural nodes.

versions of the warping algorithm to construct the deformed surface of the block. Then, the full three-dimensional version of WARPBLK perturbs the mesh in the interior of the block to achieve its final result.

WARPBLK is quite robust since it simplifies to a rigid body rotation/translation of the mesh in situations where rigid body motions are applied to the surface of the block. In addition, as the mesh deforms, the relative mesh spacings of the baseline mesh are preserved. This feature avoids problems with crossed edges in boundary layer meshes.

In a multi-block setting, the implementation of WARPBLK and WARP-MB has been named *pywarp*. It is a complete re-write of our old version and it only re-uses the WARPBLK procedure. There is a shortcoming of *pywarp*: we typically allow the surface of the object (a blade, for example) being modeled to move, while we fix the external boundaries of the mesh block that come in contact with the object. For this reason, if the displacements of the surface of the object exceed the physical dimensions of the block, WARP-MB is unable to produce a mesh without negative volumes. This has catastrophic consequences for an aero-structural simulation, since the flow solver will stop abruptly.

An alternative that has been advanced in the literature is the solution of a pseudo-structural problem for the mesh deformation problem [68]. In this approach, every cell in the mesh is replaced by a solid structural element. Displacements are applied, as required by the problem, to the appropriate nodes on the surface of this pseudo-structural mesh and then, through the solution of an elasticity problem, the motion of all of the nodes in the mesh can be computed. The stiffness of each of the elements in the mesh can be changed according to the distance from the moving surface in order to prevent the mesh from crossing. In principle, this is a generalization of the spring analogy that is often used for mesh deformation.

Although this is a very useful approach, the cost of mesh perturbation can become quite large as the number of cells in the mesh grows. This is particularly true of complete configuration viscous meshes with millions of cells, for which the mesh perturbation problem can become more expensive than the actual flow solution problem.

In order to avoid this high computational cost, we have re-written *pywarp* to introduce a hybrid, pseudo-structural model concept. Instead of modeling each cell in the mesh as a solid structural model, we have chosen to model each *block* of a multiblock mesh as a solid element. After solving a small elasticity problem (with a number of elements that is equal to the number of blocks in the mesh (say, from 200 to 1000), rather than to the number of cells in the mesh (up to 10^6) we apply the WARPBLK procedure to each of the blocks in turn. In this way we can accommodate very large displacements, but, at the same time, we are able to produce perturbed meshes at very low computational cost. *pywarp-solid*, the new Python module that we have created is therefore used in our aero-structural work.

Figures 2.69 and 2.70 below show the differences between the perturbed meshes produced by *pywarp* and *pywarp-solid* for an outboard section of the wing of a supersonic business jet. The addition of the pseudo-structural elasticity problem can generate much higher quality meshes, especially for large displacements.

Aeroelastic Iteration

The aerodynamic and structural solvers are coupled by exchanging information at regular intervals during the convergence process. One has to keep in mind that the time scales for individual solvers may be different and there might be over all reduction of accuracy if it is not done properly.

A diagram representing the aeroelastic iteration is shown in Figure 2.71. The first time the flow solver is called, the displacement field of the structure is initialized to zero. After N iterations of the flow solver, the

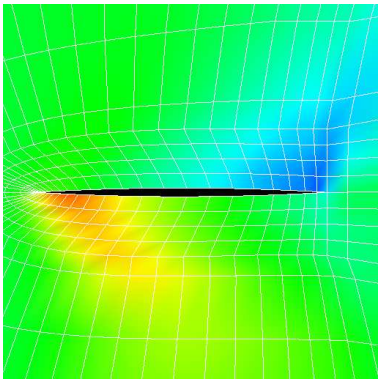


Figure 2.68: Undeflected outboard section of the supersonic jet.

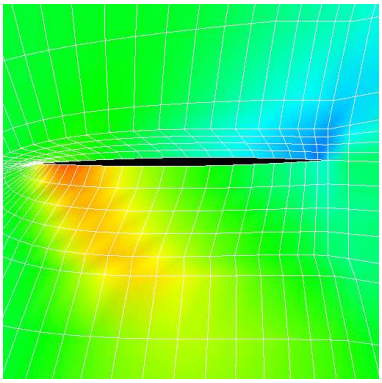


Figure 2.69: Mesh perturbed with *pywarp*.

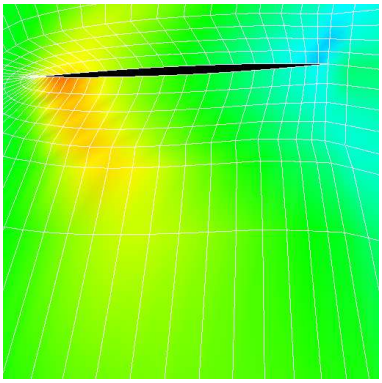


Figure 2.70: Mesh perturbed with *pywarp-solid*.

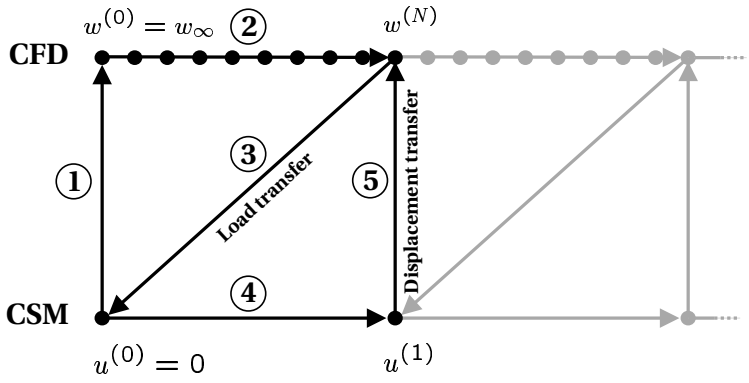


Figure 2.71: The aero-structural iteration procedure.

surface pressures are translated into nodal forces and the structural solver is called. The new displacement field is then translated to a movement of the CFD mesh and N more flow solver iterations are performed. This is one of the many ways of synchronization of the two solver may be achived [78]. displacement residuals.

Conclusions And Future Work

Most of the development for a viable High-fidelity Aeroelastic simulation has been completed and tested. Testing is underway for a full aeroelastic simulation and various test cases have been identified.

Bibliography

[1] A. Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. (91-1596), June 1998.

[2] M. McMullen, A. Jameson, and J.J. Alonso. Acceleration of convergence to a periodic steady state in turbomachinery flows. *AIAA paper 01-0152*, AIAA 39th Aerospace Sciences Meeting, Reno, NV, January 2001.

[3] M. McMullen, A. Jameson, and J.J. Alonso. Application of a non-linear frequency domain solver to the euler and navier-stokes equations. *AIAA paper 02-0120*, AIAA 40th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 2002.

[4] A.Quarteroni C.Canuto, M.Y.Hussaini and T.A. Zang. *Spectral Methods in Fluid Dynamics; Springer Series in Computational Physics*. Springer Verlag, 1988.

[5] S.S. Davis. Naca 64a010 (nasa ames model) oscillatory pitching. *AGARD Report 702*, January 1982.

[6] N. A. Pierce. *Preconditioned Multigrid Methods for Compressible Flow Calculations on Stretched Meshes*. PhD thesis, University of Oxford, Oxford, UK, 1997.

[7] S.R. Allmaras. Analysis of a local matrix preconditioner for the 2-D Navier-Stokes equations. *AIAA Paper* 93-3330, Orlando, FL, June 1993.

[8] E. Turkel. Preconditioned methods for solving the incompressible and low speed compressible equations. *Journal of Computational Physics*, 72:277–298, 1987.

[9] E. Turkel. Assessment of preconditioning methods for multidimensional aerodynamics. *Computers and Fluids*, 26(6):613–634, 1996.

[10] E. Turkel. Preconditioning techniques in computational fluid dynamics. *Annu. Rev. Fluid Mech.*, 31:385–416, 1999.

[11] E. Turkel. Preconditioning-squared methods for multidimensional aerodynamics. *AIAA Paper* 97-2025, 1997.

[12] B. van Leer, W.-T. Lee, and P. Roe. Characteristic time-stepping or local preconditioning of the Euler equations. *AIAA Paper* 91-1552.

[13] W.-T. Lee. *Local Preconditioning of the Euler Equations*. PhD thesis, University of Michigan, Ann Arbor, MI, 1992.

[14] D. Lee. *Local Preconditioning of the Euler and Navier-Stokes Equations*. PhD thesis, University of Michigan, Ann Arbor, MI, 1996.

[15] D.L. Darmofal and K. Siu. A robust multigrid algorithm for the Euler equations with local preconditioning and semi-coarsening. *Journal of Computational Physics*, 151:728–756, 1999.

[16] J. M. Weiss and W. A. Smith. Preconditioning applied to variable and constant density flows. *AIAA J.*, 33(11):2050–57, 1995.

[17] K. Hosseini and J. J. Alonso. Optimization of multistage coefficients for explicit multigrid flow solvers. *AIAA Paper* 3705, 2003.

[18] K. Hosseini and J. J. Alonso. Practical implementation and improvement of preconditioning methods for explicit multistage flow solvers. *AIAA Paper* 0763, 2004.

[19] ALVELIUS K. 1999 Studies of Turbulence and its Modeling through Large Eddy- and Direct Numerical Simulation. *Ph.D. thesis*, Dept. Mech. Eng., KTH, Stockholm, Sweden.

[20] 2003 Annual report, CITS.

[21] DURBIN, P. A. 1995 Separated Flow Computations with the k-ϵ-$\overline{v^2}$ Model. *AIAA J.*, **33**, 659-664.

[22] DURBIN, P. A. & PETTERSSON REIF B. A. 2001 *Statistical Theory and Modeling for Turbulent Flows.* John Wiley & Sons.

[23] GATSKI, T.B. & SPEZIALE, C.G. 1993 On explicit algebraic stress models for complex turbulent flows. *J. Fluid Mech.*, **254**, 59-78.

[24] GREENSPAN, H.P. 1968 *The Theory of Rotating Fluids.* Cambridge University Press.

[25] HAIRE, S., L. & REYNOLDS, W. C. 2003 Toward an Affordable Two-Equation Structure-Based Turbulence Model. *Technical Report No. TF-84*, Mech. Eng. Dept., Stanford Univ.

[26] KALITZIN, G. 2002 An efficient and robust algorithm for the solution of the v^2-f turbulence model with application to turbomachinery flow. *WEHSFF2002*.

[27] KALITZIN, G. & IACCARINO G. 2003 Toward Immersed Boundary Simulation of High Reynolds Number Flows. *CTR Annual Research Briefs*, 369-378.

[28] KASSINOS, S.C. & REYNOLDS, W.C. 1994 A Structure-Based Model for the Rapid Distortion of Homogeneous Turbulence. *Technical Report TF-61.* Mech. Eng. Dept., Stanford Univ.

[29] KASSINOS, S.C., REYNOLDS, W.C., & ROGERS, M.M. 1993 One-Point Turbulence Structure Tensors. *J. Fluid Mech.*, **249**, 337-71.

[30] LANGER, C. A. & REYNOLDS, W. C. 2003 A New Algebraic Structure-Based Turbulence Model for Rotating Wall-Bounded Flows. *Technical Report No. TF-85*, Mech. Eng. Dept., Stanford Univ.

[31] LIEN, F. S. & DURBIN, P. A. 1996 Non-linear k-ϵ-v^2 modeling with application to high-lift. *CTR summer proceedings*.

[32] MOSER R., KIM J. & MANSOUR N. 1999 Direct Numerical simulation of turbulent channel flow up to $Re_\tau = 590$ *Phys. Fluids*, **11**, 943-945.

[33] REYNOLDS W.C. 1994 Towards a Structure-Based Turbulence Model. In T. B. Gatski, S. Sarkar, and C. G. Speziale (Eds.), *Studies in Turbulence*. Lumley 60th birthday Symposium, 76-80, Springer-Verlag, New York.

[34] SHIH, T. H., LIOU, W. W., SHABBIR A, & ZHU J 1995 A new k-ϵ eddy-viscosity model for high Reynolds number turbulent flows. *Comp. Fluids* **24** 227-238.

[35] WALLIN S. & JOHNASSON A. V. 2002 Modeling streamline curvature effects in explicit algebraic Reynolds stress turbulence models. *Int. J. Heat and Fluid Flow*, **23 (5)**, 721-730.

[36] WILCOX D. C. 1993 Turbulence Modeling for CFD. *DCW Industries Inc.*, La Canada, California.

[37] M. Ji and P. A. Durbin, "On the equilibrium states predicted by second moment models in rotating, stably stratified homogeneous shear flow," Phys. Fluids **16**, 3540 (2004)

[38] SPALDING, D.B., 1961, A Single Formula for the Law of the Wall. *Trans. ASME., J. Appl. Mech.*, **28**, 444–458.

[39] M. V. Morkovin. The many faces of transition. In Viscous Drag Reduction. Plenum Press, 1969.

[40] X. Wu and P. A. Durbin. Existence of longitudinal vortices evolved from distorted wakes in a turbine passage. *Journal of Fluid Mechanics*, 446:199–228, 2001.

[41] S. Nagarajan. *Leading edge effects in bypass transition*. PhD thesis, *Stanford University*, June 2004.

[42] P. E. Roach and D. H. Brierley. The influence of a turbulent free-stream on zero pressure gradient transitional boundary layer development Part 1: Test cases T3A and T3B. In Numerical Simulation of Unsteady Flows and Transition to Turbulence, pages 319–347, 1992.

[43] R. G. Jacobs and P. A. Durbin. Simulations of bypass transition. *Journal of Fluid Mechanics*, 428:185–212, 2001.

[44] B. J. Abu-Ghannam and R. Shaw. Natural transition of boundary layers - the effects of turbulence, pressure gradient, and flow history. *Journal Mechanical Engineering Science*, 22(5):213–228, 1980.

[45] J. H. Leylek and R. D. Zerkle. Discrete-jet film cooling: A comparison of computational results with experiments. *Transactions of the ASME. The Journal of Turbomachinery*, 116:358–368, July 1994.

[46] D. K. Walters and J. H. Leylek. A systematic computational methodology applied to a three-dimensional film-cooling flowfield. *Transactions of the ASME. The Journal of Turbomachinery*, 119:777–785, October 1997.

[47] M. Tyagi and S. Acharya. Large eddy simulation of film cooling flow from an inclined cylindrical jet. *Transactions of the ASME. The Journal of Turbomachinery*, 125(4):734–742, October 2003.

[48] Z. Xiong and S. K. Lele. Numerical study of leading edge heat transfer under free-stream turbulence. AIAA Paper 2001–1016, January 2001. In 39th AIAA Aerospace Sciences Meeting and Exhibit.

[49] C. D. Pierce and P. Moin. Progress-variable approach for large eddy simulaiton of turbulent combustion. Flow Physics and Computaiton Division, Deparment of Mechanical Engineering, Stanford University. No. TF–80, June 2001.

[50] S. Gopalan., B.M.Abraham, and J.Katz. The structure of a jet in cross flow at low velocity ratios. *Physics of Fluids*, 16(6):2067–2087, 2004.

[51] M. D. Polanka, J. M. Cutbirth, and D. G. Bogard. Three component velocity field measurements in the stagnation region of a film cooled turbine vane. Technical Report 2001–GT–0402. Proceedings of ASME TURBOEXPO, June 4-7, 2001, New Orleans, Louisiana, USA.

[52] S. Ardey and L. Fottner. Flow field measurements on a large scale turbine cascade with leading edge film cooling by two rows of holes. Technical Report 1997–GT–524. ASME paper.

[53] M. Barberis and P. Molton. Shock wave - turbulent boundary layer interaction in a three dimensional flow. *AIAA paper 95-0227*, AIAA 33rd Aerospace Sciences Meeting, Reno, NV, January 1995.

[54] A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the euler equations by finite volume methods using runge-kutta time stepping schemes. *AIAA paper 81-1259*, 1981.

[55] M.A. Leschziner. Turbulence modelling for separate flows with anisotropy-resolving closures. *Phil Trans. Royal Society, Series A, 378, 1777*, pages 3247–3277, 2000.

[56] J. Dacles-Mariani, G.G. Zilliac, J.S. Chow, and P. Bradshaw. Numerical/experimental study of a wingtip vortex in the near field. *AIAA Journal*, 33(9):1561–1570, 1995.

[57] P.L. Roe. Approximate Riemann Solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43(2), 1981.

[58] J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995.

[59] Python. http://www.python.org.

[60] J. J. Alonso, P. LeGresley, E. Weide. pyMDO: A Framework for High-Fidelity Multi-Disciplinary Optimization. *AIAA Paper* 2004-4480, August 2004.

[61] J. J. Alonso, J. R. R. A. Martins, J. J. Reuther, and R. Haimes. High-Fidelity Aero-Structural Design Using a Parametric CAD-Based Model. *AIAA Paper* 2003-3429, June 2003.

[62] D. Beazley. SWIG (Simplified Wrapper and Interface Generator). http://www.swig.org.

[63] S. A. Brown. Displacement Extrapolation for CFD+CSM Aeroelastic Analysis. *AIAA Paper* 97-1090, Jan. 1997.

[64] P. Dubois. *Pyfort Reference Manual*. Program for Climate Model Diagnosis and Intercomparison, Lawrence Livermore National Laboratory, August 2000. http://sourceforge.net/projects/pyfortran.

[65] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[66] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. *User's Guide for SNOPT 5.3: A Fortran Package for Large-scale Nonlinear Programming*. Systems Optimization Laboratory, Stanford University, California, 94305-4023, Dec. 1998. Technical Report SOL 98-1.

[67] K. Jackson. pyGlobus. http://www-itg.lbl.gov/gtg/projects/pyGlobus.

[68] E. Lund, H. Mller, and L. A. Jakobsen. Shape Design Optimization of Stationary Fluid-Structure Interaction Problems with Large Displacements and Turbulence. *Structural and Multidisciplinary Optimization*, 25(5/6):383–392, 2003.

[69] J. R. R. A. Martins. *A Coupled-Adjoint Method for High-Fidelity Aero-Structural Optimization*. PhD thesis, Stanford University, Stanford, California, Oct. 2002.

[70] J. R. R. A. Martins, J. J. Alonso, and J. J. Reuther. High-Fidelity Aerostructural Design Optimization of a Supersonic Business Jet. *Journal of Aircraft*, 41(3):523–530, 2004.

[71] P. Miller. pyMPI. http://sourceforge.net/projects/pympi.

[72] P. Peterson. *f2py Fortran to Python Interface Generator*, December 2000. http://cens.ioc.ee/projects/f2py2e/.

[73] P. Peterson, J. R. R. A. Martins, and J. J. Alonso. Fortran to Python Interface Generator with an Application to Aerospace Engineering. In *Proceedings of the 9th International Python Conference, Long Beach, CA*, Mar. 2001.

[74] J. Reuther, J. J. Alonso, A. Jameson, M. Rimlinger, and D. Saunders. Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers: Part I. *Journal of Aircraft*, 36(1):51–60, 1999.

[75] J. Reuther, J. J. Alonso, A. Jameson, M. Rimlinger, and D. Saunders. Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers: Part II. *Journal of Aircraft*, 36(1):61–74, 1999.

[76] VIS, Inc. *OpenDX Paths to Visualization*, 2000.

[77] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method, 5th Edition, Vol. I*. Butterworth-Heinemann, 2000.

[78] C. Farhat, M. Lesoinne and N. Mamman. Mixed Explicit/Implicit Time Integration of Coupled Aeroelastic Problems: Three Field Formulation, Geometric Conservation and Distributed Solution. *International Journal for Numerical Methods in Engineering*, 21:807-835,1995

Chapter 3

Integrated Simulations

One of the major goals of our center is the demonstration of large-scale, multi-physics, integrated simulations and the development of the methods required for these. The example we have chosen for this demonstration is the simulation of the aero-thermal flow through an entire aircraft gas turbine engine. This problem involves many different physical processes, which are described using different models and even different computer codes. It has long been realized that to describe the combustion process in a gas-turbine engine, Reynolds averaged (RANS) methods are not accurate enough and large-eddy simulations (LES) have to be used. However, LES remains to be very costly compared with RANS, and needs particularly fine resolution, especially in boundary layers near solid walls. RANS simulations have also been found to describe the flow in turbomachinery applications quite accurately. The intention of the integration group is to develop a framework that allows using the RANS code TFLO for compressor and turbine simulations, and the LES code CDP to describe the reactive flow inside of the combustor (Fig. 3.1). Furthermore, beyond gas turbine applications, the multi-code integrated LES-RANS approach will be rendered suitable to a broader area of application.

3.1 Major Accomplishments

The work of the integration group focuses on the coupling of the different flow solvers developed in our center. The theroretical and practical challenges in the coupling of these very different flow solvers are addressed.

Following last year's proposed work plan and considering the recommendations of the review team, we accomplished the following tasks:

1. **Code Implementation:** LES outflow boundary conditions previously developed and validated in a structured LES solver were implemented into CDP and validation studies were performed. Furthermore, anti-aliasing filters developed last year were adapted for unstructured meshes and implemented into CDP. Local phase corrections were implemented into CDP and TFLO. The implementation of the interface routines into TFLO 2000 were iniated.

2. **Basic Research:** We spent effort in studying compressible boundary conditions. One of the studies addressed the compressible RANS boundary conditions in integrated simulations. Another study improved compressible outflow boundary conditions, especially for flows with strong temperature and density gradients.

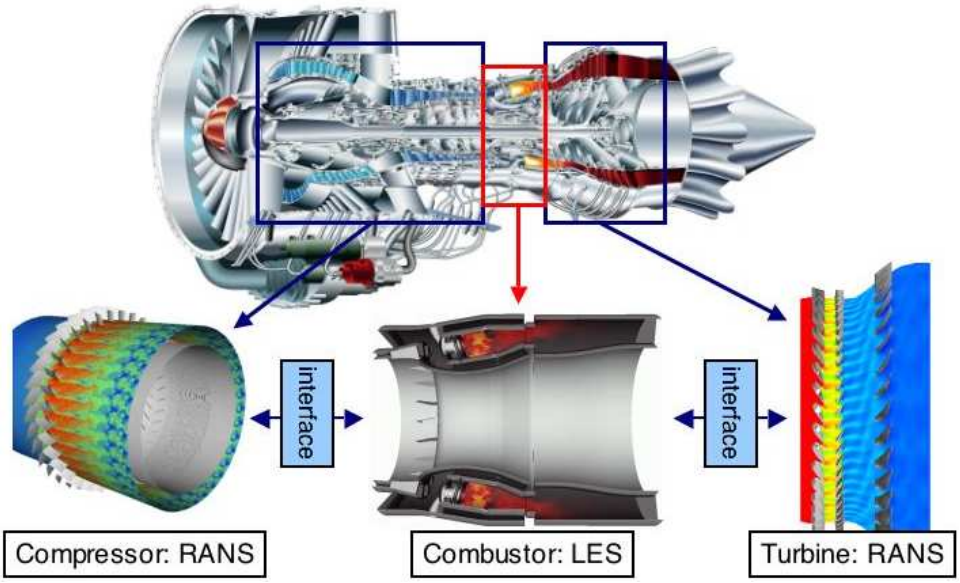


Figure 3.1: Decomposition of aircraft engine into components. Each component is computed by a separate flow solver.

3. **Application to Complex Geometries:** The computation of the NASA Stage 35/Diffuser geometry that we initiated last year was completed. We were able to demonstrate the value of integrated computations versus stand-alone simulations. We then applied for the first time the integrated RANS-LES approach to a realistic PW engine, coupling the last stage of the compressor with the entire combustor.

3.2 Motivation and Challenges of Integrated Simulations

The CDP/TFLO environment developed at the center is intended for use in multi-component simulations of aircraft engines. Simulations of this sort will be of major importance in the future design of low emissions, high performance, and high efficiency engines. The development of single components, such as combustor or compressor, is typically done in isolation from other components, which certainly prevents fine tuning of individual components and optimal design, since it requires the use of safety margins accounting for the unknown interaction of different components. Numerical simulations will first of all help to assess how important component interactions are and to find out, what are the parameters important for interactions. In addition, these simulations will ultimately be used in the actual design process in order to make safety margins smaller, which can lead to improved engines.

The main challenges in integrating these codes come first of all from the different turbulence models used in these codes. TFLO only computes ensemble averaged fields, whereas CDP resolves the large scale turbulent motions in space and time. In addition, because of the high Mach numbers encountered in the turbomachinery components, TFLO is a fully compressible code, whereas the present version of CDP is based on a variable density, low Mach number formulation, which does not resolve acoustic pressure variations, but makes the code substantially faster for the flow situations of a typical combustor. However, since this method cannot be used for predictions of combustion instabilities, our future CDP code will have the option

of a fully compressible solver, which will require the specification of different boundary conditions at the code interfaces.

One of the very important, and at the same time more difficult, challenges of our efforts is the verification and validation of the methods we are developing. Typically, there are no detailed experimental data available for integrated systems on a scale smaller than the full engine. However, the codes used in the integrated simulations have been extensively validated with simple and complex validation experiments. The development of appropriate boundary conditions and code coupling schemes can also be validated with some confidence. Typically, this validation is done for a domain, which can be computed entirely by an LES code. The solution of this full LES is then regarded as the correct solution. The same flow domain is then being computed in a coupled simulation, partly by LES, partly by RANS. If the coupling scheme and the boundary conditions are well formulated, we can expect to get the same results as with the full LES. This approach only suffers from the fact that the RANS approach, which is part of the integrated simulation, could be expected to yield different results anyways, because of the different nature of the applied turbulence model.

3.3 Code Implementation

3.3.1 LES Outflow Boundary Conditions for CDP

In previous years, we have developed LES outflow boundary conditions for integrated LES-RANS computations and tested these using a structured LES flow solver [30, 33]. The challenge is for the LES flow solver to take the upstream influence of the downstream flow development computed by RANS into account. We consider overlapping grids between the upstream LES and the downstream RANS. In the LES portion of the overlap region we use virtual body forces to drive the LES solution to a RANS solution delivered by the downstream flow solver.

$$F_i(\mathbf{x}) = \sigma \left(\overline{u}_{i,\text{RANS}}(\mathbf{x}) - \overline{u}_{i,\text{LES}}(\mathbf{x}) \right) . \tag{3.1}$$

In Eq.(3.1) $\overline{u}_{i,\text{RANS}}$ is the vector of target velocities obtained from the RANS computation and $\overline{u}_{i,\text{LES}}$ is the vector of time-averaged velocities from the LES computation.

The forcing term is acting on mean velocities, while the corresponding LES momentum equation is solved for the instantaneous velocities. Thus, temporal mean velocities are determined from the instantaneous LES flow solution in order to define the virtual body force. This allows to correct the LES mean velocity field without attenuating the resolved turbulent fluctuations. Note that the RANS velocities $\overline{u}_{i,\text{RANS}}$ are not prescribed in only one plane, but in the entire overlapped zone.

The choice of σ controls the characteristic response time of the LES solution to a change in the outlet boundary condition. If σ tends to zero, the body force becomes essentially ineffective resulting in a drift of the outflow mean velocity profile towards the unforced solution. High values of σ lead to faster change to the desired velocity field, but may lead to numerical instabilities.

Previously, we determined σ from the bulk velocity and the length of the forcing region. We now want to provide a more accurate estimate for an adequate choice of σ. This can be given by a 1D analysis of the stationary Euler equations:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -\frac{\partial p}{\partial x} + \sigma(\langle\overline{u}\rangle_{\text{RANS}} - \langle\overline{u}\rangle_{\text{LES}}) . \tag{3.2}$$

To simplify the equation, we assume a zero pressure gradient and a constant convection velocity with the bulk velocity u_B. Furthermore, the flow is stationary, which makes $\langle\overline{u}\rangle_{\text{LES}} = u$. With $\langle\overline{u}\rangle_{\text{RANS}} = u_t$, the target velocity Eq. 3.2 becomes:

$$u_B \frac{\partial u}{\partial x} = \sigma(u_t - u) \ . \tag{3.3}$$

This ordinary differential equation can be solved analytically and leads to the following expression for u:

$$u(x) = u_t + (u_0 - u_t) \exp\left(-\frac{\sigma}{u_B} x\right) \tag{3.4}$$

with u_0 being the velocity at the beginning of the forcing region.

We now want to determine σ so that at the end of the forcing region at $x = l_F$ the velocity difference is smaller than the relative error

$$\epsilon = \frac{|u(l_F) - u_t|}{u_t} \ . \tag{3.5}$$

Then, Eq. 3.4 leads to

$$\sigma_{\text{min}} = \frac{u_B}{l_F} \ln\left(\frac{|u_0 - u_t|}{\epsilon u_t}\right) \ . \tag{3.6}$$

Although this estimate for σ ensures the accuracy of the approach for steady flows, in truly unsteady coupled computations a higher value for σ should be used in order to decrease the time-lag in which the flow solution adjusts to the target velocity obtained from an unsteady downstream computation.

On the upper end, σ is limited by numerical stability. Here, it is useful to write σ as an inverse time-scale τ_F. The upper limit is then defined corresponding to the CFL-condition:

$$\sigma_{\text{max}} = \left(\frac{1}{\tau_{\text{F}}}\right)_{\text{max}} = \frac{u_{\text{c}}}{\Delta x_{\text{F}}} \tag{3.7}$$

with Δx_{F} being the size of the smallest cell in the forcing region and u_{c} is the local convection velocity in this cell.

This body force outflow scheme has been implemented in the unstructured LES solver CDP. Even though the underlying organization of the unstructured CDP is more complex than a structured LES code, the local nature of the definition of the boundary treatment requires little additional work in for the porting process to an unstructured solver. Care has been taken in the implementation to keep the body force implementation separate from the core section of CDP. The result is that the main source tree section of CDP is left untouched. All other integration subroutines were added through the user-hooks section of CDP, where user-defined customary routines are expected by code design. For example, invocation of the body force computation is made in a temporal-hook in the user section instead of being made directly inside the LES momentum equation solver. The implementation in CDP is also assisted by several utility functions in the original unstructured code such as interpolation and searching.

Swirl is an important feature in combustor fluid dynamics, since it determines the mixing behavior as well as the evolving pressure field in the combustor. The validation of the body force outflow scheme in CDP for swirling flows has been performed in a well-controlled setting for which high quality experimental data are available. The flow geometry is a pipe with a sudden axisymmetric expansion to twice the diameter. The Reynolds number based on the diameter and the average velocity in the upstream pipe is 30,000. The inflow turbulence in the simulation was generated by using a swirling pipe flow database. Inlet swirl number based on the ratio of axial fluxes of angular to axial momentum is 0.6. Prior to testing of the outflow body

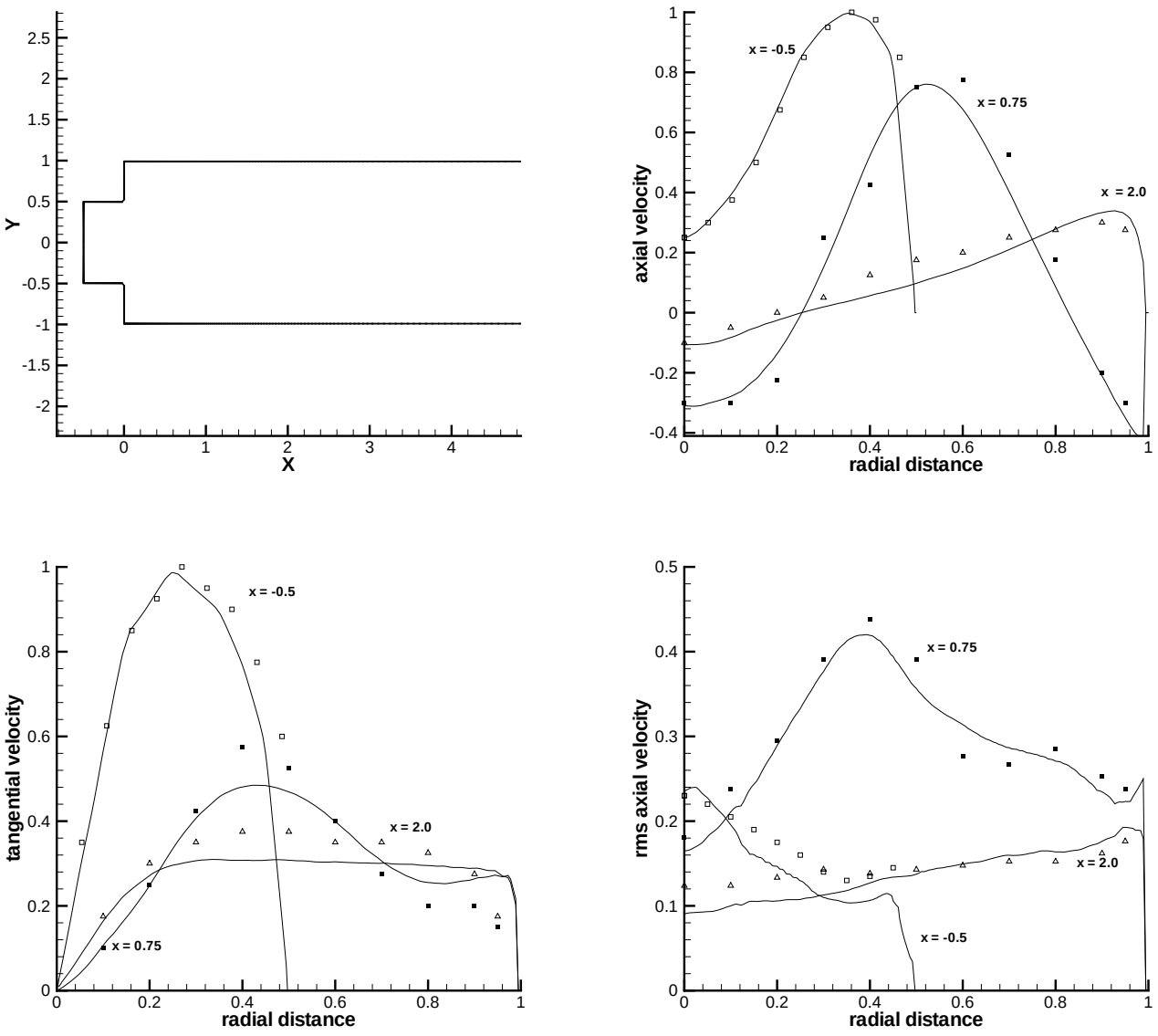


Figure 3.2: Geometry and velocities of a swirling flow through an sudden 1:2 axisymmetric expansion. Symbols are experiments and lines are from CDP.

force schemes, an LES computation of the flow using the full geometry with convective outflow boundary condition was performed. In the full LES, the flow field is computed from $x = -0.5$ to $x = 10$. Figure 3.2 shows the configuration and results from the full LES. In the figure, CDP results obtained from a 1.2×10^6 element mesh are compared with the LDA experimental data of Dellenback, Metzger & Neitzel (1988) [6]. Previously, we computed the same flow with the structured cylindrical LES code [32]. In the present LES, we use a similar number of grid points as for the LES using the structured flow solver.

To test the outflow body force scheme, the full LES geometry is split into two computational domains with a short overlap region. The streamwise coordinate range of the upstream region is $-0.5 \leq x \leq 1.5$ with a overlap zone from $1.1 \leq x \leq 1.5$, in which the body force scheme can be applied. In order to evaluate the effect of the scheme, a LES computation in the truncated domain $-0.5 \leq x \leq 1.5$ using simple convective outflow boundary condition was also performed without invoking the body force scheme. Applying the correction of the LES outflow by virtual body forces according to Eq.(3.1) leads to the question of how to define the mean value $\overline{u}_{i,\mathrm{LES}}$. Two approaches were tested in this study: using the overall time-averaged mean value or averaging over a trailing time window [33]. The first approach ensures the least interference with turbulent fluctuations but does not allow for unsteadiness in the mean profile. The proper choice of a trailing time window in the second approach makes it possible to preserve the unsteadiness of the mean profile.

Figure 3.3 shows the profiles of mean axial velocity at five streamwise stations as a function of radial distance near the expansion. Figure 3.4 shows the profiles for the tangential velocity component. A total of four sets of results are presented in the two figures:

(1) Full LES with convective outflow boundary condition,

(2) Truncated LES with convective outflow boundary condition at $x = 1.5$,

(3) Truncated LES with body force trailing window scheme applied within $1.1 \leq x \leq 1.5$,

(4) Truncated LES with body force overall mean scheme applied within $1.1 \leq x \leq 1.5$.

The results demonstrate that without proper treatment of the boundary conditions (case (2)) the results of the full LES can not be recovered. Incorporating the body force scheme according to Eq.(3.1) into the truncated LES improves the prediction. There remain only relatively small differences between cases (3) and (4) for the present flow.

3.3.2 Anti-Aliasing Filters and Local Phase Correction

Last year we demonstrated the presence of aliasing errors in multi-code/multi-physics simulations. This effect may occur, when one of the simulation codes runs at a smaller time-step than the other. High frequencies, which are resolved in the simulation code running on a small time-step, may reflect into the long-wave spectrum in the other simulation code. We proposed digital anti-aliasing filters and developed a spatial form of the filter in order to correct phase errors invoked by the filters [29, 28].

A finite response temporal filter can be written in the form,

$$r(t_k) = \sum_{n=0}^{N} b_n s(t_{k-n}) \,, \tag{3.8}$$

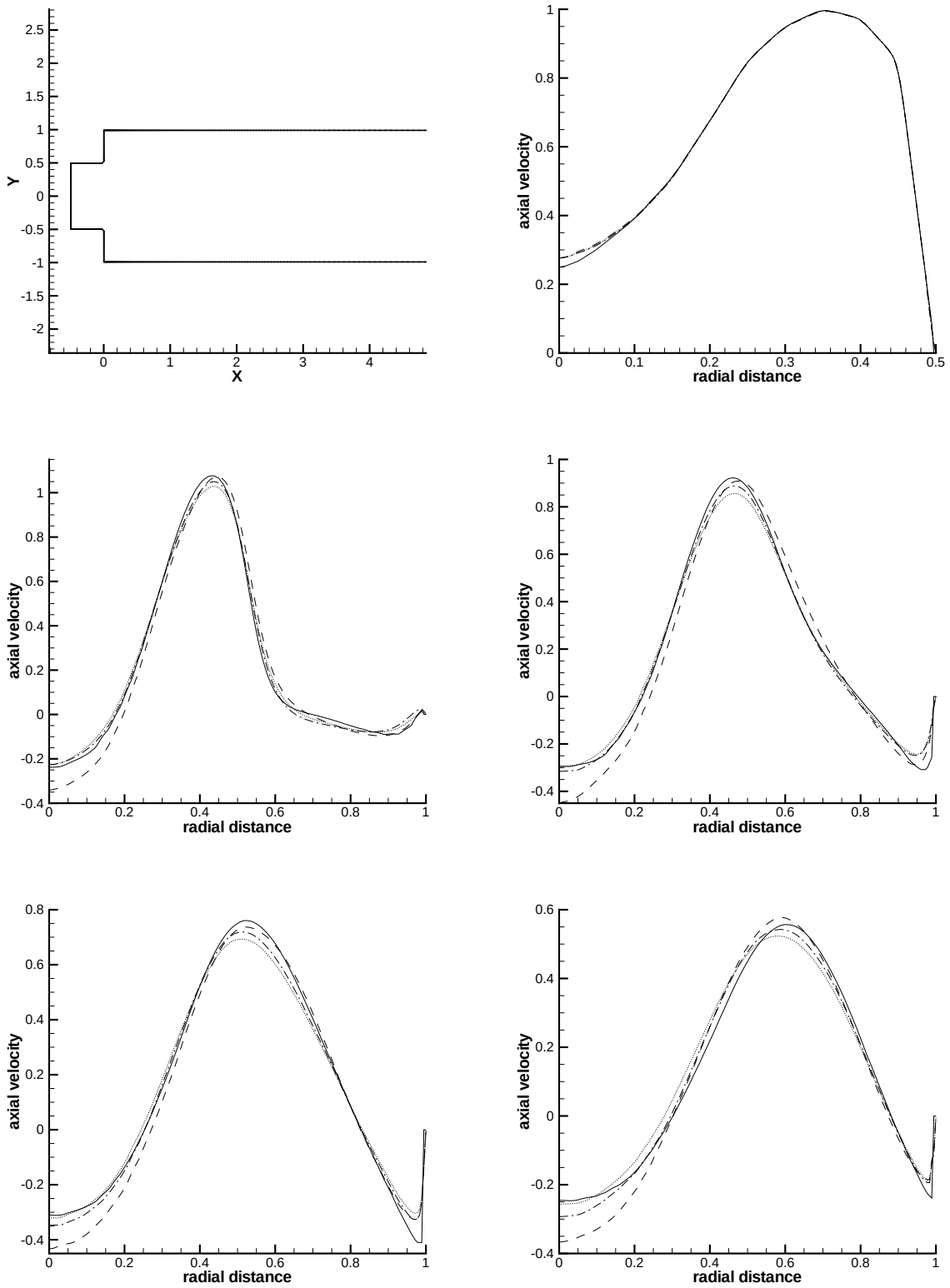


Figure 3.3: Geometry and axial velocity of a swirling flow through a sudden 1:2 axisymmetric expansion. Solid lines: full geometry; dashed lines: truncated geometry at x=1.5; dotted lines: body force scheme with trailing time window; dash-dotted lines: body force scheme without trailing time window. Results are for x=-0.49, 0.25, 0.5,0.75 and 1.0, respectively (row major).

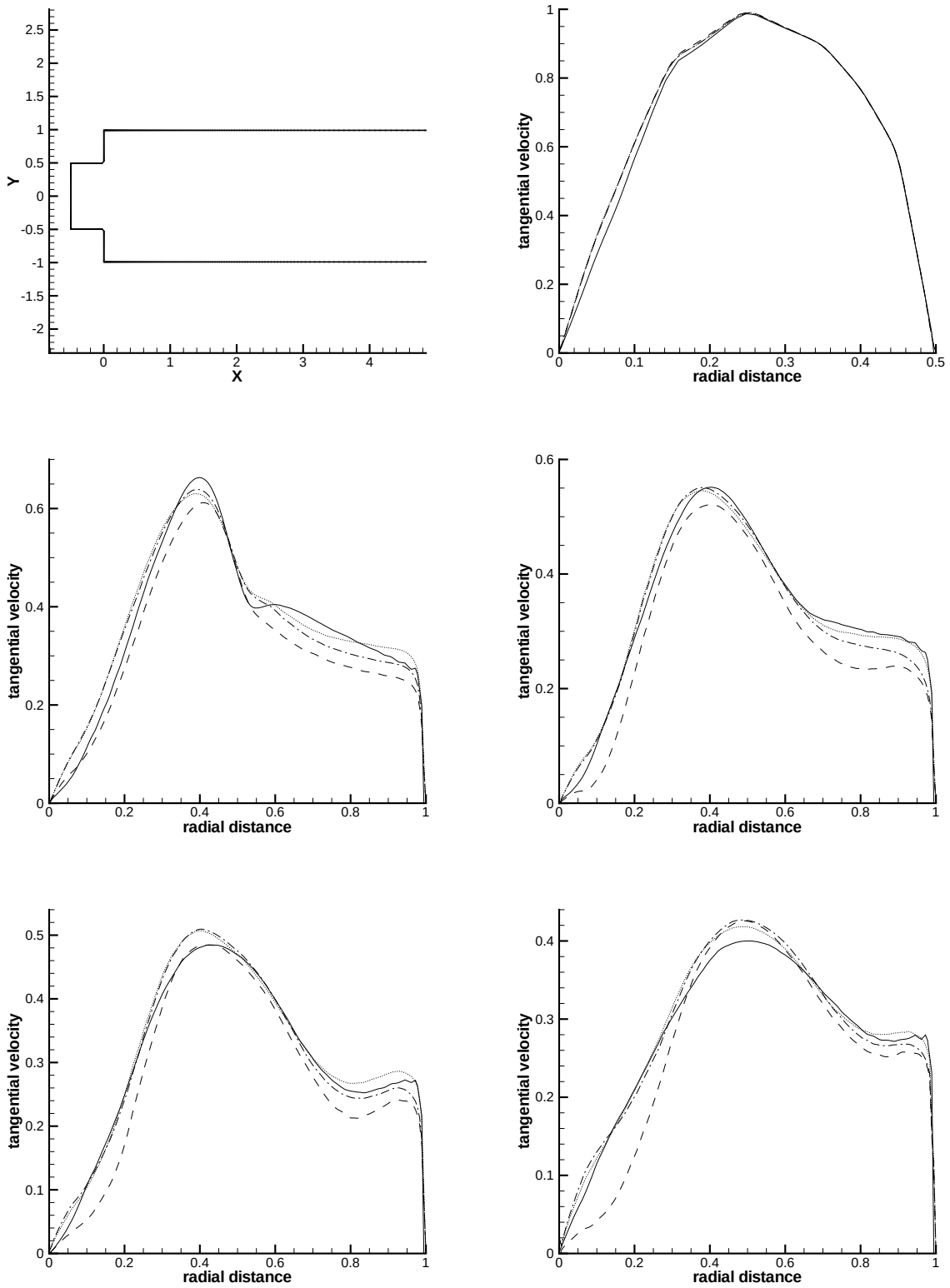


Figure 3.4: Geometry and tangential velocity of swirling flow through an abrupt 1:2 axisymmetric expansion. Solid lines: full geometry; dashed lines: truncated geometry at x=1.5; dotted lines: body force scheme with trailing window; dash-dotted lines: body force scheme without trailing window. Results are for x=-0.49, 0.25, 0.5,0.75 and 1.0, respectively (row major).

where r is the filter response, s the original signal, b_n is filter coefficients and t_k is the time at step k. Using Taylor's hypothesis the temporal filter may be converted to a spatial filter suitable for the LES,

$$r(t_k) = \sum_{n=0}^{N} b_n s(x_{k-n}) \,, \qquad \text{with} \qquad x_n - x_{n-1} = u_{\text{bulk}} \Delta t_{\text{LES}} \,. \tag{3.9}$$

While the temporal form of the filter (Eq. (3.8)) creates an artificial phase lag determined by the order of the filter creating a constant time delay of

$$\Delta t = \frac{N-1}{2 \cdot f_{\text{sample}}}, \tag{3.10}$$

The spatial form was designed not to have such a delay. The phase delay of this filter is a sum of the filter time-delay and the time correction by the shift of the origin Eq. (3.9)

$$\Delta t_{\text{total}} = \Delta t_{\text{filter}} + \Delta t_{\text{origin shift}} \tag{3.11}$$

The filter time-delay Δt_{filter} is defined corresponding to Eq. (3.10). The time correction due to the shift of the origin upstream is given by

$$\Delta t_{\text{origin shift}} = \frac{\Delta x}{u_c} = -\frac{N-1}{2} \frac{u_c}{f_{\text{sample}}} \frac{1}{u_c} \tag{3.12}$$

Equation (3.11) then becomes

$$\Delta t_{\text{total}} = \frac{N-1}{2 \cdot f_{\text{sample}}} - \frac{N-1}{2} \frac{1}{f_{\text{sample}}} = \mathbf{0} \,. \tag{3.13}$$

The zero time delay ensures a true phase transfer of a given perturbation.

Anti-aliasing on turbulent flows

While our previous study focused on laminar flows in order to demonstrate the basic concept of aliasing and anti-aliasing filters, we now want to demonstrate this effect and the effect of the anti-aliasing filters in turbulent flows.

For this investigation, a pipe flow at a Reynolds-number of $Re = 15,000$ is computed by using two fully coupled LES flow solvers, one computing the upstream and the other computing the downstream part of the pipe. Both flow solvers exchange flow information at a given time step $\Delta\tau >> \Delta t_{\text{LES}}$. Figure 3.5 shows the considered test case: a pipe is split into an upstream domain computed by one LES flow solver and a downstream domain iscomputed by the second. Both pipe segments are 3 diameters D long with an overlap of $1D$. The interface is used to communicate the flow variables between both flow solvers [34, 35].

Since the inflow of the upstream pipe has to be turbulent, the inflow was generated by a separate LES computation of a periodic pipe flow with $Re = 15,000$ and $S = 0.15$. The outflow plane of this computation is recorded into a data base, which is used as inflow for the actual LES computation [21].

As in the laminar case, the swirl velocity is periodically excited. In the turbulent case, this translates into a modulation of the mean velocity field, while the turbulent fluctuations are added to the main flow field. Since the propagation of a convective wave is disturbed by the turbulence, the upstream pipe is shortened by

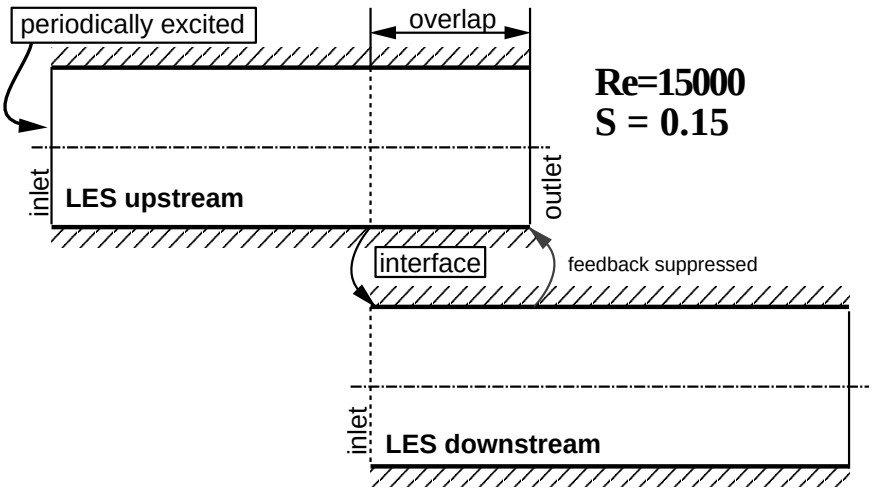


Figure 3.5: Geometry of the test-case.

$1.5D$ so that the inlet plane is $0.5D$ upstream of the interface plane. This allows to obtain a clearer signal from the forcing at the interface plane.

The solution of the upstream flow solver is filtered at the appropriate plane using a given RANS time step $\Delta\tau$. Inflow boundary conditions for the downstream LES flow solver are then generated using the data base method described in [32]. A data base of turbulent flow data generated by a turbulent pipe flow simulation is used and scaled to satisfy the mean velocity and velocity fluctuations provided by the upstream flow solver. The data base is generated by an LES computation of a periodic pipe at a higher Reynolds-number of $Re = 30,000$ in order to point out differences in the high frequency spectrum in the upstream and downstream domain.

Figure 3.6 shows the energy spectra of these simulations. In the upstream domain, the lower forcing frequency can be easily identified. The high frequency forcing is disturbed by the turbulence and is less distinct. However, in the unfiltered downstream spectrum aliasing of the second frequency can be observed easily.

Applying a temporal filter to the upstream LES computation suppresses the aliasing successfully. Some differences in the high frequency spectrum between upstream and downstream domain can be observed, which are due to the reconstruction of high frequency turbulence using a data base in the downstream domain. The use of a spatial filter yields essentially the same result as the temporal filter.

Anti-aliasing filters: Implementation into CDP

The anti-aliasing scheme Eq.(3.9) has been implemented in the unstructured LES solver CDP. The underlying organization of the unstructured CDP is more complicated than a structured LES code and hence, a fair amount of message passing has to be performed. One of the main issues is that the extend of the spatial filter defined by Eq. (3.9) can be larger than the computed domain of a single processor. This means that in order to apply the filter, flow information from neighboring processors may be needed, since some of the filter points are not in the local domain. We will call these points henceforth non-local filter points.

An initial search routine determines the location of the filter points and organizes the communication of

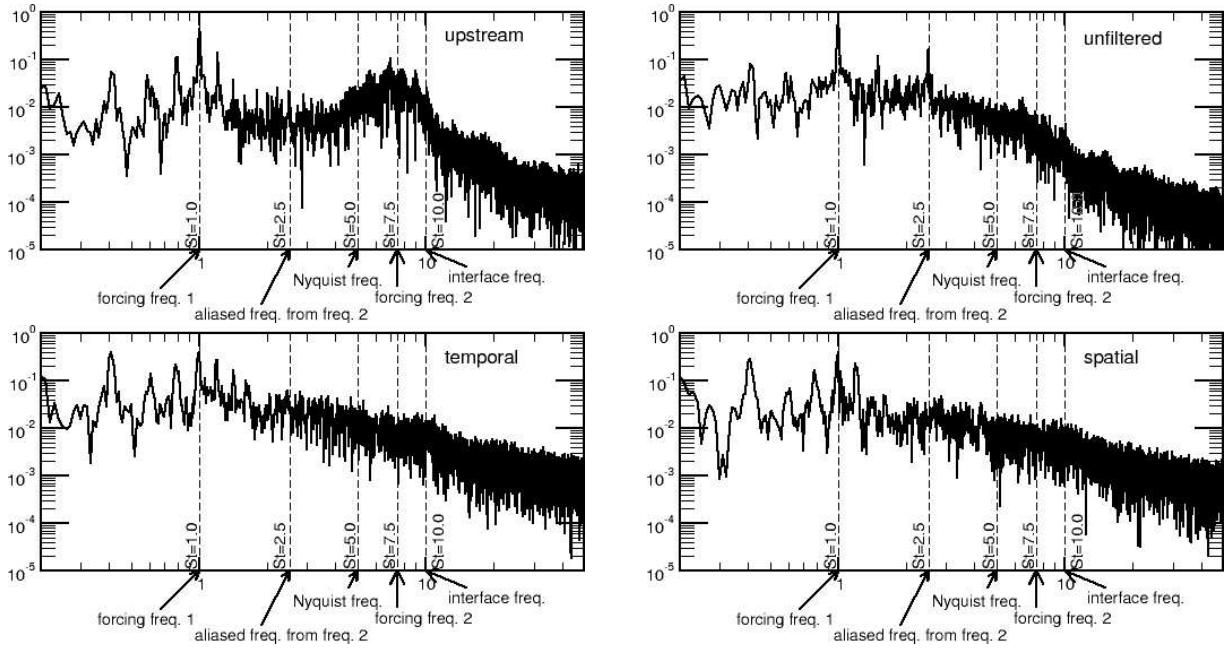


Figure 3.6: Energy spectrum at the interface. Left top: original signal in the upstream domain. Right top: downstream solution without filtering. Left bottom: downstream solution using temporal filter (Kaiser window). Right bottom: downstream solution using spatial filter.

the non-local filter points:

1. All processes serving interface points to the peer code define filter points needed for filtering for each requested interface point x_{k-n} on the current processor.

2. All processes determine whether filter points x_{k-n} can be found on this local processor.

3. All processes record the coordinates, rank, control volume index and other information for all filter points not found on local processor (non-local filter points) and communicate these to the root process.

4. The root process will compile a global list containing all non-local filter points of all processors and sends this list to all processors.

5. All processes determine if the current processor can serve any non-local filter points of other processors through three-dimensional searching of the local domain and communicate the results to the root process.

6. The root process checks whether all non-local points can be served by other processes and sends out the entire list of non-local points, including the information which processor can serve which non-local filter point.

During the actual flow computation, prior to each communication with the peer code, the following procedure is applied:

1. All processors serving non-local filter points from other processors send flow information for these points to the root process.

2. The root process sends out the global list containing flow information for all non-local points to all processors.

3. All processes serving interface points collect the flow information for each filter point locally and from the global list of non-local points.

4. Finally the filter according to Eq. (3.9) can be applied.

Implementation of the anti-aliasing code is confined within one interface subroutine and the core section of CDP has been left untouched.

Local phase correction: Implementation

In the previous year, we have developed a local phase correction in order to minimize the phase delay that is induced by the explicit coupling of the two simulation codes.

$$\begin{aligned} u_{i,\text{LES}}(t) = \quad & \overline{u}_{i,\text{RANS}}(\tau) + \\ & + (\overline{u}_{i,\text{PREDICT}}(\tau+\Delta\tau) - \overline{u}_{i,\text{RANS}}(\tau)) \cdot \frac{t-\tau}{\Delta\tau} + \\ & + \underbrace{u'_{i,\text{DB}}(t)}_{IV} + \\ & + \underbrace{(\overline{u}_{i,\text{PREDICT}}(\tau) - (\overline{u}_{i,\text{RANS}}(\tau)) \cdot \left(1 - \frac{t-\tau}{\Delta\tau}\right)}_{V}, \end{aligned} \tag{3.14}$$

with $\overline{u}_{i,\text{PREDICT}}$ a predicted value for $\overline{u}_{i,\text{RANS}}$ at $\tau + \Delta\tau$. Since the predicted value at $\tau + \Delta\tau$ is going to differ from the actual computed value by the RANS code, a corrector term is needed to fulfill a continuous transition of the inflow boundary condition (Term V).

This phase correction has now been implemented into the CDP code using a linear extrapolation for the prediction of $\overline{u}_{i,\text{PREDICT}}$. The linear form was chosen according to the findings of last year's investigation of this phase correction in a structured LES code.

The local phase correction has also been implemented into the RANS flow solver TFLO. On the TFLO side, the predicted value can be imposed directly ($u_{i,\text{RANS}}(\tau+\Delta\tau) = \overline{u}_{i,\text{PREDICT}}(\tau+\Delta\tau)$) since no intermediate values are needed.

3.3.3 TFLO 2000 Interface

During the FY04, the development of TFLO 2000 has reached a level of maturity that allows to be considered for replacing the original version of TFLO for all integrated simulations. A milestone that had to be reached by TFLO 2000 was the computation of full 3D turbomachinery applications. This milestone has been reached for a number of test cases, including a new implementation of the time-spectral flow solution approach for periodic flows. We have initiated the implementation of the interface routines into the new

code in order to replace the old TFLO version. The implementation is scheduled to be completed at the review meeting.

3.4 Basic Research

3.4.1 RANS Boundary Conditions for Integrated Simulations

In the following section, we want to have a closer look at the definition of RANS boundary conditions in integrated simulations [13]. We have chosen a turbulent pipe flow, where an upstream section of the pipe is computed with one flow solver, and a downstream section with another. For the course of this study we will use two instances of the the same flow solver TFLO. This allows us to compute the entire domain with a single flow solver, which will deliver a reference solution we can compare the integrated simulations with.

Interface boundary conditions

Assuming that all the necessary flow information is exchangeable, the communication between two separate flow solvers can be very similar to the information exchange between processors of a parallel computation in the multi-block approach [25] or/and the Overlapped (Chimera) grid approach [5, 42]. Meanwhile, in the coupling of LES and RANS flow solvers, whose mathematical formulations are different, the formulation of LES and RANS interface boundary conditions becomes a new and very challenging issue to be resolved. The following problems, then, can be addressed: first, unsteady LES boundary conditions have to be generated which produce the statistical properties of the time-averaged solution delivered by the RANS flow solver. Even if an unsteady RANS computation is assumed, the time-step of the unsteady RANS computation can be larger than the LES time-step by several orders of magnitude. The LES boundary condition then has to correspond to the ensemble-averages delivered by the RANS computation. LES inflow conditions from a RANS interface and LES outflow conditions from a RANS zone have also recently been investigated and presented [30, 31, 32, 33]. Second, all necessary flow data may not always be available for the coupling of such different codes. In the current study, for example, the compressible formulation of the RANS flow solver and the low-Mach-number formulation of the LES code posed one of the challenges since the density fields of two codes are not compatible. While the RANS code allows for acoustic waves to propagate within the limits of its domain, the density field of the LES solution is uniform since it is entirely defined by mixing and the combustion process, not by acoustics.

To compensate any shortages of the necessary flow information at the RANS interface due to this issue, we suggest that the typical inlet/exit RANS boundary analysis is to be employed. We will, for convenience, call the RANS inlet/exit boundary conditions the "interface inflow/outflow" boundary conditions when they are used at the interface. Research in this paper focuses on the RANS interface inflow/outflow boundary conditions for simple pipe flows. This paper presents a numerical experiment that justifies the treatment of the interface communication as a boundary condition problem. Although the present work should eventually provide the interface boundary conditions for general cases of RANS-LES coupling, a RANS-RANS coupling with the same interface communication routines was used for representing the RANS-LES coupling, in order to explicitly exclude sources of error that may stem from the different mathematical formulations in RANS and LES. A further simplification was made by considering only steady state problems.

Table 3.1: Number of Physical Boundary Conditions

Boundary Type	Euler	Navier-Stokes
Supersonic inflow	5	5
Subsonic inflow	4	5
Supersonic outflow	0	4
Subsonic outflow	1	4

RANS Boundary Conditions

Except from the fact that the unsteady LES flow data to be used as RANS interface boundary conditions are time-averaged over the time-step applied by the RANS flow solver, the specification of RANS interface boundary conditions from the LES data is essentially the same as the specification of the inlet and exit boundary conditions of a single RANS simulation. In this section, the treatment of typical inflow and outflow (inlet and exit) boundary conditions will be briefly described in order to help readers understand the discussion on the RANS interface boundary conditions in the following result section.

Although the concept of "characteristic lines" may be questionable for the Navier-Stokes equations, we still deal with the hyperbolic propagation-dominated systems and specify boundary conditions for hyperbolic systems using the relations based on the characteristic lines, i.e., on the analysis of the different waves crossing the boundary. This method has been extensively studied for the Euler equations, and the mathematical background of boundary conditions based on characteristic wave analysis can be found elsewhere [16, 10].

First of all, we must distinguish two classes of boundary conditions: "physical" boundary conditions and "numerical" boundary conditions. The physical boundary condition specifies the known physical behavior of the dependent variables at the boundaries. In order to solve a problem numerically, the total number of boundary condition should be equal to the number of primitive variables. When the number of physical boundary conditions (say N_p) is less than the number needed for the numerical methods (say N_t), $N_n = N_t - N_p$ numerical boundary conditions have to be introduced. The following two essential issues have to be addressed:

(1) How many physical boundary conditions are to be imposed at a given boundary?

The number of physical boundary conditions to be imposed will be equal to the number of waves propagating with each characteristic velocity, λ_i, into the interior domain. These characteristic velocities are

$$\begin{aligned} \lambda_1 &= u_n - c, \\ \lambda_2 &= \lambda_3 = \lambda_4 = u_n, \\ \lambda_5 &= u_n + c, \end{aligned} \tag{3.15}$$

where u_n is normal flow velocity and c is speed of sound.

The number of necessary and sufficient physical boundary conditions are summarized in Table 1. A detailed theoretical analysis is found in the reference [38].

(2) How are the remaining numerical boundary conditions to be defined at the boundaries?

The simplest way is to use extrapolation for variables which are not imposed by the physical boundary conditions. A method with more theoretical base is the characteristic boundary method, which uses the

Figure 3.7: Geometry for RANS/RANS coupled computation

characteristic variables (also called Riemann invariants), or the compatibility equations. These two methods are most commonly used but many other methods have been suggested. In addition, there are many more issues regarding how the physical and numerical boundary conditions are to be formulated and discretized in order to be compatible with the order of accuracy and the stability conditions of the numerical scheme. These issues and some good references have been summarized by Hirsch [11].

Integrated RANS-RANS: Setup

For the numerical experiments, we consider a pipe with a constant diameter D and a length of $5D$. We then split the pipe into two domains. The first domain is a pipe from $x = 0D$ to $x = 3D$, the second domain a pipe from $x = 2D$ to $x = 5D$. Hence, the two domains have an overlap of $1D$ between $x = 2D$ and $x = 3D$ (Fig. 3.7).

A subsonic case is considered and the Mach number is set to 0.1. The Reynolds number is 22630. Based on this Reynolds number the mesh for each pipe was generated. The meshes of two pipes are mismatched in axial direction over the overlapped region of each pipe.

Here, boundary conditions developed for Euler equations have been applied to Navier-Stokes computations.

Four cases of subsonic inflow boundary conditions and one case of subsonic outflow boundary condition, listed in Table 3.2. have been numerically tested for single and coupled pipe calculations.

Table 3.2: Boundary Conditions for Pipe Flow.

Boundary case	Physical boundary condition	Numerical boundary condition
Inflow case 1	u, v, w and T imposed	ρ extrapolated
Inflow case 2	u, v, w and ρ imposed	p(or T) extrapolated
Inflow case 3	p_t, T_t, and inflow angles imposed	Characteristic Method using Riemann invariants
Inflow case 4	ρu, ρv, ρw and T imposed	ρ extrapolated
Outflow	p_{exit} imposed	Extrapolation

where u, v, w–flow velocity in x, y, z directions, T–temperature, ρ–density,p–pressure, p_t–total pressure, T_t–total temperature, and p_{exit}–exit pressure.

Reference Test Cases: Single RANS and coupled RANS-RANS

Before investigating the RANS interface inflow and outflow boundary conditions, the first three inflow cases and the outflow case in Table 3.2 were tested for a single pipe with the same diameter, D and a total length of $5D$. These boundary conditions for a single pipe calculation are then typical inlet and exit boundary conditions for a RANS flow simulation. These computations allow an assessment of the accuracy of the subsequent coupled simulations. Numerical results show that the solutions of the single pipe calculations are converged well with all the boundary cases tested. For example, with the inflow case 3 and outflow boundary conditions the solution converges to a relative error of 10^{-8} in 5000 multigrid cycles. The pipe

flow is fully developed as shown in the pressure contours and the mass flow rate is effectively constant, while ρu varies locally (Fig. 3.8).

A RANS-RANS computation has also been carried out by coupling two TFLO codes. For this case, the inflow boundary case 3 was selected for the inlet of pipe 1 (the pipe from $x = 0D$ to $x = 3D$) and the only outflow was used for the exit of pipe 2 (the pipe from $x = 2D$ to $x = 5D$). This inlet and exit boundary condition setting will be the same for all the coupled computations presented in this report for consistency between single and coupled computations. All the necessary data was exchanged at the interfaces in this test. While the exchange of all data is not possible for coupled RANS-LES computations, since the low-Mach number LES code cannot provide all necessary data, this test case allows to demonstrate the accuracy of the coupling approach itself. Here, the coupled computation is similar to a classic domain decomposition. The relative convergence error of this coupled case is three orders of magnitude higher than that of the single pipe test case, which results in the offset of mass flow rate from the single pipe case as shown in Figure 3.9. The amount of perturbation in mass flow rate of each pipe for this coupled case is less than that of the single pipe case, but there is a difference over the overlap region between pipe 1 and pipe 2. This difference is due to the numerical error resulting from a combination of the difference in convergence between the two pipes and the linear interpolation scheme used for data communication. However, the total value is very small at 0.0273% of the average value of the mass flow rate.

The results from these single and coupled pipe calculations will be used as the base reference for the remainder of the current study. The feasibility of using the RANS boundary conditions for the interface boundaries of coupled simulations has been investigated and the results will be presented in the following subsections.

RANS Outflow Interface Boundary Condition

In order to test the outflow interface boundary condition, a RANS-RANS (TFLO-TFLO) coupled computation was carried out such that pipe 1 received the pressure from pipe 2 and provided all data needed by pipe 2. This allows to have a well-posed problem in pipe 2 similar to the previous test-case, and all effects can be associated with the variation of the exit boundary condition in pipe 1.

For a well-posed problem in pipe 1, the data for the remaining primitive variables, ρ, u, v, and w, at the interface boundary of pipe 1 were updated by zero order extrapolation from the first interior cells of pipe 1. This interface outflow boundary treatment is essentially a typical exit boundary treatment of a RANS calculation. The only difference is that the pressure at the interface of pipe 1 was imposed by the value at the same location in the pipe 2 instead of imposing the exit pressure, p_{exit}, which is normally specified by a known experimental value. As shown in Figure 3.10, the solution converged well, but approximately one order of magnitude higher than the previous coupled case, in which all variables required at the interface boundaries were exchanged. Although the mass flow rate deviation is higher (0.0754%), the error is still in an acceptable range.

RANS Inflow Interface Boundary Conditions

The interface inflow boundary condition, case 1 in Table 3.2, was tested similarly(Fig. 3.11). Here, T, u, v, and w were imposed for the pipe 2 interface using the values from pipe 1 and the density, ρ, was extrapolated for pipe 2. At the exit of pipe 1, the entire data set was imposed. This test case allows a well-posed problem in pipe 1, and the solution is now only influenced by the choice of inflow boundary conditions for pipe 2.

This coupled solution converged with the same order of convergence accuracy as the reference case. Interestingly the difference in mass flow between two pipes are even smaller (0.0121%) than for the reference solution.

The inflow boundary case 2 in Table 3.2 was tested and similar results were obtained as shown in Figure 3.12. This interface inflow boundary condition guarantees a mass conservation across the interface, if there is no interpolation error. However, this boundary condition may not be useful, since in Low-Mach number LES flow solvers, the density is not a function of the flow, and hence, does not represent a fully compressible solution.

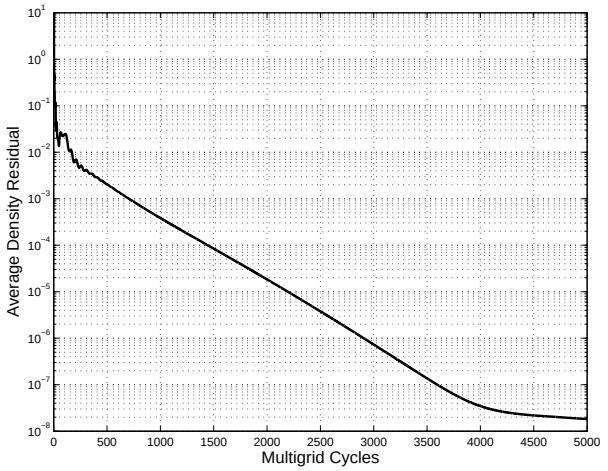
Currently in the applications of LES-RANS integration, the mass-flux vector at every point of the inlet is being specified corresponding to the value delivered by the LES computation. This means that ρu, ρv, ρw and T are imposed and ρ is extrapolated at the boundary. This allows the density ρ to fluctuate to account for the passing of acoustic waves. The velocity components u, v and w are adjusted accordingly, in order to conserve the mass-flux. This current implementation, case 4, was also tested and the smallest difference in mass flow rate was observed as shown in Figure 3.13.

Justification of RANS Interface Boundary Conditions

In the final set of simulations, the tested boundary conditions are applied on both sides of the domain simultaneously. The inflow boundary case 1 for pipe 2 and the outflow boundary condition was applied at the exit of pipe 1 (Fig. 3.14). As expected, the relative convergence error increases slightly. However, the error in mass flow rate is still low (0.0143%) and within engineering accuracy. The fact that the error is even lower than in the reference test-case demonstrates that the achieved levels of mass conservation are within machine accuracy.

In order to demonstrate the necessity of the proposed boundary treatment, an additional simulation was performed. For this, we exchanged the same data set between the two codes, but instead of applying RANS interface boundary conditions, the missing variables are updated by the numerical scheme. As can be seen in Fig. 3.15, the error in mass flux is increased by two orders of magnitude and the flow solution in the pipe differs considerably from all previous solutions.

Similarly, the inflow condition, case 2, and the outflow condition were tested and the results are shown in Figure 3.16. Interestingly, we can observe that the solution of this test case was dependent on the choice of variables (p or T) to be extrapolated for pipe 2. With pressure chosen as extrapolated variable the solution became unstable. A theoretical reason has not been found but the result is shown in Figure 3.17.



pressure contour

x−momentum contour

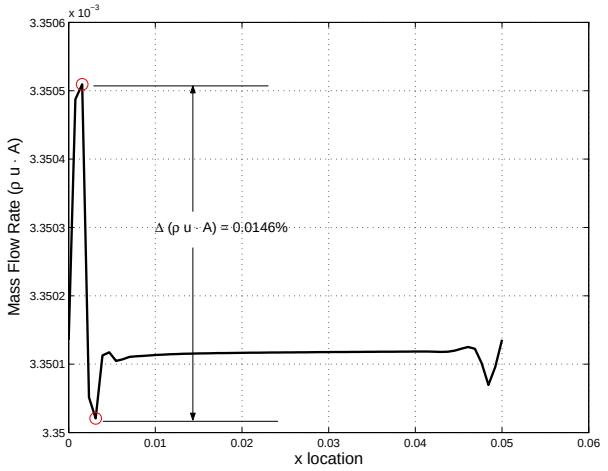
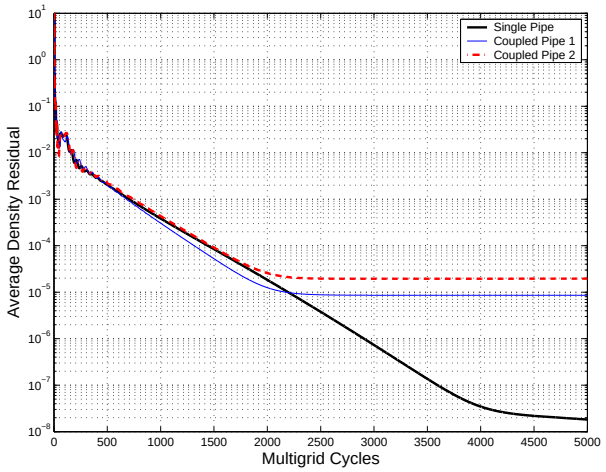
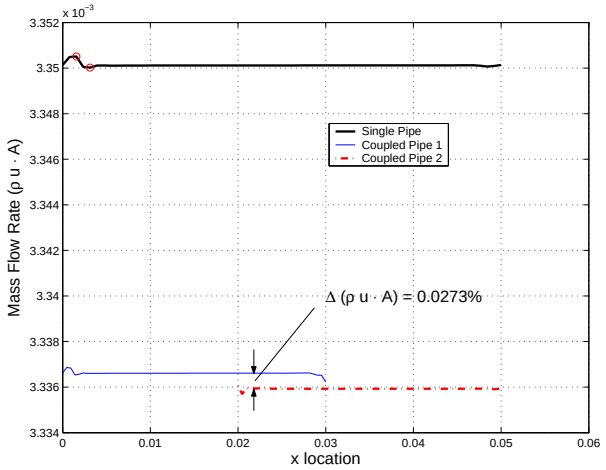


Figure 3.8: Single TFLO computation: Convergence history, pressure contour, x-momentum (ρu) contour and mass flow rate





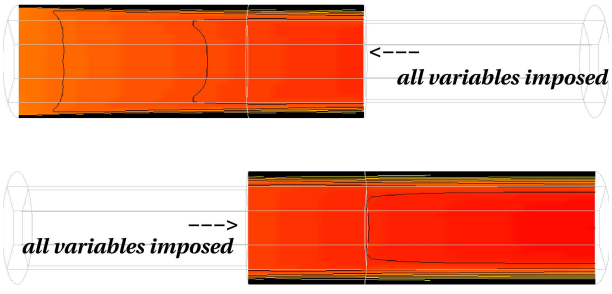
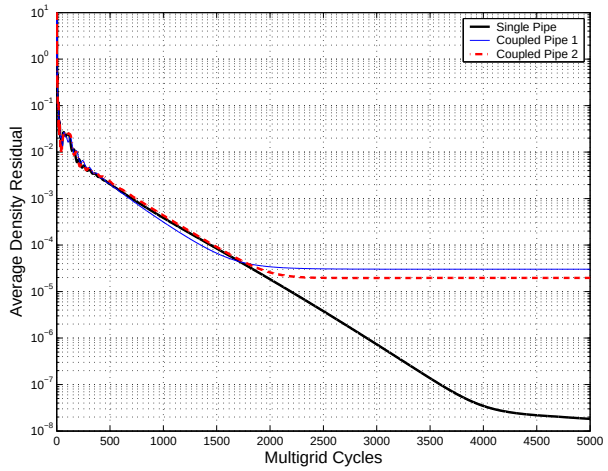
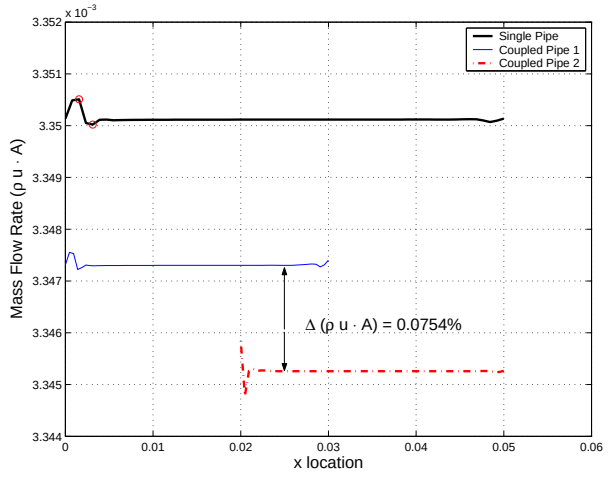


Figure 3.9: TFLO-TFLO coupled computation exchanging all variables: Convergence history, mass flow rate and x-momentum (ρu) contour





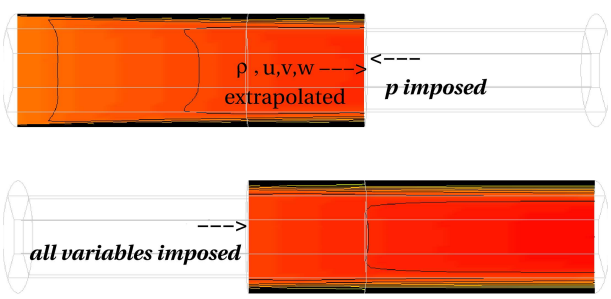
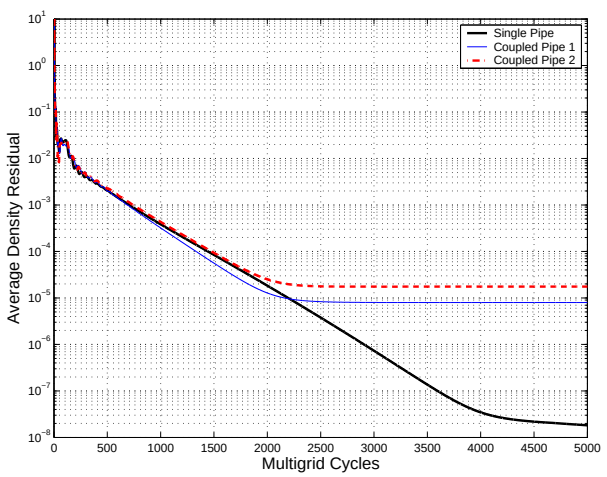
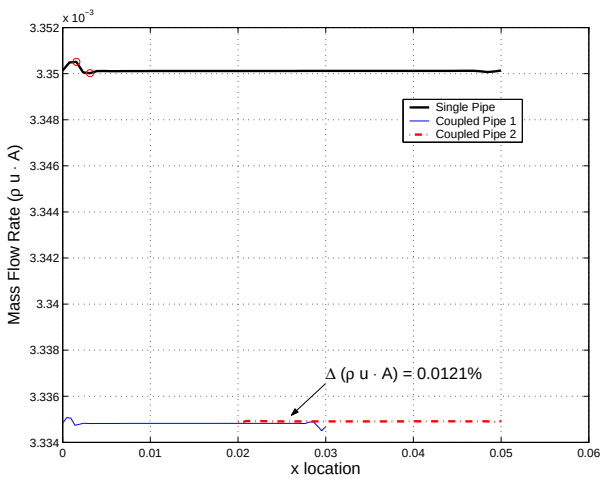


Figure 3.10: TFLO-TFLO coupled computation for testing outflow boundary condition: Convergence history, mass flow rate and x-momentum (ρu) contour





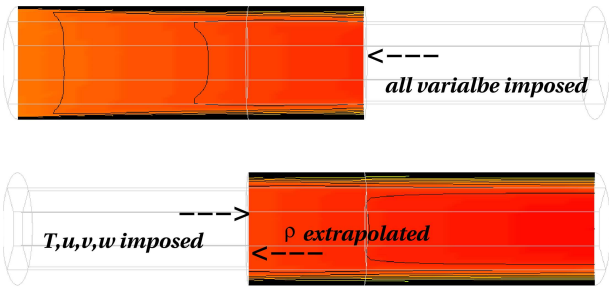
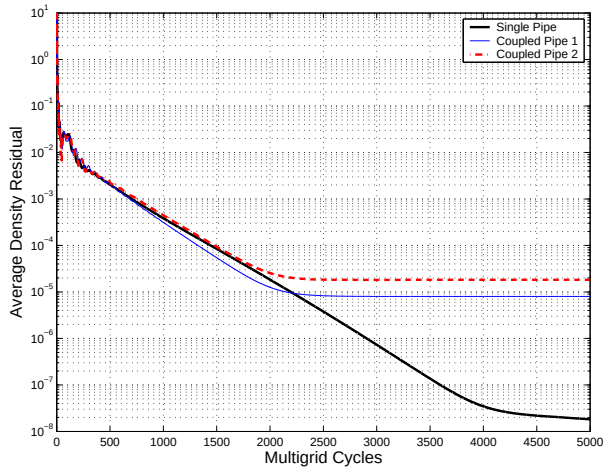
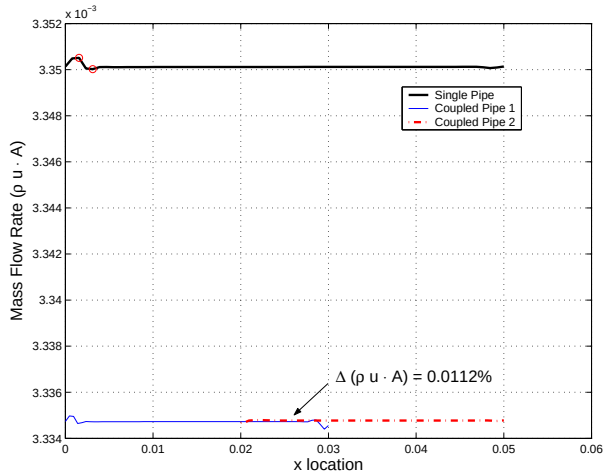


Figure 3.11: TFLO-TFLO coupled computation for testing interface inflow boundary condtion, case 1: Convergence history, mass flow rate and x-momentum (ρu) contour





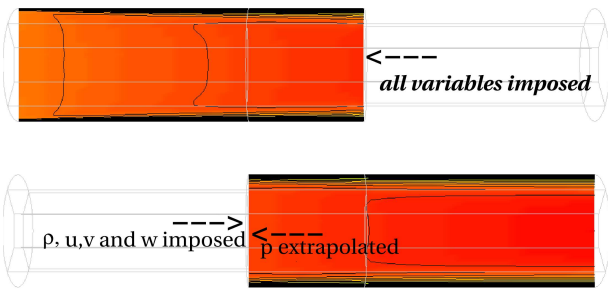
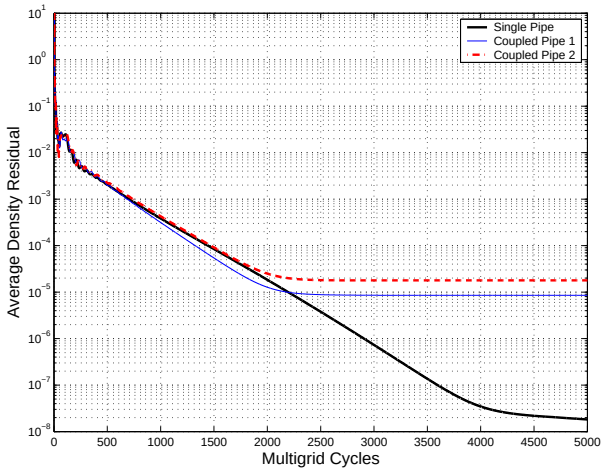
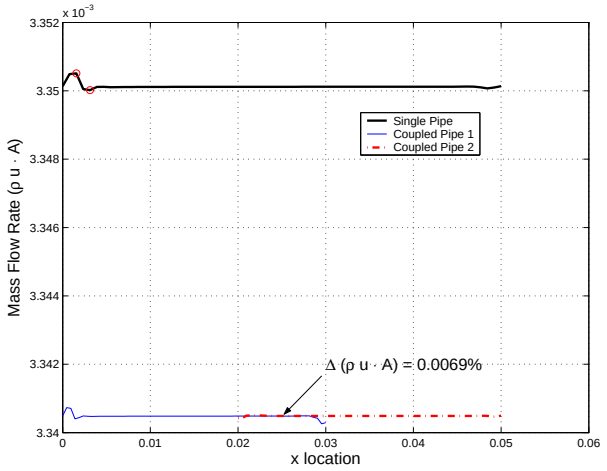


Figure 3.12: TFLO-TFLO coupled computation for testing interface inflow boundary condition, case 2: Convergence history, mass flow rate and x-momentum (ρu) contour





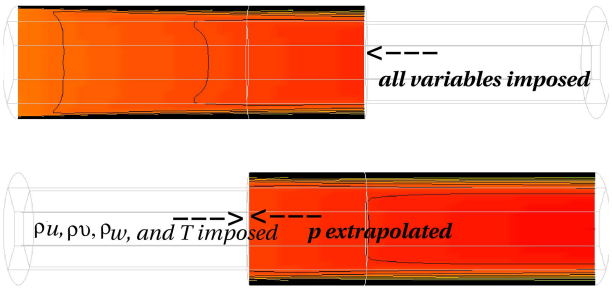
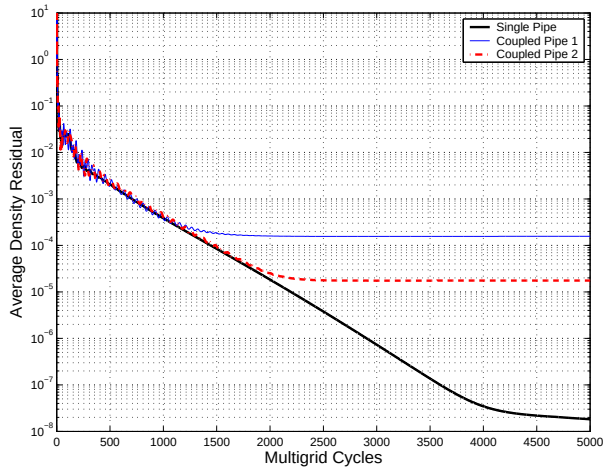
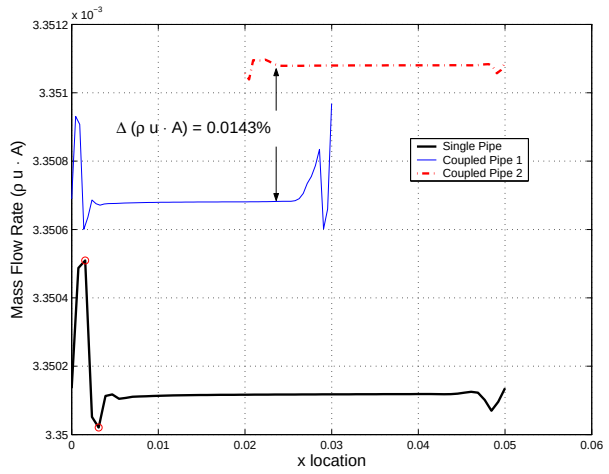


Figure 3.13: TFLO-TFLO coupled computation for testing interface inflow boundary condition, case 4: Convergence history, mass flow rate and x-momentum (ρu) contour





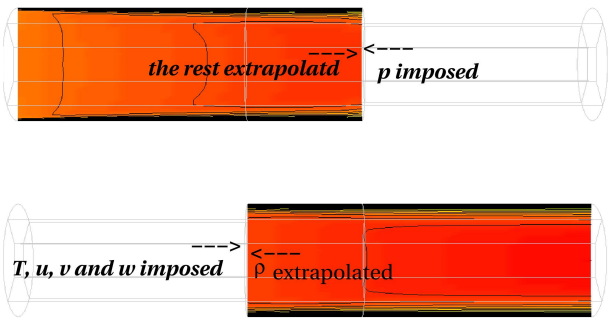
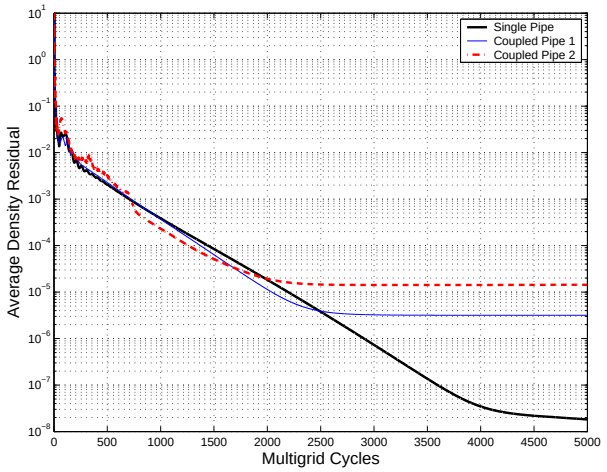
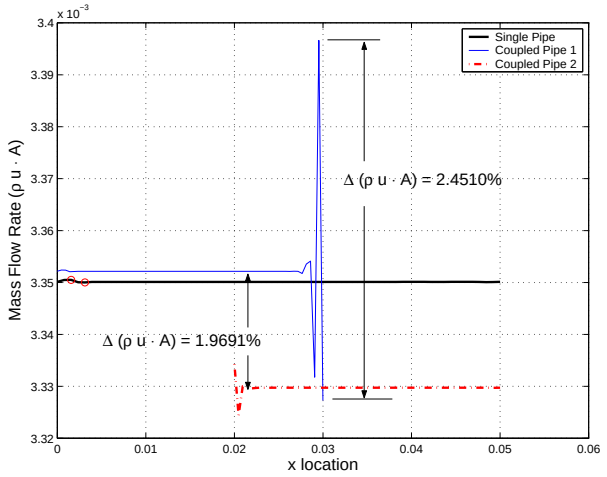


Figure 3.14: TFLO-TFLO coupled computation for testing inflow (case 1)/outflow boundary condition test: Convergence history, mass flow rate and x-momentum (ρu) contour





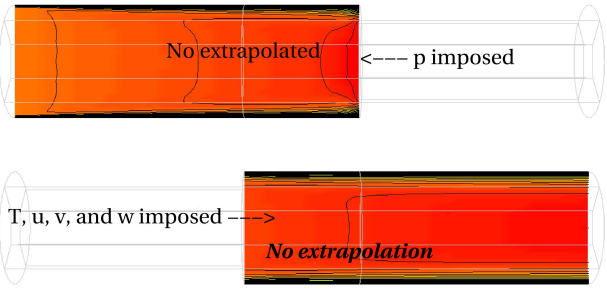


Figure 3.15: TFLO-TFLO coupled computation exchange T, u, v and w for p without boundary treatment: Convergence history, mass flow rate and x-momentum (ρu) contour

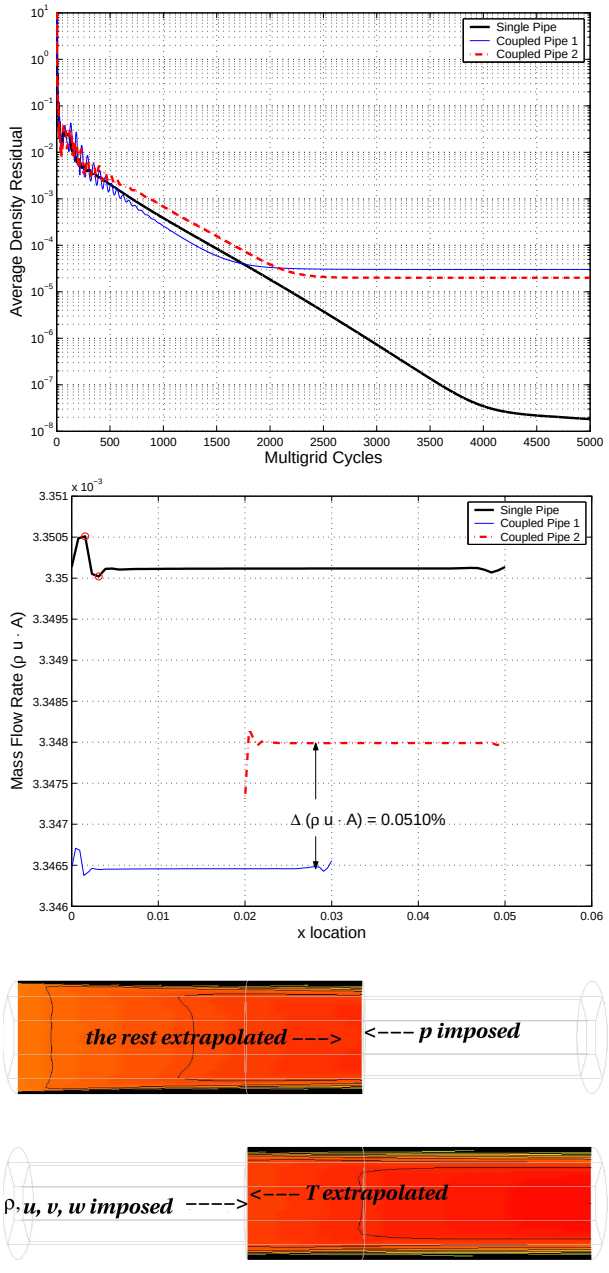


Figure 3.16: TFLO-TFLO coupled computation for testing interface inflow (case 2 with T extrapolation)/outflow boundary conditions: Convergence history, mass flow rate and x-momentum (ρu) contour

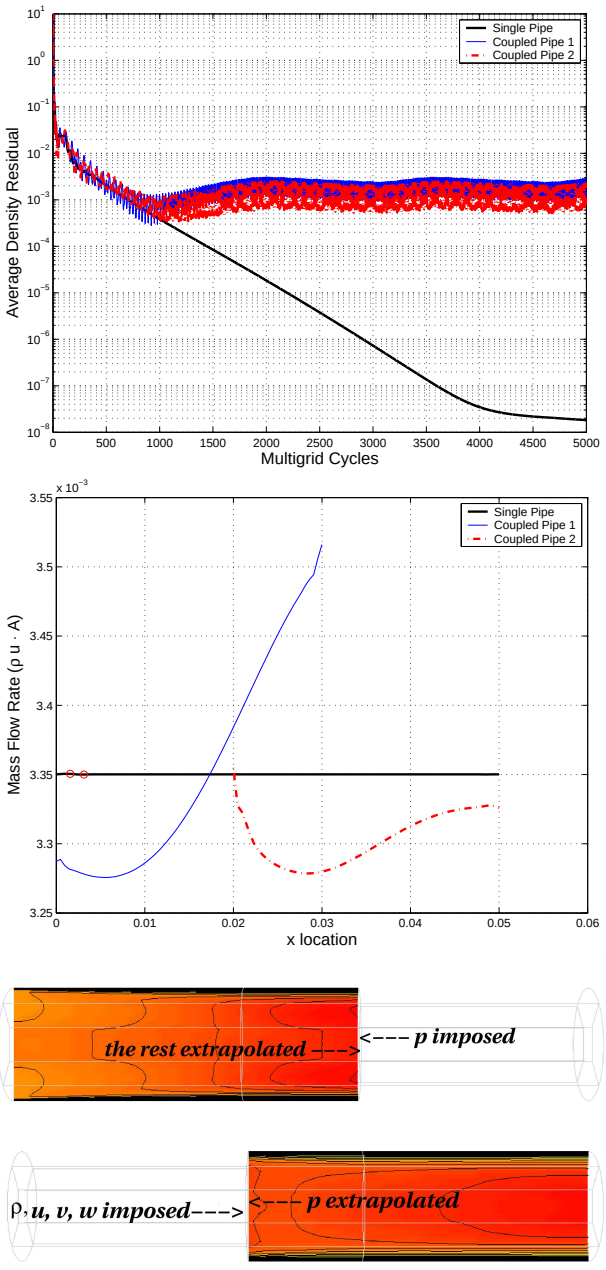


Figure 3.17: TFLO-TFLO coupled computation for testing interface inflow(case 2 with p extrapolation)/outflow boundary conditions: Convergence history, mass flow rate and x-momentum (ρu) contour

3.4.2 Improving Unsteady Compressible Boundary Conditions

The decomposition of a given flow domain into several domains will always introduce artificial boundaries. Since in our case, we use different flow solvers with differences in the approaches to turbulence modeling as well as in the treatment of compressibility, there is a need for the development of truly non-reflecting compressible boundary conditions. Although the current LES approach is based on a variable density approach, and hence, does not account for acoustic waves, future developments of the LES code will have to include acoustic wave propagation [41], especially for the investigation of combustion instabilities. For that reason, the proper specification of compressible boundary conditions is important for both, LES and URANS approaches.

The definition of unsteady, compressible boundary conditions is an open problem. The specification of time dependent boundary conditions for such flows poses a challenge to many of the currently available methods, which are mostly based on the linearized Euler equations. In order to be universal, these boundary conditions have to be local, which means the definition of local quantities does not depend on global quantities or global correction functions. Several studies used local methods to derive non-reflecting boundary conditions, such as the PML approach [12, 9] and a number of schemes based on the method of characteristics derived from Euler equations [39, 40, 8, 26, 27]. These efforts resulted in the most commonly used class of compressible boundary conditions, the **N**avier-**S**tokes **C**haracteristic **B**oundary **C**onditions (NSCBC) and the **L**ocal **O**ne **D**imensional **I**nviscid (LODI) approach [22]. These approaches are currently the state-of-the-art for engineering computations.

Improved boundary conditions

In the following study, we want to improve the boundary conditions following the work of Prosser [23, 24]. In the governing equations, each variable is written as a low Mach number expansion [17], i.e. the pressure is written as $p = p^{(0)} + Mp^{(1)} + M^2 p^{(2)} + O\left(M^3\right)$. Bracketed, superscripted numbers are used to index the terms in a given expansion. Then the convective flux terms are decomposed into inertial scales $\mathbf{x} = (x\ y)^T$ and acoustic scales $\eta = (\xi\ \theta)^T$, $\xi = Mx$, $\theta = My$. The convective derivatives appearing in the Navier-Stokes equations are consequently expressed as (in the case of the non-dimensionalized $\widetilde{x}$ derivative [15]:

$$\left.\frac{\partial}{\partial \widetilde{x}}\right|_{M,t} = \frac{\partial}{\partial x} + M\frac{\partial}{\partial \xi}. \tag{3.16}$$

The expansions are inserted into the primitive equations and the associated characteristics. By appropriate matching of terms, revised treatments for time dependant boundary conditions can be derived that better satisfy the demands of mass, momentum and energy conservation.

For a complete derivation of the boundary conditions we refer to [23, 24]. In short, it can be shown that the characteristic incoming wave at the outflow can be rewritten as:

$$\begin{aligned} L_1 &= L_4 + \frac{(\gamma-1)\,T}{a}\left(v\frac{\partial u}{\partial y} - u_b\frac{\partial u}{\partial x} - \left(\frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y}\right)\right) \\ &- (\gamma-1)\,T\frac{\partial u^{(0)}}{\partial \xi}, \end{aligned} \tag{3.17}$$

where

$$\frac{\partial u^{(0)}}{\partial \xi} = \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) - \left(\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)\right). \tag{3.18}$$

If the flow is cold, then it is straightforward to show that $T^{(0)} = const.$, and the thermal conduction term can be dropped from the preceding equation for $\partial u^{(0)}/\partial \xi$.

Improved boundary conditions: test-case and flow solver

We now want to test the boundary conditions for some simple flow configurations. For this investigation, we are using a structured compressible LES flow solver developed by Pierce [20] and Wall [41], which uses a second-order finite-volume scheme on a staggered grid [1]. The sub-grid stresses are approximated by a dynamic procedure [7, 19]. The pressure and density field are determined by solving the Helmholtz equation. Although the time-step can be chosen more or less arbitrarily, and the time-step was set to satisfy the acoustic CFL condition in order to be fully compatible with explicit compressible approaches.

Using the LES flow solver, the two test-cases were computed. The first test case is that of a single vortex leaving the domain over the outlet boundary. The flow field was initialized with the following parameters:

$$\begin{aligned} \psi &= C\left(\exp\left(-\frac{r_a^2}{2r_v^2}\right)\right) + u_b y \\ r_a &= \left((\widetilde{x}+0.5)^2 + (\widetilde{y}-0.15)^2\right)^{\frac{1}{2}} \\ u &= \frac{\partial \psi}{\partial \widetilde{y}} \\ v &= -\frac{\partial \psi}{\partial \widetilde{x}} \end{aligned} \tag{3.19}$$

C is the vortex strength, u_b is the mean velocity and r_v is a characteristic radius. For this simulation, C was set at $0.5\text{m}^2/\text{s}^2$, r_v was set at 8% of the domain size and the mean velocity u_b was set at 10m/s.

The second test case is that of a *heated* co-rotating vortex pair. Here the initial conditions are determined by:

$$\begin{aligned} \psi &= C\left(\exp\left(-\frac{r_a^2}{2r_v^2}\right) + \exp\left(-\frac{r_b^2}{2r_v^2}\right)\right) + u_b y \\ r_a &= \left((\widetilde{x}+0.5)^2 + (\widetilde{y}-0.15)^2\right)^{\frac{1}{2}} \\ r_b &= \left((\widetilde{x}+0.5)^2 + (\widetilde{y}+0.15)^2\right)^{\frac{1}{2}} \\ u &= \frac{\partial \psi}{\partial \widetilde{y}} \\ v &= -\frac{\partial \psi}{\partial \widetilde{x}} \\ T &= 300\left(1 + \exp\left(-\frac{r_a^2}{2r_v^2}\right) + \exp\left(-\frac{r_b^2}{2r_v^2}\right)\right). \end{aligned} \tag{3.20}$$

The values for the vortex strength C, mean velocity u_b and characteristic radius r_v are identical to the previous case.

The LES mesh size was $128 \times 128 \times 8$ nodes. The discretization in z-direction was intentionally left coarse in order to compute a quasi-2D flow. The mesh is concentrated near the path of the vortices. In transverse direction non-slip wall conditions were applied.

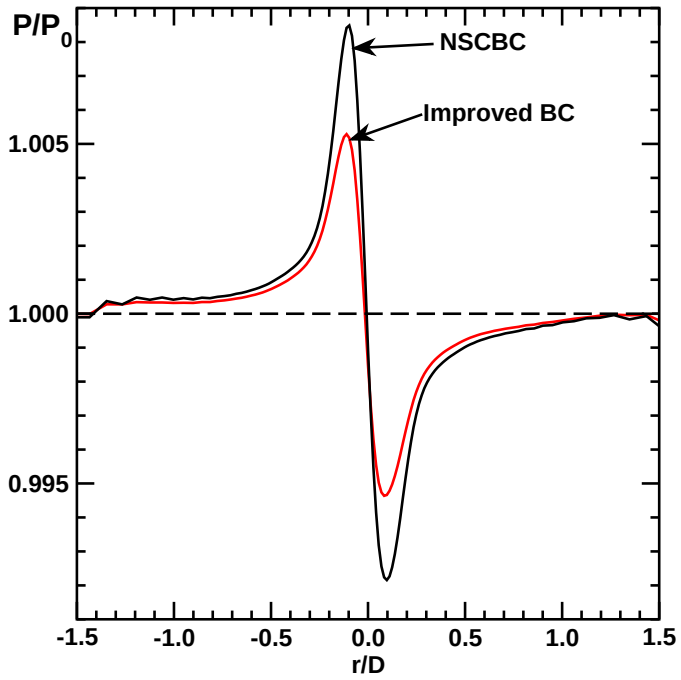


Figure 3.18: LES computation of single vortex passing outflow: cross-section pressure distribution at the instant of maximum pressure distortion.

Improved boundary conditions: Results

Figure 3.19 shows the pressure distribution at the outlet cross section at the instant of maximum distortion for the first test case, a single vortex passing the outflow boundary. We can see that the outflow boundary has a considerable effect on the pressure distribution. Inside the domain, the vortex core has a pressure drop of about 1% of the ambient pressure. The pressure distortion due to the outflow boundary is a multiple of that. We can also note that the new improved formulation of the outflow boundary decreases the impact of the outflow boundary substantially.

We now want to have a look at a flow with temperature variations. This is an interesting case for our long term project, since the outflow of the combustor will have strong temperature gradients along the boundary plane. Figure 3.19 shows the pressure distribution at the outlet cross section at the instant of maximum distortion for the case of the heated co-rotating vortices. Again, we note that the pressure distortion is much more attenuated using Eq. (3.17) compared to the standard NSCBC. Figure 3.19 also shows the pressure distortion using a convective outflow boundary condition. This or similar boundary conditions are currently still in use for steady flow computations. This boundary condition is apparently unsuitable for unsteady simulations.

We performed additional LES computations without a subgrid model in order to assess the effect of the additional viscous terms of the subgrid model. The results of these computations were virtually indistinguishable from the previous results, which means that for the chosen test case, the influence of the subgrid model on the boundary conditions is negligible.

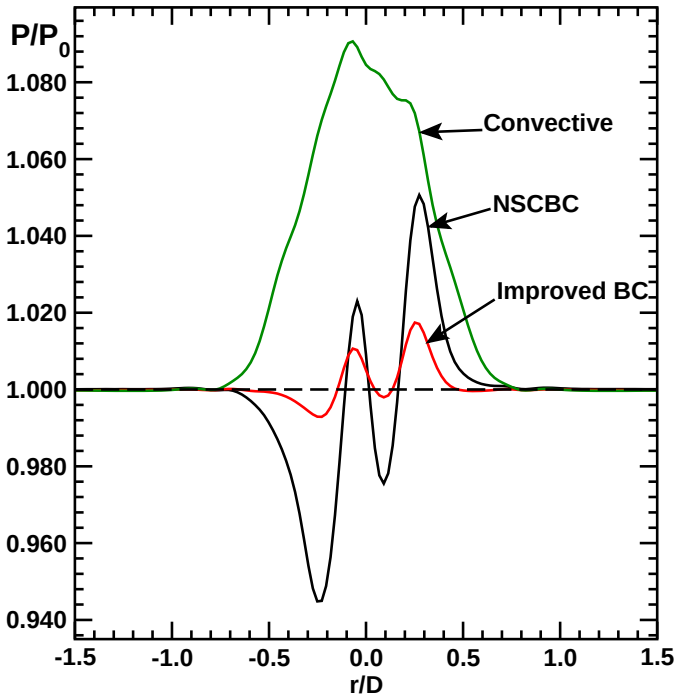


Figure 3.19: LES computation of heated co-rotating vortices passing outflow: cross-section pressure distribution at the instant of maximum pressure distortion.

3.4.3 Separated Diffuser

Before we turn our focus to the application of integrated RANS-LES, we want to assess the accuracy of the LES side of the chosen application test case of a compressor and a prediffuser of a gas turbine engine. For the design of the prediffuser, it is crucial to be able to predict flow separation. For this reason, as a validation study, a slightly separated plane diffuser was computed [43]. The geometry (Fig. 3.20) corresponds to that of an experiment by Buice [4] and extensive experimental data are available for this case.

In order to assess the accuracy of the LES flow solver and its sensitivity to mesh resolution, two different meshes were used. One consists of 2.5M cells and its resolution is equivalent to that in the diffuser of a combustor in the integrated simulation presented below. The other mesh is a refined mesh and consists of 7M cells.

Figure 3.21 shows the axial velocity profiles obtained by these two computations. The results are in good agreement with the experimental data. Most importantly, the onset of separation is well predicted. The highly resolved computation demonstrates only slightly better agreement than the the LES computation using a lower resolution.

3.5 Application of Integrated RANS-LES to Complex Geometries

In the this section, we will demonstrate the value of coupled RANS-LES computations for gas turbine applications. We have performed computations in two different geometries. One is the NASA Stage 35

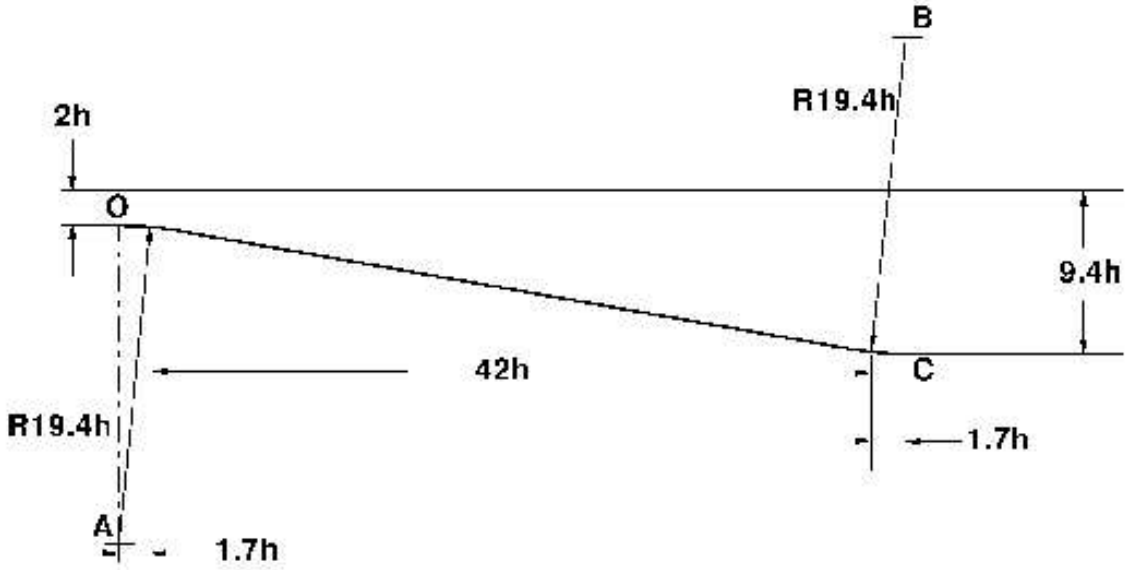


Figure 3.20: LES validation: diffuser geometry.

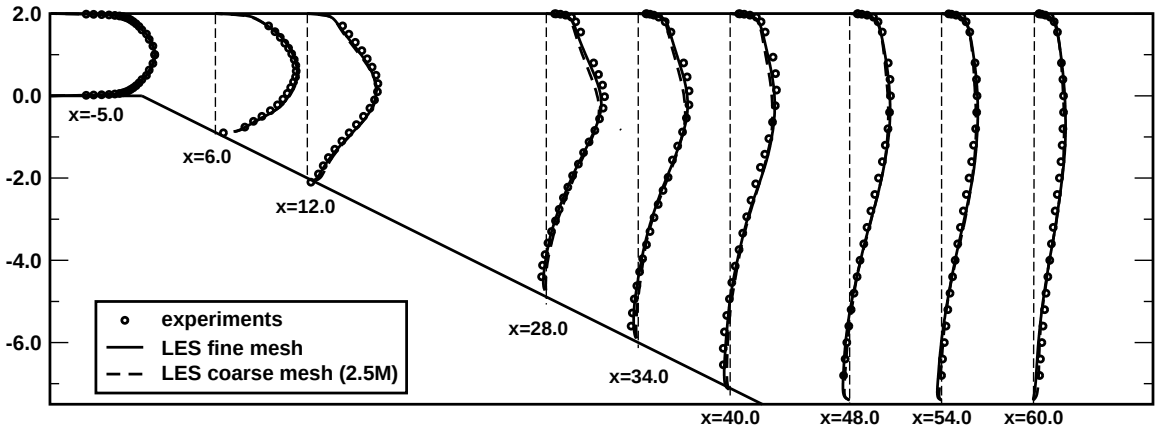


Figure 3.21: LES validation: mean axial velocity profiles.

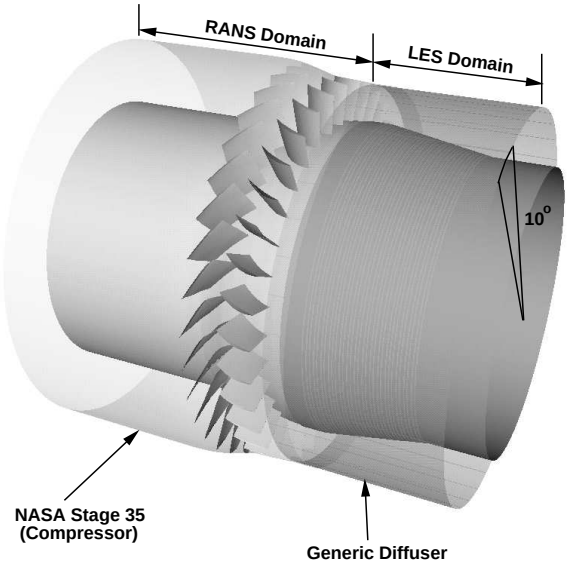


Figure 3.22: Geometry of coupled NASA stage 35/prediffuser domain.

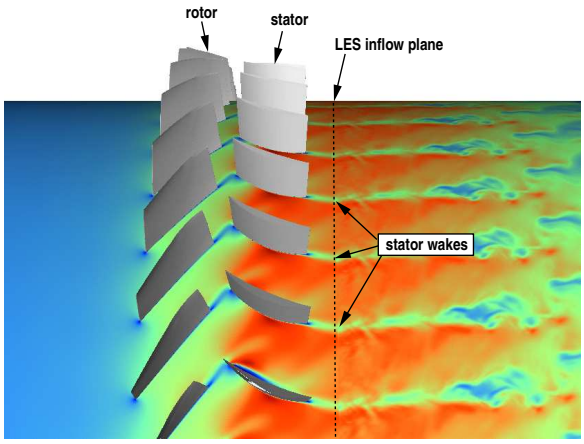


Figure 3.23: Integrated RANS-LES of compressor/prediffuser: Velocity distribution at the 50% plane. Close-up of the interface.

compressor coupled with a generic diffuser. This is a geometry we began to study in the previous year.

The other test case corresponds to a portion of a Pratt & Whitney gas turbine engine. We investigate the flow of the last stage of the compressor coupled with the entire combustor. Here, we were able to demonstrate for the first time the application of the coupled RANS-LES approach in a real industrial application.

3.5.1 NASA Stage 35/Diffuser

The test-case is that of the NASA Stage 35 compressor that we extended behind the stators with a generic diffuser-geometry, resembling that of a combustor inlet diffuser in an aircraft engine. This study has been initiated already in the previous reporting period [37, 36]. Now, we want to focus on demonstrating the advantages of the coupled RANS-LES approach in comparison to a single component simulation.

3.5.2 Geometry

The compressor geometry for the computed test-case corresponds to that of a modified NASA experimental rig stage 35. The experimental rig consists of a row of 46 rotors and a row of 36 stators. In order to simplify this geometry, the rotor stage has been rescaled to a 36 blade count, which allows to compute an axisymmetric segment of 10° using periodic boundary conditions at the corresponding azimuthal planes.

For this integrated computation, the rotor tip-gap has been closed in order to decrease the overall computational costs. The inclusion of the tip-gap is addressed in the TFLO flow solver and poses no additional problem from the integration point of view. The RANS time step was chosen to resolve one blade passing with 50 intervals.

On the LES side, we use a structured LES flow solver. A more comprehensive description of the geometry, the mesh topology and the flow conditions, as well as some results on the main flow features (see also Fig. 3.23) can be found in last years report as well as in [37].

Here, we want to assess the value of integrated RANS-LES simulations for this geometry. We have performed an integrated RANS-LES simulation of the entire domain. The results of this simulation will compared with those of a second simulation of only the diffuser geometry, where the time-average of the computed flow field from the first simulation at the diffuser inlet are used as inlet conditions. For the diffuser side, the difference between these two simulations is that in the integrated RANS-LES computation the diffuser inlet flow is unsteady in the mean, while it is steady in the mean in the single component simulation. This comparison will provide an assessment of the importance of the flow development in the compressor on the diffuser flow.

3.5.3 Results

The integrated RANS-LES computations were carried out using 64 processors for TFLO and 16 processors for the structured LES flow solver. Eight blade passings were computed in 60 hours of wall clock time using an IBM Power3. The single component diffuser computations were performed on 32 processors in 10 hours wall clock time computing a physical time-span equivalent to 5 blade passings.

Figure 3.24 compares the flow development in the diffuser for the two different simulations. The solid lines correspond to the simulation of the single component diffuser simulation. The inlet velocity profile and the level of turbulence has been specified according to the time-averaged RANS solution at the outlet of the compressor. The profiles of velocity fluctuations contain the turbulence modeled by the RANS turbulence model as well as the long-wave flow modulation, which is resolved in the RANS computation. The second set of data (dashed lines) is retrieved from the LES domain of the coupled RANS-LES computation, which means that at each RANS time step the LES inflow is updated according to the unsteady solution in the compressor.

Comparing the velocity profiles at the inlet plane, we can see that both solutions are identical. However, further downstream both solutions are distinctively different.

The profiles of the velocity fluctuations show a similar behavior. At the inlet, both profiles are identical. Here, already close to the inlet the velocity fluctuations are much larger in the integrated RANS-LES computation. This can be explained with the fact that in the integrated RANS-LES unsteady flow features from the compressor are transfered to the LES. The production of turbulent kinetic energy is determined as: $P = \overline{u_i' u_j'} \frac{\partial u_i'}{\partial x_j}$. The unsteadiness creates higher local gradients that are convected into the LES domain and the turbulence production increases in the case of the integrated RANS-LES. In the current case, the additional turbulence production delivers a different turbulence field which results in a different mean flow field than in the LES only computation. We conclude that in the current case of the NASA Stage 35/diffuser the use of an integrated RANS-LES can improve the prediction of the diffuser flow.

3.5.4 Pratt & Whitney Engine Geometry

In the previous sections we presented the coupled RANS-LES approach and its application to a simplified compressor geometry. In this section, we want to demonstrate this approach on a real engine geometry.

The geometry considered is that of a Pratt & Whitney aircraft engine (Fig. 3.25 & 3.26). Here, we present a simulation at the last stage of the high pressure compressor consisting of one rotor and the exit guide vanes (EGV) using the RANS approach. This RANS simulation is coupled with a LES of the prediffuser and the entire combustor. We chose to simulate the entire combustor including the fuel injector, since the flow blockage by the fuel injector and the resulting flow split is considered to be important for the performance

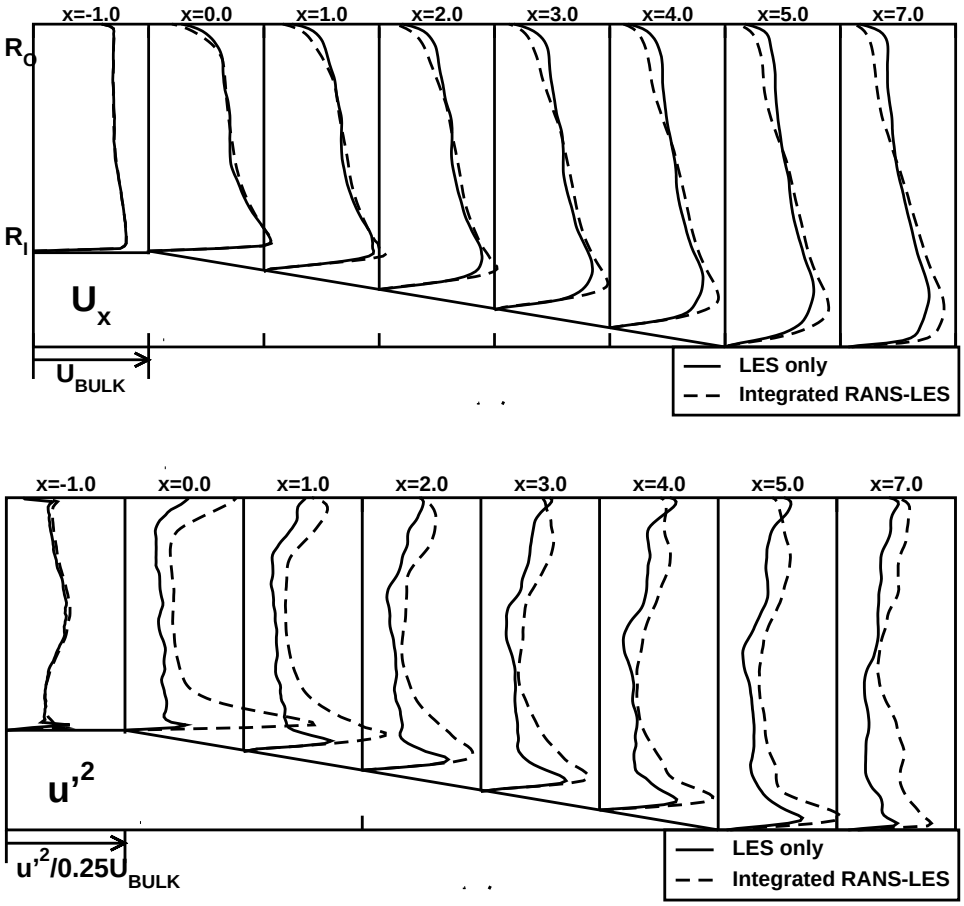


Figure 3.24: Velocity profiles in the diffuser; Solid lines: LES only; Dashed lines: RANS-LES; Above: axial velocity; Below: axial velocity fluctuations

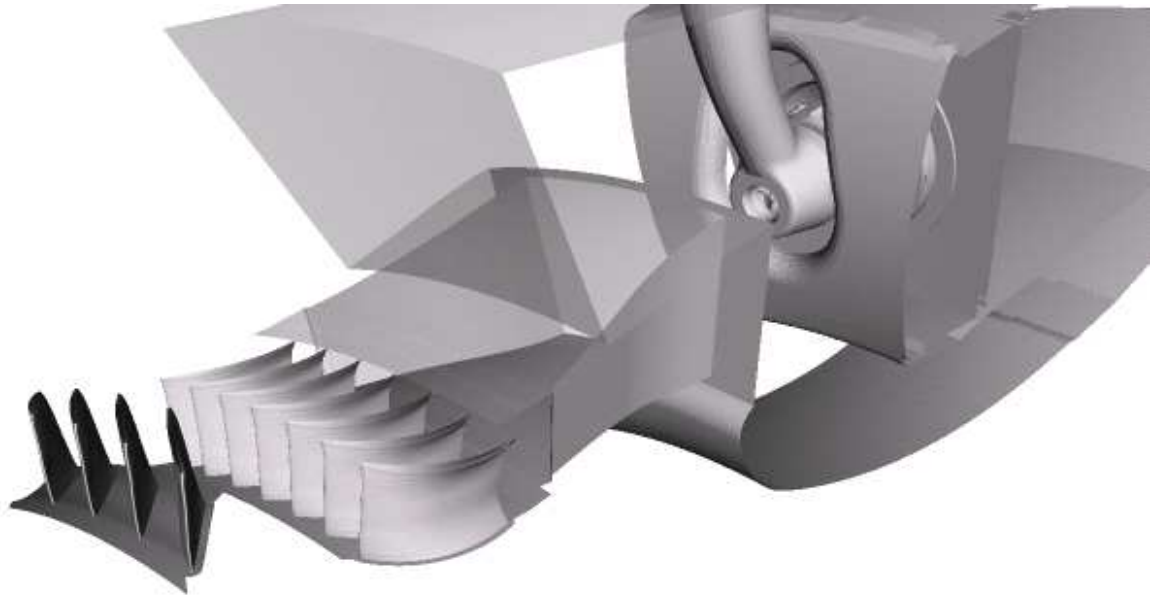


Figure 3.25: Geometry of Pratt & Whitney test case: In the front, the segment of the compressor (4 rotor blades, 7 exit guide vanes) is shown. The flow leaving the compressor enters the prediffuser box before it enters a plenum in front of the fuel injector.

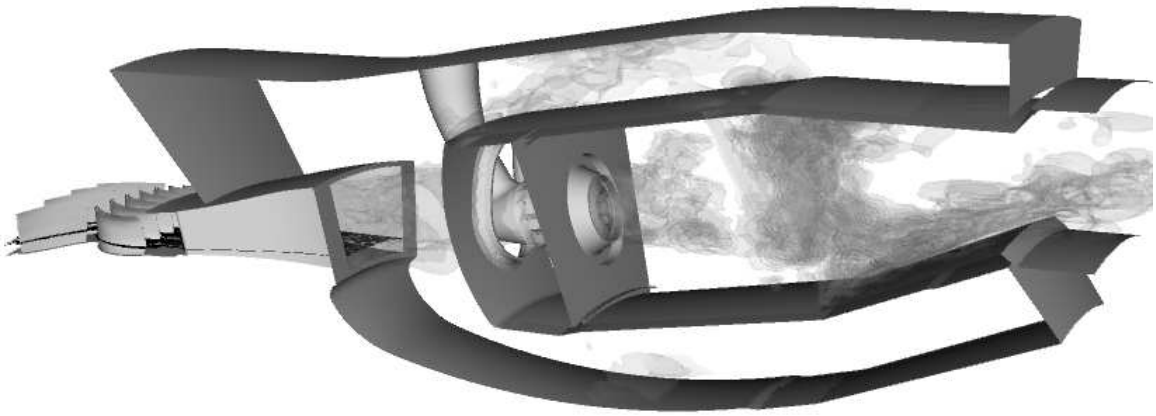


Figure 3.26: Geometry and flow visualization in the combustor. Note the compressor stage upstream of the diffuser. Smoke visualization demonstrates flow features of the cold flow.

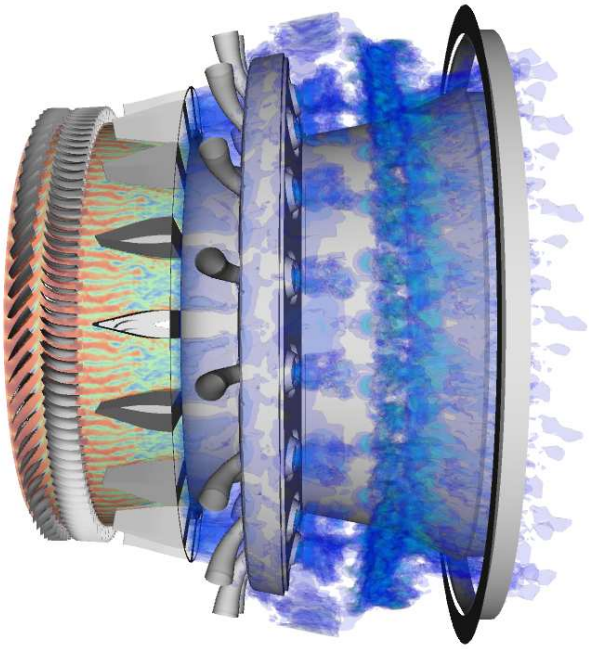


Figure 3.27: Isocontours of the axial velocity at the 50% plane in the compressor and diffuser. Isosurfaces of axial velocity in the combustor. A 20° segment is computed.

of the diffuser [2, 3, 14]. However, the flow in the combustion chamber is non-reactive. The computation of reactive flows has been demonstrated already for this geometry [18], but we consider it as not necessary for the purpose of the present demonstration.

The geometry is a 20° segment of the full engine geometry, which means that we compute one fuel injector. The blade count of the last stage of the compressor was rescaled to fit the 20° segment, and four rotor blades and seven exit guide vanes are computed in total. The RANS mesh consists of 500,000 cells in a structured multi-block mesh.

The complexity of the combustor geometry requires the use of the unstructured LES flow solver CDP. The combustor mesh consists of 3,000,000 unstructured mesh cells and is refined in the diffuser part. The mesh resolution in the prediffuser is was chosen to correspond approximately the coarse mesh resolution of the diffuser simulation (section 3.4.3).

The computation of 10 blade passings was performed using 128 processors on an IBM SP3. One blade passing needed 10 hours wall clock time. The entire computation was performed within one week.

Figure 3.29 shows the axial velocity contours at the interface. As in the previous test-case, the most striking flow features are the wakes of the EGVs entering the diffuser. The flow field is highly turbulent and we can identify coherence of the turbulence due to the wake formation.

This computation of the Pratt & Whitney real engine geometry is the first of its kind. Simulations of this portion of a gas turbine usually encounter difficulties, either in the description of the compressor flow, which can be computed only by a RANS approach, or the representation of the diffuser flow, which needs to capture the possible presence of flow detachment accurately and is largely improved by the use of LES.

The current state-of-the-art in diffuser simulations uses either radial profiles obtained from experiments

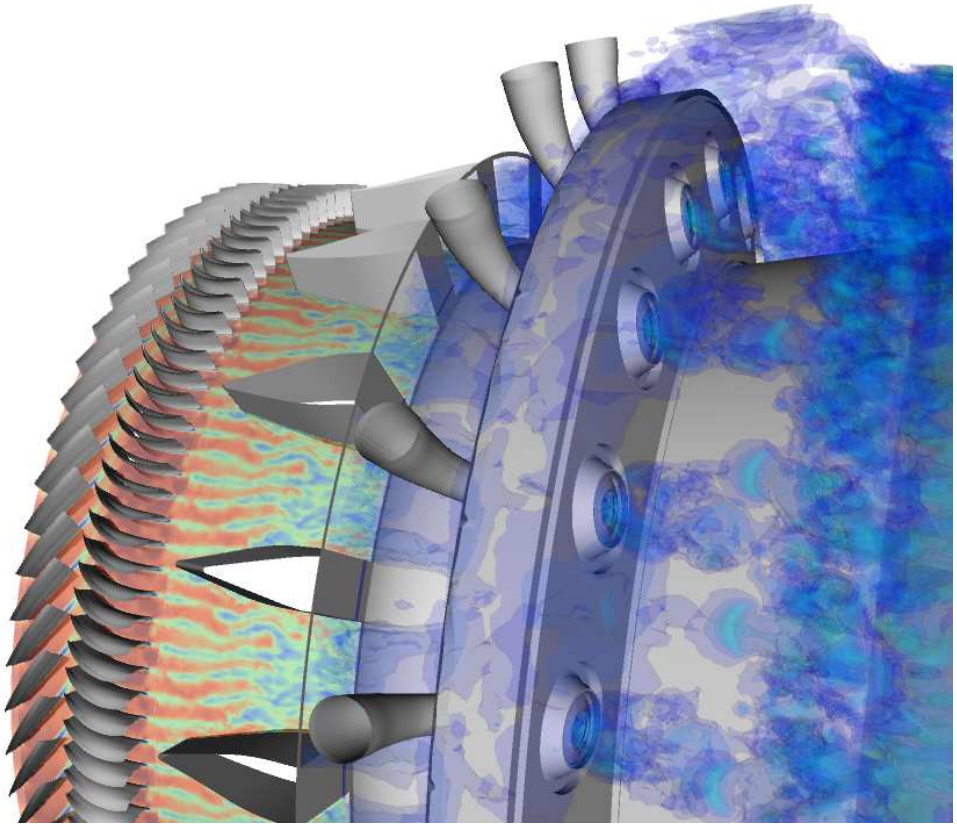


Figure 3.28: Close-up: Isocontours of the axial velocity at the 50% plane in the compressor and diffuser. Isosurfaces of axial velocity in the combustor. A 20° segment is computed.

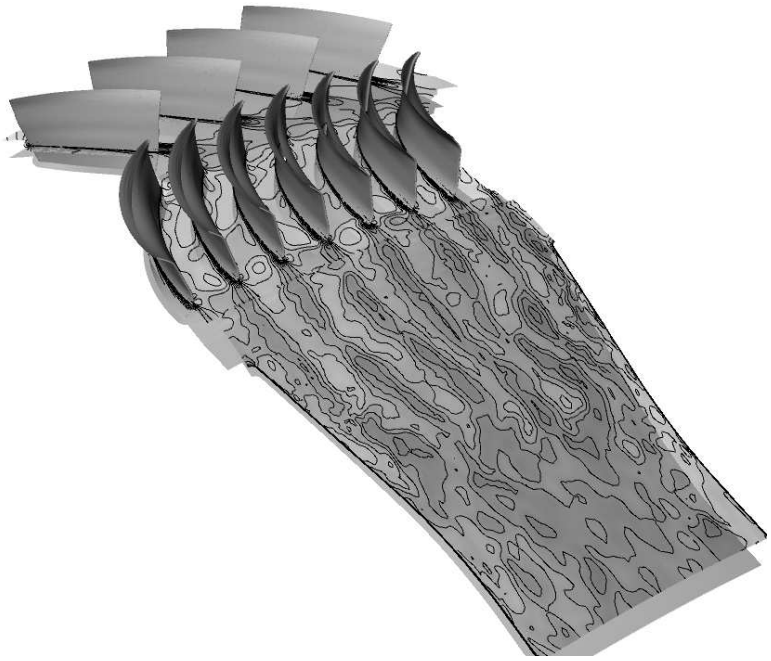


Figure 3.29: Isocontours of the axial velocity at the 50% plane in the compressor and diffuser. Close up of the interface.

or steady state data from uncoupled RANS simulations of the compressor. The integrated simulation allows not only to obtain a more precise representation of the compressor flow, but also to simulate the geometries accurately without the need of input from experiments.

3.6 Summary

In the past year we made considerable progress in three area: code implementation, basic research and application of integrated RANS-LES to gas turbine engines.

In the area of code implementation we generalized most of the approaches we developed in the previous years and, following the recommendations of the review team, implemented these into the unstructured LES flow solver CDP. Most notably, the anti-aliasing filter and LES outflow boundary conditions have been implemented into the code. Also, the implementation of the interface into TFLO 2000 has been initiated and local phase corrections, which are essential for phase conservation in multi-code simulations, have been implemented in both, CDP and TFLO.

In basic research, we focused on boundary conditions for compressible and reactive flows. Although this is a problem that we will have to revisit in the future periodically, we made considerable progress in this area.

We were also able to demonstrate the value of integrated RANS-LES simulations for compressor-diffuser computations on the NASA Stage 35-diffuser geometry. Furthermore, for the first time, we have applied the integrated RANS-LES approach to a real engine geometry, computing the last stage of a compressor coupled with a diffuser and combustor of a Pratt & Whitney engine. The computation of such a

geometry is the first of its kind and is only made possible by our unique approach of using a RANS flow solver for the compressor and LES for the diffuser and combustor.

3.7 Future Work Plan

While the computation of the coupling between the compressor and the combustor is a significant progress in assessing multi-component interactions in gas turbine engines, the more challenging case is that of the interaction between the combustor and the turbine. Based on the experience in the model assembly of the compressor-combustor case, we will create meshes for the combustor-turbine geometry and initiate computations.

Basic research will be continued on unsteady compressible boundary conditions and computations of reactive LES coupled with RANS will be performed.

Another area we would like to investigate is the definition of convergence criteria for integrated multi-code simulations. Noting that the convergence of coupled simulations is not as good as the convergence of single code simulations, we want to define convergence criteria that will ensure convergence, or at least provide a criterion that allows to assess the convergence quality of a multi-code simulation.

Furthermore, we will investigate a generalization of the coupled LES-RANS approach. Currently, our approach requires that the interfaces are predefined inflow *or* outflow boundaries. In order to develop an integrated LES-RANS method that will surpass the current specifications for a gas turbine engine, we need to develop a framework that will allow to define the interface boundaries arbitrarily. This will establish a unified LES-RANS multi-code approach that will impact the fluid dynamics research community beyond gas turbine applications.

Bibliography

[1] K. Akselvoll and P. Moin. Large-eddy simulation of turbulent confined coannular jets. *Journal of Fluid Mechanics*, 315:387–411, 1996.

[2] A. G. Barker and J. F. Carrotte. Influence of compressor exit conditions on combustor annular diffusers, part 1: Diffuser performance. *Journal of Propulsion and Power*, 17(3):678–686, 2001.

[3] A. G. Barker and J. F. Carrotte. Influence of compressor exit conditions on combustor annular diffusers, part 2: Flow redistribution. *Journal of Propulsion and Power*, 17(3):687–694, 2001.

[4] C. U. Buice. *Experimental investigation of flow through an asymmetric plane diffuser*. PhD thesis, Stanford University, August 1997.

[5] P.-M. Cali and V. Couaillier. Conservative interfacing for overset grids. *AIAA paper*, (2000-1008), January 2000.

[6] P. A. Dellenback, D. E. Metzger, and G. P. Neitzel. Measurements in turbulent swirling flow through an aprupt axisymmetric expansion. *AIAA Journal*, 26(6):669–681, June 1988.

[7] M. Germano, U. Piomelli, P. Moin, and W. Cabot. A dynamic subgrid-scale eddy viscosity model. *Phys. Fluids*, A(3 (7)):1760–1765, July 1991.

[8] G. W. Hedstrom. Nonreflecting boundary conditions for nonlinear hyperbolic systems. *Journal of Computational Physics*, 30:222–237, 1979.

[9] J. S. Hesthaven. On absorbing boundary conditions for linearized euler equations by a perfectly matched layer. *Journal of Computational Physics*, 142:129–147, 1998.

[10] R. L. Higdon. Initial boundary value problem for linear hyperbolic systems. *SIAM Review*, 28:177–217, 1986.

[11] C. Hirsch. *Numerical Computation of Internal and External Flows*, volume 2. John Wiley & Sons, New York, 1990.

[12] F. Q. Hu. On absorbing boundary conditions for linearized euler equations by a perfectly matched layer. *Journal of Computational Physics*, 129:201–219, 1996.

[13] S. Kim, J. Schlüter, X. Wu, J. J. Alonso, and H. Pitsch. Integrated simulations for multi-component analysis of gas turbines: RANS boundary conditions. *AIAA Paper*, (AIAA 2004-3415), July 2004.

[14] A. Klein. Characteristics of combustor diffusers. *Prog. Aerospace Sci.*, 31:171–271, 1995.

[15] R. Klein. Semi-implicit extension of a godunov-type scheme based on low mach number asymptotics i: One dimensional flow. *Journal of Computational Physics*, 1995.

[16] H. O. Kreiss. Initial boundary value problem for hyperbolic systems. *Communications of Pure and Applied Mathematics*, 23:273–298, 1970.

[17] P. A. McMurtry, W. H. Jou, J. J. Riley, and R. W. Metcalfe. Direct numerical simulations of a reacting mixing layer with chemical heat release. *AIAA Journal*, 24:962–970, 1986.

[18] P. Moin and S. Apte. Large-eddy simulation of realistic gas turbine combustors. *AIAA Paper*, (AIAA 2004-0330), January 2004.

[19] P. Moin, K. Squires, W. Cabot, and S. Lee. A dynamic subgrid-scale model for compressible turbulence and scalar transport. *Phys. Fluids*, A 3(11):2746–2757, November 1991.

[20] C. Pierce and P. Moin. Large eddy simulation of a confined coaxial jet with swirl and heat release. *AIAA Paper*, (AIAA 98-2892), June 1998.

[21] C. D. Pierce and P. Moin. Method for generating equilibrium swirling inflow conditions. *AIAA Journal*, 36(7):1325–1327, 1998.

[22] T. J. Poinsot and S. K. Lele. Boundary conditions for direct simulations of compressible viscous reacting flows. *Journal of Computational Physics*, (101):104–129, 1992.

[23] R. Prosser. Improved boundary conditions for the direct numerical simulation of turbulent subsonic flows i: Inviscid flows. *submitted to Journal of Computational Physics*, 2004.

[24] R. Prosser and J. Schlüter. Toward improved boundary conditions for the DNS and LES of turbulent subsonic flows. *Proceedings of the 2004 CTR Summer Program*, 2004.

[25] J. Reuther, J. J. Alonso, J. C. Vassberg, A. Jameson, and L. Martinelli. An efficient multiblock method for aerodynamic analysis and design on distributed memory systems. *AIAA paper*, (97-1893), January 1997.

[26] D. H. Rudy and J. C. Strikwerda. A nonreflecting outflow boundary condition for subsonic Navier-Stokes calculations. *Journal of Computational Physics*, 36:55–70, 1980.

[27] D. H. Rudy and J. C. Strikwerda. Boundary conditions for subsonic compressible Navier-Stokes calculations. *Comput. Fluids.*, 9:327–338, 1981.

[28] J. U. Schlüter and H. Pitsch. Anti-aliasing filters for coupled LES-RANS computations. *submitted for publication to AIAA Journal*, 2003.

[29] J. U. Schlüter and H. Pitsch. Anti-aliasing for integrated LES-RANS computations. *AIAA paper*, (AIAA 2004-0258), January 2004.

[30] J. U. Schlüter, H. Pitsch, and P. Moin. Consistent boundary conditions for integrated LES/RANS simulations: LES outflow conditions. *AIAA paper*, (2002-3121), 2002. 32nd AIAA Fluid Dynamics Conference, June 24-27, St, Louis, MO.

[31] J. U. Schlüter, H. Pitsch, and P. Moin. Boundary conditions for LES in coupled simulations. *AIAA paper*, (AIAA-2003-0069), 2003.

[32] J. U. Schlüter, H. Pitsch, and P. Moin. Large eddy simulation inflow conditions for coupling with Reynolds-averaged flow solvers. *AIAA Journal*, 42(3):478–484, March 2004.

[33] J. U. Schlüter, H. Pitsch, and P. Moin. Outflow conditions for integrated large-eddy simulation/Reynolds-averaged Navier-Stokes simulations. *accepted for publication to AIAA Journal*, 2004.

[34] J. U. Schlüter, S. Shankaran, S. Kim, H. Pitsch, J. J. Alonso, and P. Moin. Towards multi-component analysis of gas turbines by CFD: integration of RANS and LES flow solvers. *ASME paper*, (ASME GT2003-38350), 2003. ASME Turbo Expo 2003, June 16-19, 2003, Atlanta, GA.

[35] J. U. Schlüter, X. Wu, S. Kim, J. J. Alonso, and H. Pitsch. A framework for coupling Reynolds-averaged with large eddy simulations for gas turbine applications. *Journal of Fluids Engineering*, submitted, 2004.

[36] J. U. Schlüter, X. Wu, S. Kim, J. J. Alonso, and H. Pitsch. Integrated RANS-LES computations of gas turbines: Compressor-diffuser. *AIAA Paper*, (AIAA 2004-0369), January 2004.

[37] J. U. Schlüter, X. Wu, J. J. Alonso S. Kim, and H. Pitsch. Integrated RANS-LES computations of turbomachinery components: Generic compressor/diffuser. *CTR Annual Research Briefs*, pages 357–368, 2003.

[38] J. C. Strikwerda. Initial boundary value problems for incompletely parabolic systems. *Communications on Pure and Applied Mathematics*, 30:797, 1977.

[39] K. W. Thompson. Time dependent boundary conditions for hyperbolic systems. *Journal of Computational Physics*, 68(1-24), 1987.

[40] R. Vichnevetsky. Invariance theorems concerning reflection at numerical boundaries. *Journal of Computational Physics*, 63(268-282), 1986.

[41] C. Wall, C. D. Pierce, and P. Moin. A semi-implicit method for resolution of acoustic waves in low mach number flows. *Journal of Computational Physics*, 181(2):545–563, 2002.

[42] Z. J. Wang. A fully conservative interface algorithm for overlapped grid. *Journal of Computational Physics*, 122:96–106, 1996.

[43] X. Wu, J. Schlüter, G. Iaccarino, F. Ham, H. Pitsch, and P. Moin. Computational study on the internal layer in a diffuser. *to be submitted to Physics of Fluids*, 2004.

Chapter 4

The Merrimac Streaming Supercomputer

Merrimac is a scientific supercomputer designed to deliver high performance in a cost effective manner scaling efficiently from a $20K 2 TFLOPS workstation configuration to a $20M 2 PFLOPS configuration. The high level of performance and economy are achieved by recasting conventional applications into stream programs and by leveraging commodity high-performance memory chips, a high-radix interconnection network, and using a standard CMOS processes. Cost effectiveness is achieved by re-examining the balance between compute rate, memory bandwidth and memory size. Current levels of VLSI integration afford the ability to fabricate several hundred FPUs in the die area of a typical high performance microprocessor in a 90nm CMOS process [9]. However, typical commodity processors can deliver only a few words of operands per cycle to the function units. Thus, global bandwidth is the bottle-neck preventing high performance, not arithmetic operations. Merrimac is designed from the ground up as a streaming architecture which recognizes that bandwidth is an expensive resource. It uses a hierarchy of register resources of increasing bandwidth including a *stream cache*, *stream register file* (SRF) and clustered execution units fed by *local register files* (LRFs) to simultaneously improve data parallelism while conserving bandwidth. The resultant architecture has a peak performance of 128 GFLOPS per node at 1 GHz in a 90nm process. The key technology that makes such high throughput possible is the stream oriented style of processing and programming.

4.1 Project Summary

The Merrimac project started in September 2001 with the goal of solving key technical problems associated with building and programming a streaming supercomputer and to demonstrate the feasibility of building such a machine in a manner which reduces the risk to the point where it will be attractive to the industry. On our third year, we are glad to report that we have made significant advances towards building a prototype machine. The progress over the last three years can be summarized as follows.

FY 2002 We sketched the architecture, developed a rudimentary programming system, demonstrated a few simple applications on a simulated machine, and identified key technical issues. Our goal of completing a quick feasibility test on the technology was met, though we did not expend significant effort to build infrastructure required for a more elaborate study.

FY 2003 During the second year, we focused on building a robust experimental infrastructure. We refined

the Brook language to make it both more expressive and easier to compile. We were delayed by about six months from our original goal by two false starts on developing a new compiler. In the end we decided to co-develop the compiler with Reservoir Labs. Reservoir Labs was already developing a stream compiler for Brook with sponsorship from DARPA IPTO in the Polymorphous Computer Architecture (PCA) program administered through AFRL Rome (contract number F30602-03-C-0033). We also addressed a variety of key technical issues on both the hardware and software side, and demonstrated more complex (3-D and irregular) applications and kernels.

FY 2004 During the current year our efforts have concentrated on moving towards building a prototype and continued application development. To this end, we have elaborated the details of the architecture and programming system to the point where we could successfully get an initial bid from the industry. In particular:

1. We refined the architecture and instruction set to the point where we were able to release a detailed 130 page architectural specification for Merrimac. Details may be found in Section 4.3 [5].
2. As a direct result, we had enough detail to get three candidate partners from the industry interested in designing the Merrimac Processor chip and potentially collaborating on building a prototype. We made contact with CPU Technologies, Silicon Logic Engineering and Cadence Design Systems, and received initial bids for the chip design. We also had positive discussions with Cray Systems on using their future interconnect for Merrimac. This interconnect is largely based on our initial interconnection network design for Merrimac.
3. Following initial discussions based on the architecture specification with Lawrence Livermore National Lab, LLNL has expressed interest in collaborating with us on the definition, design, and implementation of the micro-kernel operating and run-time systems for Merrimac.
4. We have also initiated talks with Lawrence Berkeley National Lab (LBNL) toward using their Unified Parallel C (UPC) compiler, and parallel programming expertise and experience in developing Merrimac's multi-node software system support.
5. In FY 2003, we began a research collaboration with Reservoir Labs to develop a stream compiler targeting Merrimac. During the current year we have been working to improve programming support by identifying modifications required to make the Reservoir compiler generate efficient code for Merrimac. We have also worked on the Low Level Compiler (LLC), made initial strides toward stream programming for conventional architectures and supporting legacy coded on Merrimac. Details may be found in Section 4.6.
6. Our efforts to improve multi-node performance of Merrimac on scientific applications lead to improvements to the fetch and add support first described in the 2002 report. During the current year we worked out details of multi-node cache coherency and did architectural studies the details of which may be found in our recent publication [2].
7. To improve our ability to perform architectural experiments we integrated a MIPS scalar processor model and multi-node support into our simulation infrastructure. Details may be found in Section 4.5.
8. During previous years we had identified reliability and fault detection as critical issues for a Peta-FLOP super computer with thousands of processors. During the current year we have

worked out additional details of reliability and redundancy in the processor's execution clusters and stream register file. Details are available in Section 4.4.3.

9. We completed our study of the force calculation for Gromacs, a molecular dynamics code for protein simulation, on Merrimac. We analyzed various implementations and showed that the use of conditional streams and indexable SRF resulted in significant performance improvement. We started a study of the WENO scheme in CDP. We completed the implementation but haven't obtained performance and schedule results yet.
10. Effort was spent on demonstrating that Graphics Processing Units (processor found on graphics cards) can be used for scientific computing. We developed a programming environment based on Brook to easily port scientific codes to GPUs without requiring specialized knowledge about graphics hardware and languages. An entire application, Gromacs, was ported to GPUs and performance analysis are now under way. We also benchmarked a finite-element calculation for solid mechanics and demonstrated a speed up by a factor greater than 10.

The remainder of this chapter is structured as follows. Section 4.2 provides an introduction to stream processing and the Merrimac architecture. This is followed by Section 4.3 which summarizes the architectural specification and refinements we have developed over the past year. This is in turn followed by Section 4.5 that explains the improvements we have done to our architectural simulation infrastructure. Section 4.6 elaborates our efforts in the area of programming support. The last section 4.7 presents the accomplishments of the Merrimac Applications group.

4.2 Overview of the Merrimac Architecture

Stream processing is characterized by the repeated application of algorithms to a sequence of similar data records. Application code and data are decomposed into a collection of *kernels* and *streams* respectively. A stream is a sequence of data records. Kernel functions are implicitly applied to each record in a stream. A kernel reads a set of input streams and transforms them into zero or more output streams. A kernel may access local variables, but it is not allowed to access arbitrary memory locations.

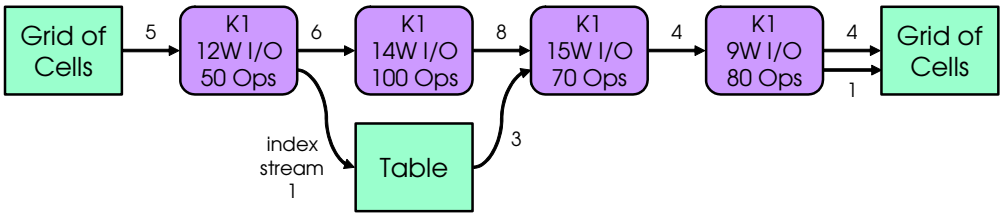


Figure 4.1: Example of a Stream Application

Figure 4.1 shows an example synthetic stream application for finite element modeling patterned after StreamFEM [3]. Each data record processed by this stream application is a set of 5-word grid cells. These are processed by a sequence of 4 kernels $K1...K4$ where each kernel operates on the output of its predecessor. Updated grid cell values generated by kernel $K4$ are finally written back to memory. Kernel $K2$ generates a set of indices for table lookups from main memory and these values become one of the input streams

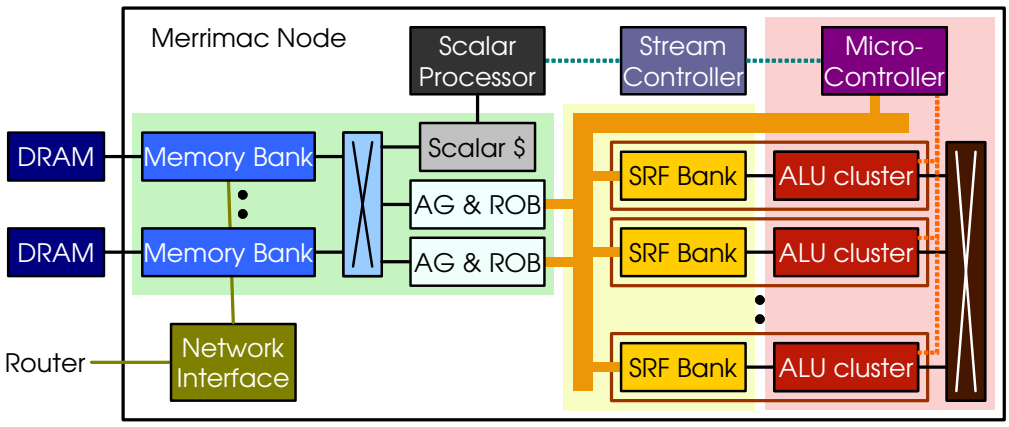


Figure 4.2: Organization of a Merrimac Node

for kernel $K3$. The example clearly illustrates both the parallelism inherent in stream processing and the opportunities to conserve bandwidth. The Merrimac stream processor capitalizes on the producer-consumer locality wherein the output of a kernel is fed as the input to the next kernel by sourcing these records from large on-chip register files. Spatial and temporal locality that are difficult to predict, for example the table lookups in Figure 4.1, are captured using a stream cache. The result is a large reduction in the fraction of data values that need to be fetched from main memory and an increase in arithmetic intensity.

4.2.1 Node Architecture

Figure 4.2 shows the high level architecture of a Merrimac node while Figure 4.3 shows the bandwidth hierarchy and additional details of the organization of a node. At the highest level, a Merrimac node consists of a scalar processor, a stream execution unit, a stream register file, a high performance streaming memory system, and a network interface.

The scalar processor executes the non-kernel scalar code as well as instructions to configure and control the stream processor. The configuration instructions appear to the scalar processor as writes to registers contained in the stream processor.

The stream execution unit consists of a set of 16 arithmetic clusters which represent Merrimac's computational engine. The clusters execute stream kernels expressed as *very long instruction word* (VLIW) microcode. Kernel execution in these units is done in a *single-instruction multiple-data* (SIMD) manner where all clusters operate based on the same wide micro-instruction word. The micro-instructions are sequenced under the control of a micro-controller unit (uC). The wide instruction word also includes an instruction field which the micro-controller itself executes. Each arithmetic cluster consists of four fused integer and floating-point multiply-add units (MULADDs), an iterative operation acceleration unit (ITER), a communication unit (COMM) which aids in inter-cluster communications, four units for accessing data in the stream register file (INOUT), and a jukebox unit (JB) that supports conditional stream accesses [7].

The goal of increasing arithmetic intensity entails taking advantage of the locality found in stream applications, and feeding the execution units from on-chip storage structures whenever possible. This purpose is served by the register hierarchy. Each function unit has its own local register file that captures short term

producer-consumer locality within kernels. The LRFs collectively provide enough bandwidth to keep all the functional units busy. The stream register file is a distributed storage structure, with one 8-KWord bank per arithmetic cluster. It acts as a staging area that decouples the main memory and the execution clusters. Since its total storage of 128 KWords is an order of magnitude larger than the LRFs, it could potentially capture additional long term producer-consumer locality not captured in the LRFs. The SRF also provides storage for persistent state and helps hide memory access latency. The function units do not have direct access to the SRF banks. Instead, their LRFs are fed and drained by a set of stream buffers, which perform rate matching and decoupling between the SRF and the LRFs.

The main memory system responds to stream load and store requests using a heavily banked DRAM memory back-end to transfer data into and out of the SRF. The memory system is distributed as 8 internal memory banks, where each bank in turn has access to 2 external DRAM chips, yielding a total of 16 DRAM interfaces per node. A stream cache, similarly organized as 8 banks, is incorporated into each memory bank. Each stream cache bank is 4-way associative with 4 KWords of capacity per way, yielding an aggregate capacity of 1 MB. The stream cache is primarily intended as a bandwidth amplifier. High throughput is achieved by using two *address generators* (AGs), which together are capable of issuing 8 addresses every cycle. The generated addresses may be scheduled to optimize DRAM throughput and the resultant data may be reordered by two *reorder buffers* (ROBs) paired with the AGs. The system uses a flat address space and it is possible to access the memory attached to remote nodes in a multi-node configuration. Access to local and remote memory is governed by a memory segmentation scheme that maps virtual address ranges across physical nodes and specifies cacheability and protection attributes for memory ranges. Virtual to physical address translation is handled in the AGs during address generation or in the scalar–stream interface. Remote addresses are identified and sent over the network. The network connects multiple Merrimac nodes into a global shared memory machine by forwarding remote memory requests made by each node to the appropriate home node. It also facilitates communication between scalar processors. The stream controller handles data flow between various modules on the chip, and is responsible for scheduling and controlling stream instructions.

The Merrimac chip which integrates all the subsystems described above is expected to have an area of roughly $12.5\text{mm} \times 12.5\text{mm}$ in a 90nm CMOS process. An approximate node-level floor plan appears in Figure 4.4.

4.2.2 System Architecture

The Merrimac streaming supercomputer is designed to scale from a single node to a 16 or 32 K node configuration while delivering tremendous bisection bandwidth. Figure 4.5 shows the internal organization of the super computer. At the lowest level, 16 Merrimac nodes are organized on one board. Each node has 2 GB of local memory organized as 16 Rambus XDR-DRAM chips connecting to 8 on-chip DRAM interfaces yielding 64 GB/s of peak memory bandwidth per node. The board level interconnect provides 12 GB/s bandwidth between any two nodes on the same board or a total board level bandwidth of 192 GB/s. A backplane integrates 512 nodes or 32 boards and provisions for 3 GB/s bandwidth between any two nodes or an aggregate bandwidth of 1.5 TB/s. The system level network connects up to 32 backplanes together and allows for 1.5 GB/s bandwidth between any two nodes yielding a system wide bisection bandwidth of 24 TB/s and a peak arithmetic rate of 2 PFLOPs.

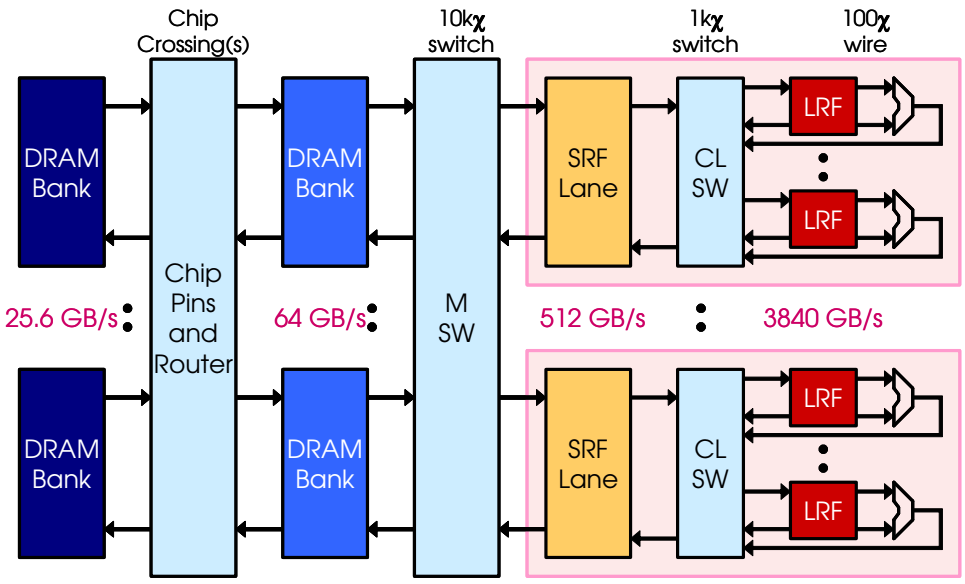


Figure 4.3: Bandwidth and Register File Hierarchy

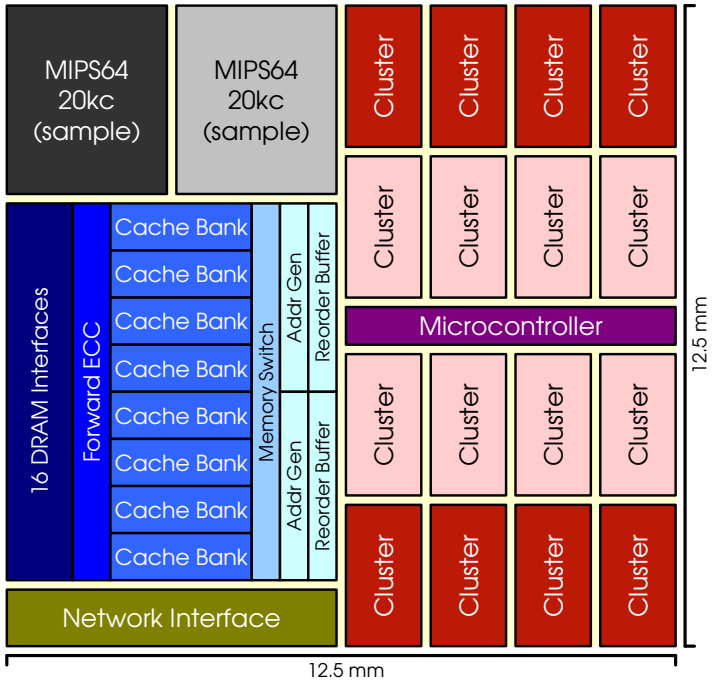


Figure 4.4: Merrimac Chip Floor Plan

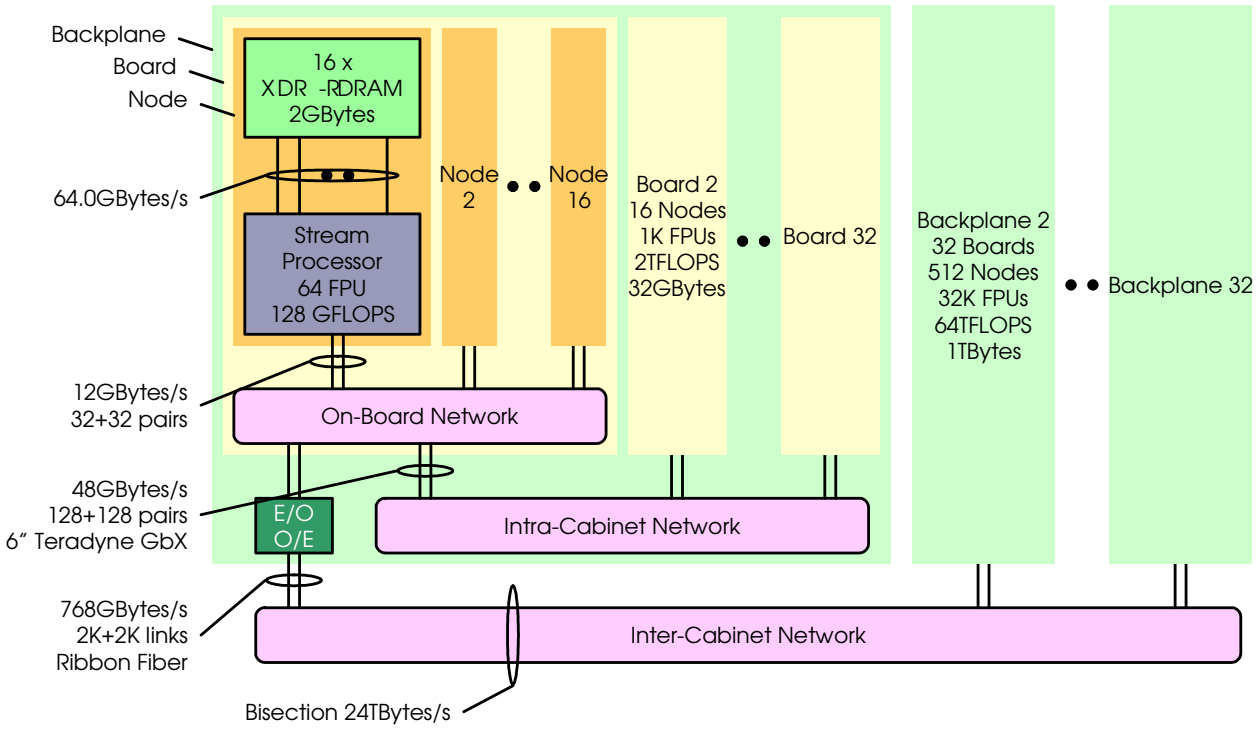


Figure 4.5: System Organization

4.2.3 Fault Detection and Reliability

The Merrimac processor is designed for both workstation configurations with a small number of nodes and for massively parallel supercomputers. While the workstation configuration may not require any special features for reliability and fault detection, a large configuration with thousands of nodes is likely to encounter multiple soft errors per week. Provisioning for reliable fault detection in the presence of soft errors leads to an increase in the processor's die area. The resultant cost increase must be balanced against the need for providing cost-effectiveness in situations that do not require extreme reliability. Therefore, Merrimac supports two operation modes: *productivity mode* for a workstation configuration, and *high-reliability mode* for large systems.

In both modes, all large storage arrays and cross-chip networks and buses are protected with an *error correcting code* (ECC) that has single error correction and double error detection capabilities. Fault-detection is also provided for control structures. This may be done using self-checking logic designs, or by replicating the scalar processor, stream controller, micro-controller, and the network and memory units. The extra circuitry increases the chip area without providing much benefit to the workstation configuration, but it is of crucial importance for a large configuration. Since the arithmetic clusters which are major area contributors are unaffected, we roughly estimate the area penalty at 25%.

In productivity mode, all 16 arithmetic clusters operate on independent data providing the full peak performance of 128 GFLOPS. However, in this mode cluster computations will not be checked. High-reliability mode requires sacrificing the peak performance by using pairs of arithmetic clusters to operate on the same data. Values will then be checked when they are written to the SRF or communicated on the inter-cluster data network, which are the only paths out of the clusters. High-reliability mode allows for a peak performance of 64 GFLOPS. While in high-reliability mode, the Merrimac processor will still provide its full memory bandwidth and SRF capacity. The SRF banks are also paired with the clusters and pairs act as a single SRF bank that is twice as wide.

4.2.4 Overall Operation

Applications for Merrimac may be written in Brook, StreamC and KernelC, or the extended version of C accepted by R-Stream (see Section 4.6.1. A compiler then performs two different decompositions. In the domain decomposition step, data streams are partitioned into pieces that are assigned to the local memory on individual nodes. The goal is to improve locality and bandwidth. The application code is partitioned into a scalar program and a set of kernels. The scalar code is compiled into a scalar processor binary and each of the stream kernels is compiled into microcode. The domain decomposed data and program binaries are loaded into node local memories and the scalar processors at each node commence execution.

Stream loads and stores in the scalar program initiate data transfer between global memory and a node's stream register file. Similarly, kernel load instructions copy kernels that were previously loaded in the stream register file into microcode memory. Kernel execute instructions in the scalar program cause the stream execution unit to start running a previously loaded kernel. Kernel execution is controlled by a micro-controller unit and proceeds in a SIMD manner with all clusters in the same node progressing lockstep. Data transfers between various levels of the on-chip memory hierarchy including the stream register file, stream buffers and local register files are controlled from the microcode. As kernel execution progresses, the scalar program continues to load more data into the stream register file and save output streams to main memory. The scalar processor also handles global synchronization primitives such as barriers. Multiprogramming is

supported by the use of segment registers which protect memory regions of individual programs from each other.

4.3 Merrimac Specification

Over the past year, we refined the architecture and instruction set to the point where we were able to release a detailed 130 page architectural specification for Merrimac [5]. The component of the architecture specification document that represents our main effort during the year is the Merrimac instruction set architecture (ISA). Previously, we were using an initial ISA that was a slightly modified version of the Imagine processor's ISA. Until now, the new features and capabilities of Merrimac were not dealt with in a systematic and rigorous fashion. The new ISA specification is a conglomeration of three separate ISAs and one requirements specification. They are:

1. The Kernel ISA specifies instructions that are executed while running a kernel on the compute clusters and microcontroller.
2. The Microcontroller ISA includes instructions that are executed by the microcontroller while it sequences the microinstructions that comprise a kernel including support for loops and handling communications between clusters. Since the microcontroller subinstruction is a field in the wider kernel microinstruction, the microcontroller ISA may be thought as a component of the kernel ISA.
3. The Stream ISA includes instructions that set stream related architectural registers, invoke kernels, and initiate and control memory operations.
4. A minimal set of requirements that the ISA of the scalar core must satisfy for easy and efficient interaction with the Merrimac stream unit were also stipulated.

Our focus has been on adding support for the modified local register file organization of Merrimac, accelerated iterative operation lookup capability, and indexed SRF access modes including more flexible conditional streams in the kernel ISA, defining Merrimac's exception model, generalizing support for kernel control in the micro-controller, and the scalar-stream interface.

4.3.1 Kernel ISA

Our work on the Kernel ISA has been focused on on adding support for Merrimac's improved local register file (LRF) organization, accelerated iterative operation lookup capability, indexed SRF access modes including more flexible conditional streams and defining an exception model

LRF Related Changes

The local register file of a MULADD unit contains 96 64-bit words. The LRF has 3 read ports, and 2 write ports. However, in order to strike a good balance between scheduling performance and the LRF physical dimensions, power, and speed, our studies indicated that the LRF should be composed of 3 individual 1 read and 1 write port register files along with a 3×3 switch on the read ports and a 2×3 switch on the write ports. Details may be found in Section 4.4.2. To facilitate this LRF organization, fields to configure the LRF read and write switches were added to MULADD kernel instructions.

Iterative Operations Support

Merrimac's clusters include a unit designed to accelerate the iterative computation of floating-point reciprocal and reciprocal square-root operations. High level details of the iterative unit were proposed in the 2003 report. During the past year, we have refined the algorithm and microarchitecture and specified kernel instructions executed by this unit. Details are presented in Section 4.4.1.

The operation of this ITER unit consists of looking up an 8-bit approximation of the function to be computed in a dedicated ROM and expanding it to a 27 bit approximation which can then be used to compute a full precision result in a single Newton-Raphson iteration. The ITER unit supports both an inverse and an inverse-square-root operation.

SRF Access Modes

Merrimac handles SRF accesses within the kernels through 4 INOUT units in every cluster. The INOUT units instruction set has been significantly modified and now includes support for in-lane block-indexed, in-lane word-indexed, and cross-lane word-indexed SRF accesses. Lane-wise and globally conditional accesses are also possible. Imagine only allows non-indexed block accesses and globally conditional streams. Another important addition is the ability to handle streams whose length is not a multiple of the number of clusters.

In-lane and Cross-lane Accesses

To perform in-lane block-indexed accesses, the INOUT units issue read and write instructions (`IN` and `OUT`). A read returns a 64-bit word from the SRF as well as a 1-bit valid flag onto two ports of the intra-cluster switch. A value of `1` in the valid flag indicates the data is valid (invalid data is returned when a read is requested past the end of the stream. Similarly, a write places the 64-bit word from the intra-cluster bus `data_bus` into the stream-buffer (SB).

Indexing is supported by the stream-buffers which expose an address FIFO accessible by INOUT instructions. A pair of word-address and word-count can be pushed onto the address FIFO. The word-count is decremented and the word-address is incremented on every read or write access. Once the word-count reaches zero, the next address/count pair is popped off the address FIFO. If no valid entries are left in the FIFO, subsequent read requests return invalid values indicated by a `0` in the valid flag, and write requests are ignored by the SB.

Cross-lane indexed access is very similar to in-lane accesses with the exception that only word-granularity read operations are permitted, and that the cluster ID holding the data to be accessed must also be provided.

Conditional Accesses

The INOUT unit support conditional accesses to the SBs in conjunction with the *jukebox* unit (JB). Conditional accesses allow each cluster to conditionally read or write a value from or to a stream independently of the other clusters. *Globally conditional* accesses are to a word from the next record of the global stream that is striped across the clusters, whereas *lane-wise conditional* accesses are to the next word within a specific lane. The inter-cluster coordination required for globally conditional accesses is handled by the JB unit and is described in the next section. As far as data access is concerned, the INOUT units handle both types of conditional access identically. Each SB has a FIFO of enable bits for determining whether an INOUT data

access should proceed or not. The bits are pushed onto the FIFO by a JB instruction and used by the INOUT conditional access instructions. When one of these instructions is issued a `1` at the head of the SB valid FIFO enables the access to proceed as an unconditional access, while a `0` suppresses the access, preserving the LSCR and SB states. The last word of a record pops a valid bit off the top of the valid FIFO.

Merrimac Floating-Point Support

During the last year we also worked on formalizing Merrimac's floating-point support and defining an exception model. While Merrimac employs IEEE 754 64-bit double-precision floating point representation, it does not strictly adhere to the standard. This section does not present the IEEE 754 standard but merely mentions the important points in which Merrimac adheres to the standard, and cases where Merrimac diverges from it.

Computations

Merrimac does not perform divisions, square roots, binary to/from decimal conversions, and remainders in hardware. Some hardware acceleration is provided, but the degree of IEEE 754 compliance is determined by the software library. In particular:

$+$, $-$, and $\times$ operations:– Computation is fully IEEE 754 compliant, exceptions and rounding are not fully compliant (see below).

Comparisons– Comparisons are fully IEEE 754 compliant. This including the behavior of reversing the result of equalities when comparing to `NaN`, even though this is unnecessary as Merrimac provides an IsNaN() function through the MULADD `FNAN` instruction.

$\frac{y}{x}$ – Division is implemented in software with hardware acceleration for reciprocals ($\frac{1}{x}$) through a special low-precision iterative operation acceleration unit (ITER). In this respect, the hardware does not comply with the standard. Please see precision, rounding, and exceptions below for more detail. In addition, while $\frac{1}{0}$ results in ∞, producing `NaN`s for $\frac{\infty}{\infty}$ and $\frac{0}{0}$ is left for software.

$\sqrt{x}$ – The hardware provides acceleration through a fast low-precision computation of $\frac{1}{\sqrt{x}}$. The implications are similar to those for division and are discussed below. The ITER unit will raise a flag and produce an `NaN` for $\frac{1}{\sqrt{-|x|}}$.

Precision

Merrimac supports double-precision IEEE 754 numbers including denormalized numbers, infinities, zeros, and `NaN`s. Extended double-precision is only used within the MULADD pipeline between the multiply and addition of a fused multiply-add.

Merrimac does not follow the standard regarding divide and square-root operations, as it does not provide dedicated hardware for these functions. Binary to decimal conversions and remainders are supported in software only, and are not guaranteed to be exactly rounded.

Rounding Modes

IEEE 754 specifies 4 rounding modes: round to nearest even, round to zero, round to $+\infty$, and round to $-\infty$. Moreover, the standard permits the program to change the rounding mode at run-time. Merrimac only provides rounding to nearest even, as this is the most commonly used mode and also because this mode offers superior numerical precision. Round to $\pm\infty$ modes are commonly used for interval arithmetic and are not supported.

Exceptions

Merrimac does not support precise exceptions due to the very large number of concurrent operations. The exception model is designed to allow for useful information to be passed from the hardware to the software system and allow the user to understand what exception was caused and the circumstances leading to it. An *exception control unit* records the first occurrence of an exception and signals the scalar processor an exception has occurred. This signal can be masked to allow the program to only be interrupted when a critical exception occurs. Either after an interrupt is generated, or when the program reaches a certain point in its execution, the exception unit's status can be queried and any raised exceptions handled. The information kept for every exception type varies, and includes enough details to locate the precise instruction causing the exception.

Compared to other processors, the type of exceptions generated by Merrimac is quite limited. Opcodes, bus numbers, and register numbers, etc., are generated by the compiler and are not checked by the hardware. At the price of reduced performance, a load-time scan of each kernel's microcode can ensure that only valid kernels are loaded into the SRF and the microcode store.

Table 4.1 lists the types of possible exceptions, the FUs affected by the exception, the hardware response and whether static microcode inspection can detect the exception. Further details for floating-point exceptions are provided below. Other exceptions are raised by the memory system in case a memory operation attempts to access an address outside its defined segment, or when an atomic floating-point memory operation faults.

Merrimac provides a mechanism to flag the first occurrence of 4 of the 5 IEEE 754 exceptions:

- **Invalid –** This flag is raised when an invalid result is generated producing a `NaN`. Table 4.2 lists the situations in which `NaN` is produced.

- **Divide by Zero (DVBZ) –** raised when $\frac{1}{0}$ is executed on the ITER unit (`SFINV(0)`). The result returned by the ITER unit in this case is always a $+\infty$.

- **Overflow –** raised when an addition, subtraction, multiplication, or fused multiply-add results in a number that is larger than the maximum representable using double-precision. In addition to raising the overflow flag, the result returned is $+\infty$.

- **Underflow –** raised when a computation result is smaller than the minimum representable number using denormalized form of double-precision. The result returned is a ± 0. Merrimac only raises the underflow exception flag when a denormalized number underflows, not when gradual underflow begins.

The *inexact* exception type of IEEE 754 is not supported. Similar to the case of the less common rounding modes, it is most often used for interval arithmetic.

Also, the Merrimac hardware does not pass any parameters to the trap handler other than the exception location in the instruction stream (PC, cluster ID, and FU number).

IEEE 754 does not call for precise exceptions. Instead, it specifies that each instruction with an exception raise a flag, and suggests that software should be able to trap on it. Merrimac provides a limited form of this specification. Since potentially up to 144 different floating point exceptions can be raised on every cycle of kernel execution, and a kernel typically performs tens-of-thousands of floating point operations yet appears as a single instruction to the controlling scalar processor, it is unreasonable to allow software traps on every FP operation. Instead, traps are allowed at the granularity of an entire kernel. Every instruction that produces a value requiring an exception, signals the exception control unit of Merrimac. If that particular exception had not yet occurred during the current invocation of the kernel, the PC, cluster ID, and functional-unit ID are recorded. At some point, no later than completion of the kernel, the exception unit will raise the appropriate exception flag that can be trapped by the scalar control unit.

4.3.2 Micro-controller ISA

The microcontroller is responsible for controlling the execution of the microprograms executed on the arithmetic clusters. The same micro-instruction is issued to all the clusters simultaneously by the microcontroller. Hence, the execution of the clusters occurs in lockstep, in SIMD fashion. During microprogram execution the microcontroller also responds to certain fields in the microcode instruction word of the currently executing microprogram. The functions of the microcontroller are:

1. Sequence the micro-instructions that comprise a kernel.
2. Handle communications between the compute clusters and the stream controller (SC).
3. Squash state changing operations (register writes and INOUT operations) to correctly prime and drain software pipelined loops.

We extended and modified the micro-controller instruction set to allow a more general approach to performing its functions. The micro-controller now contains an arithmetic and logic unit (ALU) to allow for generalized control flow and interaction with the stream controller and scalar unit such as double-buffering streams and kernels, and communicating scalar values to and from kernels.

The main components of the microcontroller are (Figure 4.6):

ALU: The microcontroller ALU performs integer arithmetic and comparison operations to support loops and sequencing.

uCR: The microcontroller register file holds 64 64-bit values that can be used for scalar variables within microcode loops, variables that persist across multiple kernels, data that needs to be communicated to the clusters or the stream controller, etc... This register file is shared between the microcontroller and the stream controller. The microcontroller owns 2 read ports and 1 write port, and the SC owns 1 read and 1 write port.

SCR: The microcontroller holds 12 *stream control registers* (SCRs) that describe current cluster streams. Stream descriptions are copied into the SCRs from the SDR register file when kernel execution is started by the RUN_KRNL SC instruction. The content of the SCRs is used to provide the base address for indexing in kernels. The contents may be read by the RSCR instruction, but is otherwise not accessible via the microcontroller instruction set.

COMM: The microcontroller COMM unit allows reading and writing 64-bit values to/from the inter-cluster data switch. It outputs a value from a local register file on to the data switch, and selects one of 17 data sources (16 clusters and the microcontroller itself) from the inter-cluster data switch and outputs the chosen 64-bit value on its output bus.

COMMC: The microcontroller COMMC unit allows reading 16 1-bit condition codes sent by the 16 clusters from the inter-cluster conditions network.

PC-unit: This unit handles jump and loop instructions and increments the micro-controller's program counter.

Fetch, Decode, SWP Logic: This component fetches and decodes microinstructions and modifies instructions to squash any operations causing state change while priming or draining software pipelined loops.

Microcode store: SRAM for storing the microinstructions.

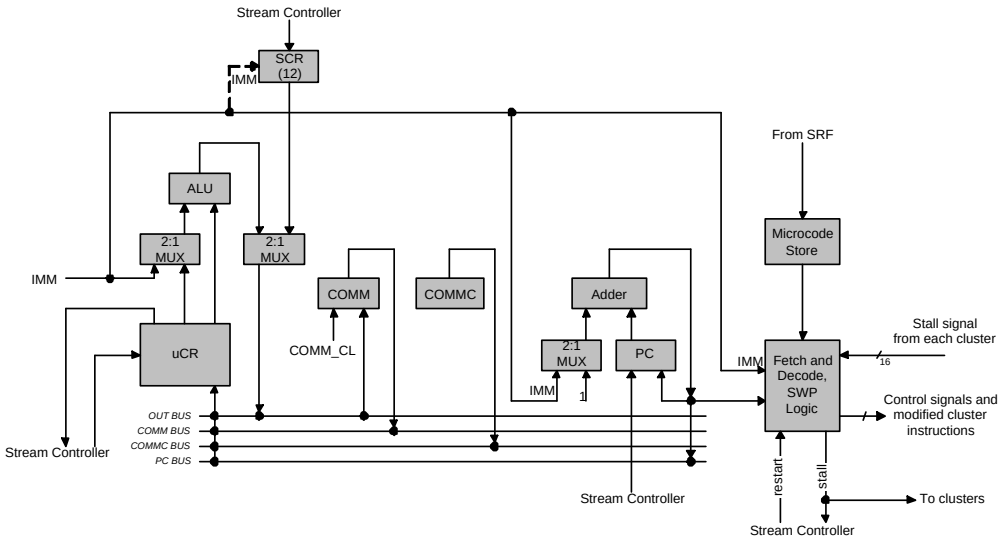


Figure 4.6: Microcontroller Architecture

Communication with Other Units

The microcontroller and stream-controller exchange data via the shared *micro-controller registers* (uCRs) as both units own read and write ports to this register file. In addition, the `RSCR` instruction may be used to read the values of SCRs set by the SC during kernel initialization. The micro-controller can pause execution (conditionally or unconditionally) with the `PAUSEC` or `PAUSE` instructions which help to synchronize with the SC. Once the microcontroller is paused, the SC can safely update uCRs or the PC and then restart the microcontroller.

The inter-cluster data switch enables the microcontroller to broadcast a 64-bit word to the clusters every cycle. In addition a word may be read from the inter-cluster switch. The source of the word to be read refers to one of 17 possible data sources including 16 clusters and the microcontroller. The selected value will then

be placed on the COMM unit's output bus. In addition, the value on the inter-cluster conditions network can be written into the lower 16 bits of a uCR.

Instruction Sequencing

The chief function of the microcontroller is to sequence the VLIW instructions to the clusters and handle SWP loops and other control flow. For control flow, the microcontroller has an integer-only ALU that can execute a variety of compare operations for testing branch (and pause) conditions (`EQ`, `NEQ`, `LEQ`, `GEQ`) as well as arithmetic operations to manipulate counters (`ADDI`, `ADDR`, `SUBR`, `SHFTR`, and `POPC`).

Since the PC needs to be updated every cycle, a special functional unit (*PC-unit*) is in charge of this task. Unless the micro-controller is stalled (`STALL` is true), the PC-unit will implicitly add a value to the PC on each cycle. The value added is 1, unless one of the jump or loop instructions is issued. Only relative jumps are supported in which case the offset is supplied by the immediate constant field of the instruction. The PC-unit can also overwrite the current value of the PC with a value set by the SC.

In addition to simply sequencing instructions, the micro-controller also provides hardware support for software pipelined loops in the kernels. This is done with the use of two special instructions: `LOOPB` and `LSTART`. The SWP unit contains a state-machine that governs the priming and draining of software pipelined loops. The `LSTART` opcode is issued just before a loop starts to reset the SWP unit and to set the maximum number of stages in the software pipeline. The `LOOPB` opcode is used to mark the backward jump and the loop continuation test. `LOOPB` will jump back to the beginning of the loop until the SWP is fully drained (even after the loop continuation test fails for the first time). Every execution of `LOOPB` updates the stage counter of the SWP unit, incrementing it while looping, and decrementing it while draining. The stage counter is used along with the stage field associated with each write port's micro-code field to suppress register writes that should not be executed while priming or draining the SWP. JB and INOUT unit operation is also similarly predicated.

Looping Behavior

The microcontroller ISA supports six loop constructs:

1. Loop until a predetermined iteration count (*loop-count*)
   A uCR is set to the number of desired iterations either by the SC while the micro-controller is stalled, or by the result of a micro-controller operation. It then decremented by an explicit micro-controller instruction at the end of every loop iteration. The following example illustrates such a loop assuming the counter register is `uCR[1]` and the software pipeline has 4 stages.

```
uCR[2] <-- EQ    uCR[1], 0x0
...
// Wait EQ latency cycles or do equivilant cycles of work
...
JMPF  uCR[2], $end_of_loop

LSTART 0x4    // Start an SWP with 4 stages
start_of_loop:
uCR[1] <-- ADDI  uCR[1], -1
```

```
...
// Body: wait/work for at least (LSTART + ADDI) latency cycles
...
uCR[2] <-- GEQ   uCR[1], 0x1
...
// Wait GEQ latency cycles or do equivilant cycles of work
...
LOOPB uCR[2], $start_of_loop
end_of_loop:
```

2. Loop until a stream is fully read (*loop-stream*)
   This is similar to looping for a predetermined number of iterations, except that the count register (`uCR[1]` for example) is set to the length of a particular stream, and the the number of reads performed on the chosen stream during an iteration are subtracted from the count. In the case of normal streams, the number of reads is predetermined by the kernel scheduler. In the case of conditional streams the number of reads is dynamically computed using a `POPC` instruction on the condition values collected over the inter-cluster conditions switch. An example follows.

```
uCR[2] <-- EQ    uCR[1], 0x0
...
// Wait EQ latency cycles or do equivilant cycles of work
...
JMPF  uCR[2], $end_of_loop
LSTART 0x4    // Start an SWP with 4 stages
start_of_loop:
...
// Body: wait/work for at least LSTART latency cycles
...
uCR[2] <-- cluster conditions (COMMC of JB)
uCR[3] <-- POPC uCR[2]
uCR[1] <-- SUBR uCR[1], uCR[3]
...
// More body: (minimum (SUBR + conditions comm + POPC) latency)
uCR[2] <-- GEQ    uCR[1], 0x1
...
// Wait GEQ latency cycles or do equivilant cycles of work
...
LOOPB uCR[2], $start_of_loop
end_of_loop:
```

3. Loop while any cluster's condition is true (*loop-while-any*)
   This form of loop is similar to the counter based loop, but the condition is dependent on conditionals in the clusters.

```
uCR[1] <-- cluster conditions (COMMC of JB)
```

```
uCR[2] <-- NEQ    uCR[1], 0x0
...
// Wait (NEQ + conditions comm) latency cycles or do equivilant cycles of work
...
JMPF  uCR[2], $end_of_loop
LSTART 0x4    // Start an SWP with 4 stages
start_of_loop:

...
// Body: wait/work for at least LSTART latency cycles
...
uCR[1] <-- cluster conditions (COMMC of JB)
...
// More body: (minimum conditions comm latency)
uCR[2] <-- NEQ    uCR[1], 0x0
...
// Wait GEQ latency cycles or do equivilant cycles of work
...
LOOPB uCR[2], $start_of_loop
end_of_loop:
```

The last three loop types are similar except for the loop condition.

4. Loop while the conditions in all clusters are true (loop-while-all)
5. Loop until any cluster's condition is true (loop-until-any)
6. Loop until all cluster's conditions are true (loop-until-all)

Double Buffering Long Streams

In some situations the high-level stream compiler might decide to double buffer the inputs or outputs of a kernel through the SRF instead of strip-mining multiple kernels together. This type of operation is supported through the cooperative efforts of the microcontroller, the stream controller and the scalar processor.

The scalar processor executes a loop until the stream computation is complete. Within this loop, the stream controller initiates double buffered data transfers. It sets the buffer length in a uCR register at which point the microcontroller executes a loop over the buffered data, say a counted loop. When the data has been consumed, the microcontroller stalls using a `PAUSEC` instruction at which point the stream controller restarts the microcontroller after setting the uCR count register to the number of data records it buffered up while the microcontroller was executing the kernel. An example for a counted loop follows, but other loops including double-buffered conditional output streams follow a similar form:

```
while(...) // This line runs on the scalar processor
{
  if (FIRST_BUFFER) {
    STREAM_CONTROLLER sets uCR[4] to 0x1 indicating the microcontoller
```

```
                                              should continue running the kernel
    STREAM_CONTROLLER sets uCR[1] to length of first buffer
    STREAM_CONTORLLER starts kernel in microcontroller
  }

  // The following code is part of the kernel in the
  // microcontroller.
  uCR[2] <-- EQ    uCR[1], 0x0
  ...
  // Wait EQ latency cycles or do equivilant cycles of work
  ...
  JMPF  uCR[2], $end_of_loop
  LSTART 0x4    // Start an SWP with 4 stages
  start_of_loop:
  uCR[1] <-- ADDI  uCR[1], -1
  ...
  // Body: wait/work for at least LSTART latency cycles
  ...
  uCR[2] <-- GEQ    uCR[1], 0x1
  ...
  // wait GEQ latency cycles or do equivilant cycles of work
  ...
  PAUSEC uCR[2]   // Stall until the SC restarts the microcontroller
  // next 3 lines are run in the SC
  STREAM_CONTROLLER sets uCR[4] to 0x0 if this was the last buffer or
                                leaves it at 0x1 if more buffers remain
  STREAM_CONTROLLER sets uCR[1] to length of the next buffer
  STREAM_CONTORLLER restarts kernel in microcontroller
  // final line of the kernel loop in the microcontroller
  LOOPB uCR[4], $start_of_loop
  end_of_loop:
}
```

A similar sequence can be used to double-buffer instructions in the microcode store, but this should only be done as a last resort.

4.3.3 Stream ISA

Stream instructions are embedded within the scalar instruction stream and are used to control the stream unit, issue stream memory operations, initiate and control kernel execution, and manipulate stream unit state. The stream instructions are processed by the stream controller to allow their execution to be decoupled from scalar execution. The stream controller (SC) accepts instructions from the scalar unit interface and autonomously schedules them to the appropriate streaming resource. Along with the instruction, the scalar unit also sends encoded dependency information of the instruction on preceding stream instructions. The SC uses a 64-entry score-board to dynamically schedule the out-of-order execution of stream instruc-

tions on 5 resources it controls: 2 address generators (AGs) for running stream memory instructions, the microcontroller which sequences kernels, a cache controller, and the SC itself.

Stream Controller Control Instructions

To allow efficient control over stream instruction execution Merrimac provides a set of instructions affecting the stream-controller.

The ADD_DEP_S and ADD_DEP_C instructions are used to communicate dependencies for stream instructions in the stream controller. ADD_DEP_S specifies the start dependencies for the next stream instruction (unless it is an ADD_DEP_C). These dependencies indicate to the SC that the next instruction should not begin execution until all the instructions specified by the ADD_DEP_S already started. If an ADD_DEP_S is not issued before a regular stream instruction, the stream instruction is not dependent on any preceding instruction starting execution. ADD_DEP_C performs a similar function but for dependencies on prior instructions completing execution. Each stream instruction must be preceded by either a ADD_DEP_S or a ADD_DEP_C or both.

KILL is issued to remove instructions that have not yet started to execute from the SC.

STRBAR is used to control the sequence of instructions issued by the SC. The SC will not issue any instructions that follow a STRBAR before all its dependencies are cleared (dependencies issued by ADD_DEP_S and ADD_DEP_C).

Stream Memory Operations

Merrimac can issue strided memory reads and writes, indexed reads and writes, and various flavors of these operations which include one of the following operations: integer increment, integer add, and floating-point addition. The memory operations are dynamically scheduled to one of the 2 AG resources.

Before a memory operation can proceed, several hardware control registers must be set up. These are a single *memory address register* (MAR) used to map a logical stream to a location in memory, and up to 3 SDRs: an SDR specifying the source of a write or a destination of a read, an SDR specifying the stream containing the indices of a gather or scatter operation, and a stream containing the data values in case the memory instruction includes an atomic memory operation.

In addition to these set instructions, the memory instruction should also depend on any instructions generating streams that are required for the memory operation, or instructions that are accessing areas of the SRF that are to be overwritten by the memory operation. It is important to remember that Merrimac does not guarantee any implicit memory ordering between different stream memory operations and between multiple nodes in a system. The compiler must use the dependencies between stream instructions or barriers to ensure all hazards are removed.

Memory operations may raise exception flags under two conditions: a) an address is requested that is outside the specified segment (segment violation) or b) a floating-point exception occurs. When exceptions occur, the stream element number of the first address that caused the exception is stored in the AG's exception register for the appropriate exception type. In the case of an FP exception, the ID of the FU that faulted (one of the 8 FUs in the atomic-memory-operation units) is also recorded. Once the memory operation finishes execution, software can check the exception register belonging to the AG used for the memory operation. The AG number can be determined by examining two AGID registers that contain the score-board entry number of the last memory instruction executed on each AG.

Merrimac also provides a mechanism to perform gang invalidation and flushing of the stream cache. `INVALIDATE` can be used to perform gang invalidation in the stream cache of all lines that match one of the valid bits, and `FLUSH` can be used to flush lines out of the cache based on the valid bits as well. `INVALIDATE` and `FLUSH` use the cache controller resource.

Kernel Control Operations

Merrimac's stream ISA provides several instructions for controlling kernel execution. The kernel control operations are executed on the microcontroller resource, and only one of the following operations can execute at any given time:

1. `LD_UCODE` transfers data corresponding to the microcode of a kernel from the SRF into the microcode store. The microcode must first be read into the SRF from memory. This instruction cannot run while a kernel is executing and must wait for the microcontroller to pause.
2. `RUN_KRNL` sets up the state required for running a kernel (including SCR state, PC, and control signals) and then releases the microcontroller from its paused state. This instruction is used when a kernel is first run, or if the SC overwrites SCR values while a kernel is paused. This instruction cannot proceed if a `LD_UCODE` is in progress.
3. `RESUME_KRNL` can be used to resume a paused kernel after the SC manipulates the uCRs or the PC, but does not require changes to the SCRs. This instruction cannot proceed if a `LD_UCODE` is in progress.
4. `SETPC` is used by the SC to set the microcontroller PC while the microcontroller is paused. This instruction is used before first starting a kernel, for performing absolute jumps (the microcontroller is only capable of executing relative jumps), and for potentially "double-buffering" a kernels micro-code into the micro-code store.

Scalar Stream Instruction Encoding

This subsection deals with issuing stream instructions from the scalar unit to the stream unit. We will use the MIPS coprocessor interface (COP2) as an example for a mechanism to interface with the SC and to transfer instructions and information. We will assume the following capabilities by the scalar processor:

- We can send a 26-bit immediate value to the stream unit. This field will be used to issue instructions (2 bits will be used as an opcode) and pass data as described below.
- We can directly transfer register values between the 32 GPRs and a set of 32 logical registers in the stream unit as detailed in Table 4.3.
- All COP2 instructions are issued in order by the scalar processor.

The stream/scalar interface is based on the MIPS coprocessor interface (COP2). In particular we assume that the scalar processor can transfer value between its general purpose registers and the stream unit. The scalar processor can issue a single instruction to write a general purpose register value or an immediate to the stream unit. Several of these instructions issued in sequence are used to fill an instruction buffer in the stream unit and then issue the stream instruction (the instructions described above in this subsection.

In order to read values from the stream unit, the scalar processor can issue an instruction to fill one of its general purpose registers. The value can be read from one of several sources in the stream unit and is referred to by a set of logical register numbers detailed in Table 4.3.

The ISA allows for scalar/stream synchronization by way of polling. The scalar processor can read the scoreboard status register and wait until a specific bit which corresponds to the stream operation of interest is cleared.

4.4 Architectural Enhancements

4.4.1 Iterative Unit

The iterative unit is designed to accelerate the iterative computation of floating-point reciprocal and reciprocal square-root operations. High level details of the iterative unit were proposed in the 2003 report. During the past year, we have refined the algorithm and microarchitecture and specified kernel instructions executed by this unit. We have also shown that kernel run-time for divide limited applications like StreamMD, is cut in half by improving the sustained rate at which divide computations may be issued and by reducing the pressure on MULADD units.

The iterative function acceleration unit is a fully pipelined unit designed to accelerate the iterative computation of floating-point reciprocal and reciprocal square-root. The operation of the ITER unit consists of looking up an 8-bit approximation of the function to be computed in a dedicated ROM and expanding it to a 27 bit approximation which can then be used to compute a full precision result in a single Newton-Raphson iteration performed on the regular MULADD units. The internal structure of the unit is shown in Figure 4.7.

The equation solved by the significant portion of the datapath is listed below:

$X_0 \approx \frac{1}{b}$
$Y_0 \approx \frac{1}{\sqrt{b}}$
$d = (1 - bX_0)$
$\frac{1}{b} = X_0 \left(1 + d + d^2 + d^3\right)$
$\frac{1}{\sqrt{b}} = Y_0 \left(1 + \frac{1}{2}d + \frac{3}{8}d^2 + \frac{5}{16}d^3\right)$

The lookup table uses the 8 most significant bits of the fraction (not the implicit leading 1). The table provides approximations of both the inverse and inverse square root with 8 bits of precision. The inverse square root is only used in the final multiplication if the function is inverse square root.

Because X_0 is an approximation of the inverse of b, the first 7 bits of their products are guaranteed to be 0, which reduces the size of the multiplier. The squaring and cubing units are much smaller than full multipliers because of the fact that the same two numbers are multiplied by each other. Since we are multiplying fractions, the least significant bits are not very important. Of course they could potentially all be ones and generate a carry that would cause an additional bit, but we can tolerate up to 0.5 bits of error. We will use that property to cut off whole columns from the partial products that we have to calculate in the squaring and cubing units.

Depending on whether the operation is inverse or inverse square root, the results are multiplied by fractions of powers of 2 (shifts and additional partial products). In the case of an inverse square root operation, there are 6 partial products to be added by the adder just before the final multiplier.

The final multiplier multiplies the result of the addition with an 8 bit number from the lookup table. The 27 bit precise result will be padded with zeroes and transferred to the register file of a MULADD unit.

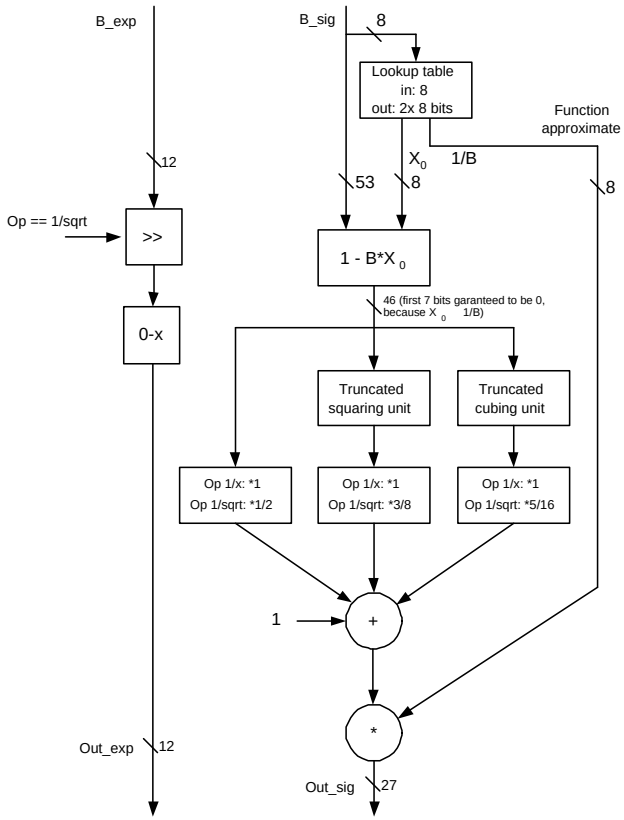


Figure 4.7: Accelerator Unit for Iterative Operations (ITER)

4.4.2 LRF Organization

In the past year we re-examined the organization of the local register files of the MULADD functional units. Our new LRF is more flexible than before while occupying a smaller die area.

The original LRF organization was 3 separate 64-entry 1-read/1-write ported register files, each connected to one of the 3 input ports of the MULADD unit. This organization worked well on the Imagine processor for media applications and also in initial studies on Merrimac. However, on closer examination, we discovered that the rigid connectivity caused excessive register pressure and a load imbalance in the register allocation of the static VLIW kernel scheduler. The fact that an input to a specific port of the MULADD must be in the register file connected to that port, and that the each individual register file has only a single output port, necessitates data replication when the same values are used in different operations. In addition, one of the register files was only used for select and fused multiply-add operations leading to unbalanced allocation. Ameliorating this unnecessary pressure allowed us to reduce the number of registers per functional unit while maintaining performance.

Our first attempt at a new organization was a 5-ported register file with 64 entries capable of supplying 3 operands per cycle to the MULADD unit and receiving two values from the intra-cluster switch. While this organization allows very flexible and balanced register allocation, the multiple ports require large area

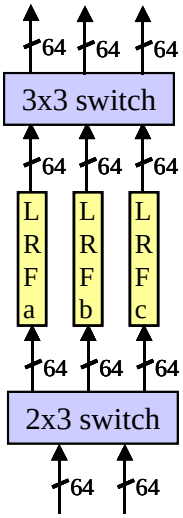


Figure 4.8: Organization of a MULADD LRF

and consume significantly more power than single ported structures, as summarized in Table 4.4.2. To keep the area reasonable we were forced to reduce the number of total registers available to the MULADD unit.

After analyzing the results of the two organizations we realized that we could increase the connectivity, and thus performance, without resorting to a large number of ports in the register file. We retained the 3 separate 1-read/1-write ported register files and introduced a 3×3 switch on the read side and a 2×3 switch on the write side. This *switched* organization emulates the true 5-ported case in most cases, requiring less area and power. The more flexible connectivity allows for better scheduling and balanced register allocation. The more efficient use of the available registers allows us to reduce the total number of registers required to maintain a given performance level leading to area and power savings compared to the original organization.

4.4.3 Enhancements to the Stream Register File

The stream register file (SRF) microarchitecture for the Merrimac processor was specified in detail. The main improvements over the generic indexed SRF presented previously are described below.

Unified control and address generation for indexed and sequential stream accesses

A single, unified set of address buffers are used for controlling both indexed and sequential streams instead of the separate addressing mechanisms used for the two different stream types previously. This enables more efficient access to streams that have hybrid indexed and sequential characteristics, such as those that are composed of several sequential data segments, but the segments are not located contiguously in the SRF. This arrangement also enables more efficient conditional stream access as described below.

Support for read/write (scratchpad) data

Prior streaming architectures provided separate on-chip memories for scratchpad data that are read and written using explicit indices during a single kernel's execution. The key distinction compared to an indexed SRF is hardware support for identifying concurrent reads and writes to the same address so that the latest data is returned as a result of the read (typically, data in an SRF is read only or write only during a kernel's execution and therefore does not lead to this problem).

We have added hardware support for read/write data to the indexed SRF of the merrimac architecture so that scratchpad data can also be stored in the SRF. We were thus able to eliminate dedicated scratchpad storage. The bandwidth available for read/write data is approximately half of that for read only or write only data in order to keep the hardware overheads low, but is comparable to the scratchpad bandwidth available in prior architectures. Storing both stream and scratchpad data in the same memory (i.e. the SRF) provides two important advantages:

- Eliminates the overhead of having to transfer data from the SRF to the scratchpad memory before performing read/write accesses.
- Eliminates allocation inefficiencies that arise in the statically partitioned scheme where separate memories are used for stream and scratchpad data

The drawback of storing scratchpad data in the SRF is increased access latency. Access latency of a small, dedicated scratchpad memory is approximately 2 to 3 cycles, compared to approximately 6 to 8 cycles for the average indexed SRF access. However, our studies have shown that software pipelined application loops whose bodies are 25 cycles or longer (which include all currently available Merrimac application loops that require read/write scratchpad accesses) see no slowdown due to increased scratchpad access latencies up to 10 cycles (we did not study latencies over 10 cycles as this is a reasonable upper bound for the expected access latency).

Efficient conditional stream accesses

Conditional streams are accessed sequentially but consumed (or generated) based on conditions that are dynamically computed by the compute clusters. In a conventional SRF, sequential streams must be accessed in lock-step by all clusters at once, requiring intermediate storage in scratchpad memory in order to accommodate the variations due to the dynamic condition codes. The unified sequential and indexed stream control of the Merrimac SRF enables conditional accesses to be supported directly from the SRF, eliminating the intermediate storage step.

Support for high-reliability mode

In high-reliability mode, clusters i (where i = 0, 2, 4, 6, ..., 14) and (i + 1) perform the same computation and compare results to ensure correctness. Result comparison is done for all data that is to be written to the SRF from the clusters. However, data in the SRF banks i and (i + 1) are not replicated in order to maintain high utilization of the expensive memory resources. Therefore, during execution in this mode, a single SRF bank must provide the same input data to a pair of clusters, and on SRF writes, the results of the two clusters must be compared and if a match is found, the result should be stored to a single SRF bank. If a mismatch is found, execution is rolled back to the previous checkpoint and re-executed from there. The high-level SRF

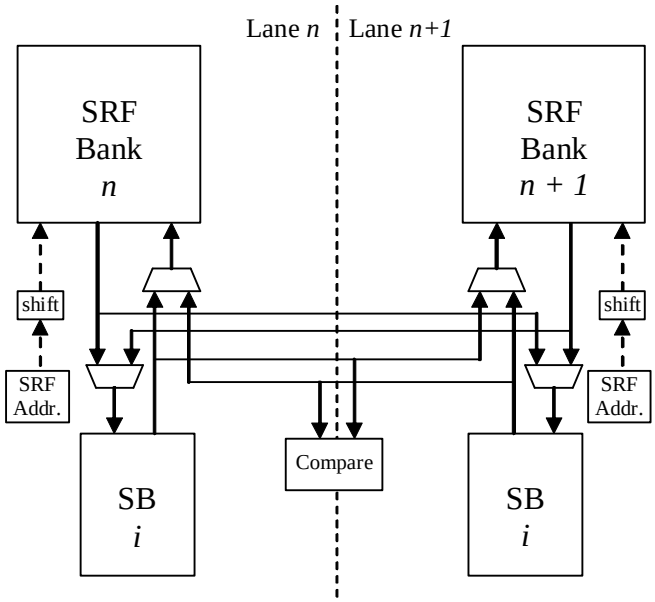


Figure 4.9: SRF support for high-reliability mode

modifications necessary to provide the proper inputs and outputs and to perform the correctness checks for high reliability mode are shown in Figure 4.4.3.

In addition, the memory array of the SRF and all busses providing SRF inputs and outputs are protected with ECC that enables single-bit error correction and two-bit error detection.

4.4.4 Cache Coherency and Consistency

The Merrimac memory system includes a stream cache that acts as a bandwidth multiplier for streams with reuse. Given Merrimac's flat address space, caching data located on remote nodes is attractive if they have spatial or temporal locality. Conventional cache coherency mechanisms like snooping or directory based models don't scale well and are expensive to implement. In prior years, we have assumed that multi-node configurations will be supported down the line, but did not fully develop the underlying cache coherency, consistency and scalability issues which have been studied extensively in the current year. As a result of our studies, we have developed a segment based cache coherency model for Merrimac to relieve the burden of word granularity protocols by exploiting the properties of stream accesses. Additional coherency issues for integrating scalar caches are also addressed. In addition we have also studied the coherency requirements that a scalar processor (host) would need to satisfy so that it can operate synergistically with the stream unit. Details are presented in the next two sections.

Stream Cache Coherency and Consistency

The use of a compiler-controlled stream cache and relaxed consistency allows Merrimac to be scaled to a large numbers of nodes. The stream cache supports explicit flush and invalidate instructions at a stream granularity. This is implemented via 4 valid bits in each stream cache line. The line is valid if any of the

valid bits are set. In each memory address register (MAR) there is an entry specifying whether the stream is memory-uncacheable or network-uncacheable (or both/neither), and another entry specifying which stream cache valid bit to use. If the memory-uncacheable flag is set, then any data whose address is in the node's local memory will not be cached. If the network-uncacheable flag is set, then any data whose address is in a remote node (i.e. over the network) will not be cached. The stream cache sits in front of the local memory and the network interface and satisfies requests from the local stream units, the local scalar processor, and remote nodes (via the network interface). The source of the memory requests is irrelevant to deciding stream-uncacheable properties.

An invalidate operation on a cached stream is cheap as it simply sets the associated valid bit line to zero, while a flush is expensive (as is usually the case) since the cache must be traversed to write dirty data back to memory.

The stream cache does not perform any coherency operations. It is up to the compiler/application to use the cacheable/uncacheable stream properties in a safe manner using explicit invalidate and flush operations. A large number of cached streams creates the possibility that the streams will interfere with each other in the cache and degrade performance. Typically, only one or two streams, which exhibit temporal locality, and are on the critical performance path will be cached. Consequently, the stream cache needs only 4 valid bits per line.

When the stream unit requests a memory word, the request is first checked in the local node's stream cache and satisfied from there in the case of a cache hit. If the case of a stream cache miss, the request is sent to the home node of the memory word. That node in turn will check its stream cache before reading the word from its main memory.

Due to the ack-based memory system, when a stream store completes it is safe to assume that all writes are committed to main memory. When a stream load completes, it is safe to assume that all reads are available in the SRF. Similarly, when a stream-cache-flush instruction completes, it is safe to assume that the dirty data has been committed to main memory.

Aside from these, the machine does not make any other consistency guarantees with respect to stream memory. In particular, in the case of a stream scatter with multiple updates to the same address, which write wins is undefined.

Hardware support to ensure coherency of read-write streams in multi-node configurations is expensive and does not match the stream programming model. Hence, the above stream cache specification was based on the assumption that stream accesses are read-only or write-only (i.e.) a stream is either a kernel input or a kernel output.

Scalar Cache Coherency and Consistency

Each segment has a *scalar-cache-memory-uncacheable* flag and a *scalar-cache-network-uncacheable* flag in its associated segment register. If the *scalar-cache-memory-uncacheable* flag is set, then any scalar processor memory requests to addresses in the home node's memory will not be cached. Similarly, if the *scalar-cache-network-uncacheable* flag is set, any scalar requests to addresses in a remote node's memory (i.e. requests which go over the network) will not be cached.

The scalar cache needs to support an explicit flush command, but it is not required to perform any coherence operations. It is up to the compiler/application to use the cacheable/uncacheable segment properties in a safe manner. The scalar cache does not see remote memory requests to its node since these go straight to the stream memory system. If multiple scalar processors use shared variables to communicate, then these

variables should be placed in segments which are both memory and network uncacheable.

Even though the stream units cannot see the scalar cache, the converse is not true. The scalar processor can see the stream caches, and if a local reference misses in the scalar cache then it may be satisfied by the stream cache rather than accessing main memory. However, scalar references which miss in the stream cache will not cause a line to be read into it. Data may be placed in the stream cache only as the result of a stream memory instruction whose MAR has the uncacheable bit(s) disabled.

The scalar processor should have a node memory barrier instruction similar to the SYNC instruction of the MIPS II. The compiler may use this instruction when it needs to guarantee the order of scalar memory instructions. The memory barrier instruction prevents new loads and stores from being issued until all the currently outstanding memory references from the scalar processor have completed.

The scalar processor is required to provide the following two consistency guarantees.

1. After a node memory barrier, all outstanding scalar accesses from that node will have been committed.

2. Multiple scalar accesses from a given scalar processor to the same memory address in its home node will exhibit sequential consistency. There are no guarantees with respect to the ordering of accesses to different addresses or to/from different processors.

The preceding scalar cache requirements were specifically aimed at minimizing modifications to any off-the-shelf scalar processor which may be used in Merrimac. The L1 cache (and L2 cache, if there is one) are accessible to the scalar processor alone, and not accessible to the stream units or other nodes. Unlike the stream cache, the scalar cache does not need bits for any kind of segment or stream identification. The memory consistency and coherence models are as relaxed and inexpensive as possible. The only additional requirement is that the processor should heed the scalar-uncacheable flag associated with Merrimac memory segments. This may be done by adding logic to check segment registers or by using a non-cacheable bit in the page table similar to the MIPS. Merrimac does not use paging or virtual memory, but an identity mapped TLB is acceptable if it is used only for the purpose of specifying cacheability attributes for memory ranges.

4.4.5 Atomic Memory Operations

Atomic memory operations (AMO) are a special class of memory operations that perform fetch&add, compare&swap etc. They perform memory read, modify and write sequences atomically. In conventional processors, these operations are used to implement various synchronization primitives such as barriers and locks. The concept of atomic memory operations for Merrimac was introduced in the report of year 2002. During the current year we have done a detailed analysis of the concept, designed the microarchitecture for an atomic memory operation unit, extended our simulation infrastructure to conduct related experiments and characterized the performance benefits using a small set of benchmark applications. The results were reported in a research paper at HPCA 2004 [2]. Scatter-add and gather-add operations, which are the data parallel version of the fetch&add operation, are introduced in Merrimac and they are used for computations like superposition operations and for parallel queue allocation. The microarchitecture of the atomic memory operation unit (AMU) and the multi-node expansion of AMO are detailed below.

Atomic Memory Operation Unit

The atomic memory operation unit (AMU), which is located in each internal memory bank, processes scatter-add and gather-add operations. All the scatter-add and gather-add requests are directly routed from

the bank input switch to the AMU. On receiving a scatter-add or gather-add request packet, it waits until the data corresponding to the target address is ready. Then it adds the data in the request packet with data from the target address, and sends the result to both the target address and the return location indicated in the request packet.

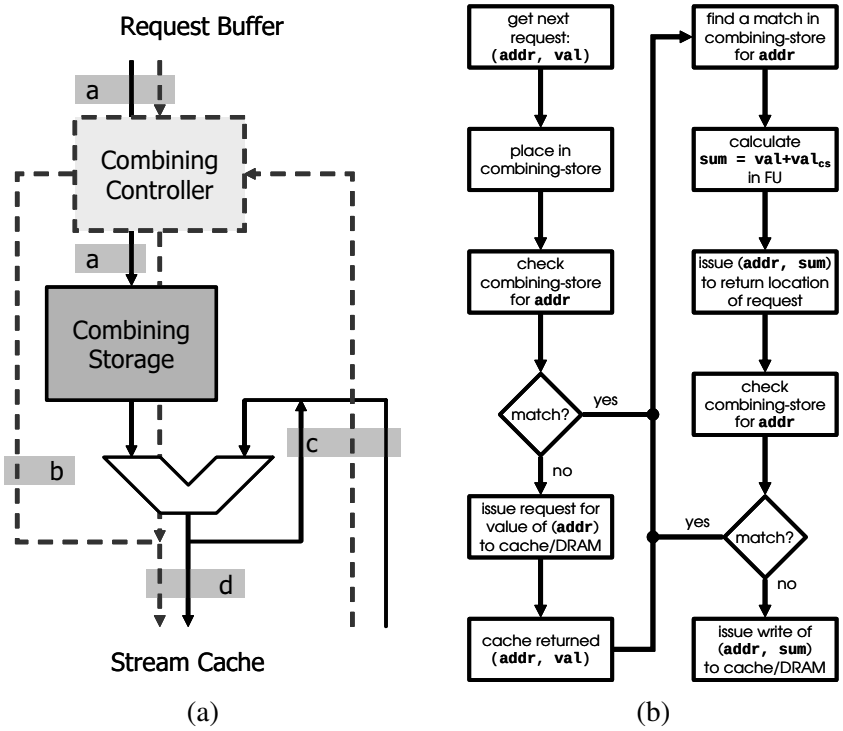


Figure 4.10: Atomic Memory Operation Unit (a) diagram, (b) control flow.

The AMU consists of a simple controller, multiplexers, a functional unit that performs integer and floating-point additions with fully-pipelined 4 cycle latency, and a combining store that is used to ensure atomicity of the operation (Figure 4.10). The combining store is analogous to the miss status handling register (MSHR) and write combining buffer of a data cache, and serves two purposes. First, it acts as an MSHR and buffers scatter-add and fetch-add requests until the original value is fetched from memory. Second, it buffers and orders scatter-add requests while a multi-cycle add operation is performed.

In Figure 4.10a, the alphabet annotations will be referred to in the order of operations performed. Solid lines represent data movement, and dotted lines represent address paths. The process is also depicted using a flow-diagram in Figure 4.10b. A gather-add or scatter-add request arrives at the AMU with a target address (addr), a value to be accumulated (val), and a return location. Scatter-add and gather-add requests must be performed atomically, and to guarantee this the AMU unit uses the combining store structure. Any scatter-add/gather-add request is placed in a free entry of the combining store, if no such entry exists, the scatter-add/gather-add operation stalls until an entry is freed. At the same time, the request address is compared to the addresses already buffered in the unit using a *content addressed memory* (CAM) (a). If the address does not match any active addition operations a request for the current memory value of this bin is sent to the memory system (b). If an entry matching the address is found in the combining store no new memory accesses are issued. When a current value returns from memory, it is summed with the combining store entry

of a matching address (again using a CAM search) (c). Once the sum is computed by the integer/floating point functional unit an return packet with summed value is sent back to the location marked in the request packet (d), and the combining store is checked once more for the address belonging to the computed sum. If a match is found, the newly computed sum acts as a returned memory value and repeats step c. If there are no more pending additions to this value, it is written out to memory (d). Since atomicity is achieved by the combining register, no further synchronization need take place. Using the combining register we are able to pipeline the addition operations and achieve high computational throughput.

Atomic Memory Operation in Multi-node

The procedure described above illustrated the use and details of the atomic memory operation within a single node Merrimac processor. When multiple nodes perform a atomic memory operation concurrently, the atomicity of each individual operation is guaranteed by the fact that a node can only directly access its own part of the global memory. The network interface directs memory requests to pass through the remote AMU where they are merged with local requests[1]. For multi-node configurations with local data-caches an optimization of this mechanism is to perform the AMO in two logical phases:

- A *local* phase in which a node performs an AMO on local and remote data within it's cache. If a remote memory value has to be brought into the cache, it is simply allocated with a value of 0 instead of being read from the remote node.
- In the *global* phase the global AMO is computed by performing a *sum-back* of the cached values. A sum-back is similar to a cache write-back except that the remote write-request appears as an AMO on the node owning the memory address.

The global sum is continuously updated as lines are evicted from the different caches (via sum-back), and to ensure the correct final result a flush-with-sum-back is performed as a synchronization step once all nodes complete.

4.5 Simulation Environment

4.5.1 Scalar Processor Integration

Until this past year, all of the simulated Merrimac results were obtained using a simulation infrastructure which had a cycle-accurate model of the stream execution units, but did not possess a similar model of the scalar (host) processor. Rather, the host code effectively executed in zero time. The same was true of the Imagine simulation infrastructure from which the Merrimac simulator was derived. When evaluating the arithmetic performance of a StreamC/KernelC application on a single Merrimac node, this simulation simplification was acceptable since the scalar overheads were minimal, as was confirmed for Imagine by comparing the simulator output with actual results from the prototype chip that was built. However, some of the scientific applications considered for Merrimac do contain non-trivial scalar portions, often related to data structure book-keeping. This results in inaccurate simulation results if the scalar code is ignored. Further, multi-node simulations exhibit interactions between the scalar processors on different nodes for

[1]The combining store will send an acknowledgment to the requesting node once an operation is complete

the purposes of synchronization, and this effect should also be simulated to achieve useful performance estimates.

To address this shortcoming in our simulation infrastructure, we took the GNU MIPS execution-driven scalar simulator which is bundled with GDB and extended it with a real memory system which was integrated with the Merrimac stream memory system. The GNU tool-chain was built to be a cross-compiler targeting this MIPS architecture, and we are now able to compile arbitrary C applications to run on the scalar core in a simulated Merrimac node. The MIPS Coprocessor 2 interface (COP2) is used to issue stream instructions and handle other communications with the stream units. The Low-Level Compiler takes the SVM output and produces a C program which contains embedded stream instructions (implemented as COP2 instructions), which is compiled in the cross-compiler to produce a binary executable for simulation.

4.5.2 Multi-Node Support

Until now, our architecture simulations focused on single node experiments because the stream virtual machine is expected to play the main part in identifying the multi-node characteristics of applications and their optimization. Pursuing cycle accurate simulator support for multi-node configurations raises the accuracy of measurement and provides valuable information for micro-architectural studies for areas such as multi-node atomic memory operation support.

Simulator Extension Multi-Node Configurations

Merrimac has a distributed shared global memory. Any data located in the memory of a node can be accessed by any other Merrimac node. When a node accesses the data located in a remote node, the memory access request is delivered to the remote node through the network interface of the local node, the router(s) in between and the network interface of the remote node. Acknowledgements and return data go through the same path.

To support multi-node memory accesses, a network interface and router were added to the Merrimac simulator. The network model implemented is an input-queued crossbar with back-pressure. It models the bandwidth of the Merrimac network, and it will be replaced by a more accurate model once the specification of the router is finalized. This extension was used to study the scalability of atomic memory operations in a multi-node system. Details follow.

Atomic Memory Operation Scalability Study

We explored the multi-node performance of scatter-adds with and without the optimizations described in Section 4.4.5. The multi-node system was comprised of 2–8 nodes. We ran experiments limiting the maximum per-node bandwidth to 1word/cycle (*low* configuration in the results below), and also 8words/cycle (*high* configuration). In addition to varying the network bandwidth we also turned the combining operation (Section 4.4.5) on and off. We concentrate on scatter-add alone and report the scalability numbers for only the scatter-add portion of the three test applications - random scatter-add accesses on different index ranges used for the histogram, molecular dynamics application called GROMACS, and the sparse matrix vector multiplication program called SPAS. For Histogram, we ran two separate data-sets, each with a total of 64K scatter-add references: *narrow* which has an index range of 256, and *wide* with a range of 1M. GROMACS uses the first 590K references which span $8,192$ unique indices, and SPAS uses the full set of 38K references from over $10,240$ indices of the element-by-element method.

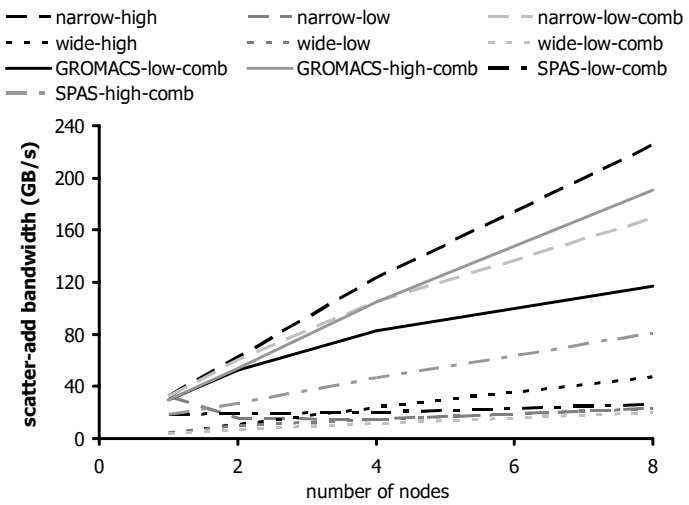


Figure 4.11: Multi-node scalability of scatter-add

Figure 4.11 presents the scatter-add throughput (additions per cycle) achieved for scaling 1–8 nodes, where each of the four sets of lines corresponds to a different application. The *wide* version of Histogram is limited by memory bandwidth in both the single node Amy multi-node cases. For any number of nodes, when using the high bandwidth network, it achieved perfect scaling. When the network bandwidth is low, performance is limited by the network and the application does not scale well. The optimization of combining in the caches does not increase performance due to the large range of addresses accessed, which leads to an extremely low cache hit rate. On the contrary, the added overhead of cache warm-up, cache combining, and the final synchronization step actually reduced performance and limited scalability.

In the *narrow* case of Histogram, the results are quite different as the high locality makes both combining within the scatter-add unit itself and in the cache very effective. No scaling is achieved in the case of the low-bandwidth network, although with high bandwidth we see excellent speedups of up to 7.1 for 8 nodes. Employing the multi-node optimization of local combining in the caches followed by global combining as cache lines are evicted provided a significant speedup as well. The reduction in network traffic allowed even the low bandwidth configuration to scale performance by a factor of 5.3 for 8 nodes. Again, the overhead of warming up the cache, flushing the cache, and synchronization prevents optimal scaling.

The molecular dynamics application (GROMACS) exhibits similar trends to *narrow*, as the locality in the neighbor lists is high. Scaling is not quite as good due to the fact that the overall range of addresses is larger, and the address distribution is not uniform. SPAS experiences similar problems but its scaling trend is closer to that of *wide*. The large index range of both these applications reduces the effectiveness of cache combining due to increased overheads incurred on cache misses and line evictions. Increasing the network bandwidth limits the effect of this overhead and scaling is improved (*GROMACS-high-comb* and *SPAS-high-comb* in Figure 4.11).

4.6 Programming Support

The Merrimac compilation system transforms programs written in a streaming style and produces code that runs efficiently on Merrimac hardware. The input to the compiler is a high level language such as C with language extensions for high level arrays. This approach is a different from the prior approach of using a stream programming language like Brook. After evaluating Brook over the past year, we concluded that much of the parallelism expressed in Brook can also be easily expressed in C with some minor extensions. However, it is important to note that the *streaming style of programming* emphasized by Brook continues to be used to write applications. Only the *syntax* in which the application is represented has changed. Also, it is easy to do syntax translation from Brook to extended C.

Figure 4.12 depicts the main steps involved in the transformation of extended C to Merrimac microcode. The dotted line (Applications → StreamC/KernelC → Merrimac Code) represents a tool chain that was used initially to get quick results to demonstrate the potential of streaming applications.

We currently use a two-phase structure for the compiler (depicted by the solid line), which is divided into a *high-level compiler* (HLC) and a *low-level compiler* (LLC). The interface between the two is based on the Stream Virtual Machine (SVM). HLC is being implemented in collaboration with Reservoir Labs, a private company that is developing a general purpose stream compiler for several morphware architectures with support from DARPA. We, at Stanford, are working very closely with Reservoir to retarget this compiler to Merrimac by adding features and optimizations that are specific to Merrimac. The SVM code generated by HLC is tested for correctness and for performance estimates using an SVM simulator.

The SVM code is transformed to Merrimac code using a low level compiler (LLC), which is being developed at Stanford. We use a freely available source analysis tool named *cTool* to convert SVM code into an intermediate representation amenable to performing low-level optimizations. These optimizations include loop unrolling, instruction scheduling, value placement in register banks, modulo scheduling, and generation of microcode.

The following sections describe the HLC and the LLC in more detail and also present the status of these efforts.

4.6.1 High Level Compiler (HLC)

HLC accepts programs written in the C programming language with some extensions, performs high-level mapping, and generates SVM code. Its input is a dialect of C with language extensions for high-level (abstract) arrays. Such extensions release HLC from the burden of maintaining C array layout semantics. The application's parallelism exposed as loop nests is sufficient to make appropriate scheduling, placement, and transformation decisions. HLC accepts a limited degree of dynamic control flow and data-dependent memory access, but such control is expressed in terms of idioms that are a good match for operations provided by the machine, for example, conditional stream element production and scatter/gather operations.

Figure 4.13 provides an overview of the main steps in generating SVM code using the R-Stream 1.9 compiler. First, the program written in *extended* C is transformed into an intermediate representation (IR) suitable for performing scalar transformations. These transformations include conventional data flow analyses like dead code elimination, constant propagation, etc. The IR is then converted to a parallel IR which is optimized in the *mapper* stage and transformed into *mapped* IR.

Mapping involves selecting the size of computation and memory grains, scheduling and placement of these grains on the machine, management of local streaming memories, and the generation of data transfer

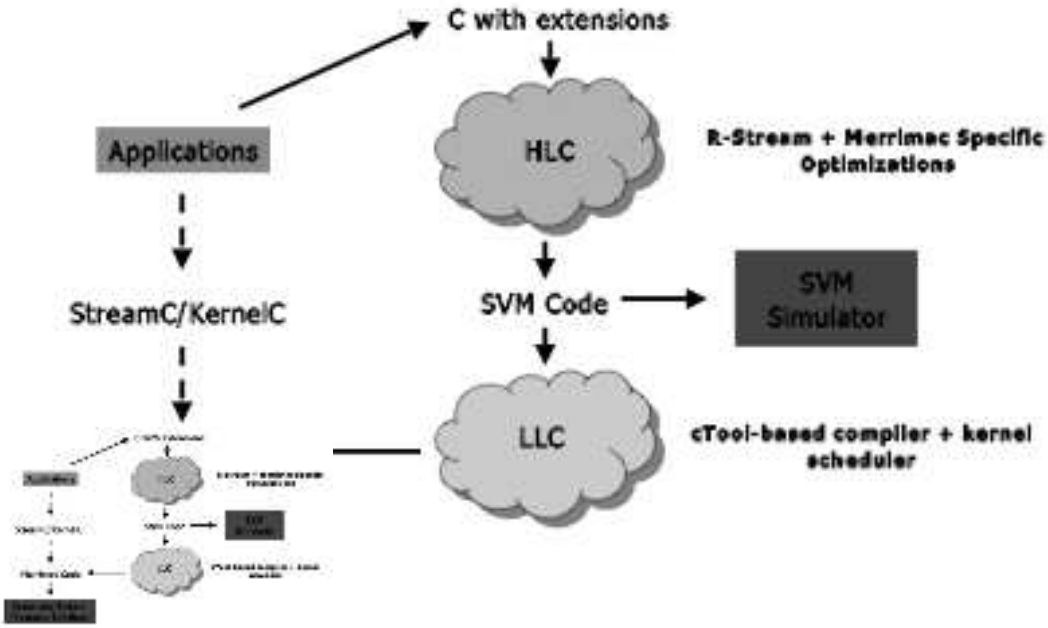


Figure 4.12: Overall Compiler Framework

operations. The mapped IR is then converted to a representation that the code generation phase uses to produce SVM code.

The SVM interface standard uses a C API that represents how kernels are partitioned and assigned to various stream processors, how data is split across various memories and how data transfers between various memories occur. The generated SVM code, encapsulates the partitioning of the program between control code and kernel code, in the ”streaming” computational model.

Realizing Merrimac’s performance advantage depends upon achieving greater degrees of arithmetic intensity than on a conventional architecture. Program transformations to increase arithmetic intensity have, for some time, been the objective of advanced compiler research. This research has resulted in a suite of ”classic” transformations of nested loops that improve locality of reference and thus cache effectiveness. Tiling and scalar replacement are good examples. R-Stream attempts to exploit this knowledge base in the design of the *mapper*. One challenge for using such classic loop transformations is determining the order in which they should be applied to optimize a given objective.

HLC uses a coarse description of the target machine, which is expressed in the Stream Machine Model (MM) description language, to make mapping decisions. The MM allows descriptions of the components of the target machine, particularly, the control processors, stream processors, the memory hierarchy, and the interconnect. For each of these units, descriptions of their computational, storage, and bandwidth capacities are provided in the MM. HLC uses the MM in its decisions on how to partition, place, and schedule the computation.

Figure 4.13: Skeleton of HLC (R-Stream) compiler

Current Status of HLC

In last year's ASC review, we had demonstrated the use of the R-Stream 1.0 compiler for programs written in Brook. The R-Stream compiler was completely rewritten for the next major release in September 2004 (R-Stream 1.9). This release, however, does not support many of the features essential for running our stream applications. We are working very closely with Reservoir to support these features in the next release (scheduled for Nov 2004). Some of these features are listed below.

- Scatter/Gather support: R-Stream 1.9 does not efficiently map loops that contain indexed accesses into arrays (e.g. a[index[i]] = c). The indexed accesses have to be converted to *Gather* and *Scatter* DMA calls in order to run efficiently on Merrimac's memory subsystem which supports gather and scatter operations in hardware. In order to support this feature, both the mapper and the code generation phases need to be modified. Supporting gathers/scatters is essential for compiling all of our major applications including FEM, MD, and LES.

- Support for SIMD clusters: The code generation phase of the R-Stream 1.9 compiler does not emit correct code to run on SIMD clusters. The current procedure for handling SIMD clusters in the compiler is the following. In the mapper phase, the SIMD clusters are viewed as separate processors each with $1/16$ the execution bandwidth, $1/16$ memory bandwidth and $1/16$ local memory (SRF) size, where 16 is the number of SIMD clusters. The SIMD lanes are allocated just by allocating the *local* memory of each processor with separate DMA calls. The data shared across clusters is replicated in each processor's local memory. We are currently modifying the code generation phase such that the DMA calls across the processors are coalesced into one call. In addition, we are also exploring alternatives to map more efficiently to SIMD clusters to avoid too many copies in the local memory due to replication.

- Support for multiple processors: In R-Stream 1.9, the machine model does not allow for multiple control scalar processors. The MM allows only one master/thread processor controlling one or more stream processors. Merrimac, however, has multiple thread processors executing in an SPMD fashion. We are currently working with Reservoir to support this feature in the upcoming release.

Because R-Stream's infrastructure was under active development at Reservoir Labs for the greater part of the past year, the focus of the Stanford compiler team has been on developing the low-level compiler which we discuss next.

4.6.2 LLC

The Stream Virtual Machine (SVM) abstracts and parameterizes features of different stream architectures. It acts as the intermediate target for the HLC (high-level compiler), which performs resource allocation for various stream architectures, including Merrimac. The SVM API provides a set of constructs to expresses the streaming portion of an application. They help to separate the communication and data-intensive code from control code, thus making it easy for the low-level compiler (LLC) to map to a specific stream architecture.

The SVM API includes three categories of execution and code:

Kernel (Slave): Computation and data-intensive functions that consume and produce streams on a single stream processor. Memory and other data movement operations are treated kernels and are predefined by the SVM.

Streams and Blocks: Streams and blocks are sequences of data elements allocated to a specific memory among those defined in the SVM machine model for the architecture in question. Kernels may only operate on streams or blocks allocated to the stream processor's local memory (SRF).

Control (Master): Scalar control code, which allocates space and sets up data in various memories, performs data transfer between levels of the memory organization (main memory and SRF), and initiates, monitors and terminates kernels.

The HLC extracts kernels, streams, and blocks, from the high-level input language, maps them to processors and memories, and performs global dependency analysis. The low-level compiler then translates the SVM constructs to hardware primitives and allocates any remaining low-level resources not exposed to the SVM.

Kernels

The SVM API for kernels includes several primitives for accessing data in a stream or a block, and only allows a subset of C for expressing the computation. The restrictions imposed do not allow pointers, global variables and dynamic memory allocation. Also the arrays are of fixed length and all control-flow is structured, including function calls with inline semantics only. These restrictions correspond well to KernelC restrictions, and we are able to leverage our existing kernel scheduler that has been in development for several years and dates back to the Imagine project. The LLC performs a source-to-source translation of SVM kernels into KernelC kernels and then schedules them for Merrimac using our existing tool. The source-to-source portion is based on cTool and performs the following transformations:

- Streams are treated as standard KernelC streams.
- Blocks are treated as indexable streams.
- Stream/block types are derived from types involved in SVM kernel API calls.

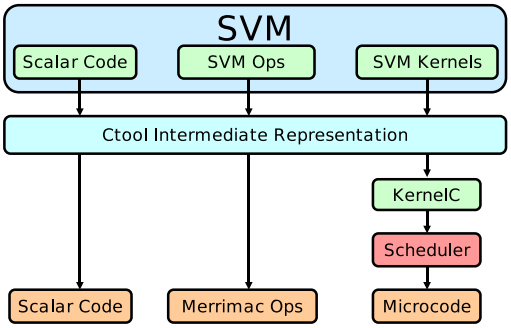


Figure 4.14: The LLC Framework

- Scalar kernel operands are transformed into micro-controller variables, which are mapped to uCRs.
- Structs are transformed into records.
- Direct transformations of arithmetic operations into KernelC syntax as detailed in Table 4.4. Automatic optimizations for reciprocals are not supported yet.
- Use of KernelC condition-code type for boolean variables and values.
- Loops in SVM kernels are changed to use KernelC loop constructs. The `loop_count` KernelC construct is used for `FOR` loops with known bounds, and `loop_while_all` is used when the SVM kernel loop iterates over an input stream.
- Conditional `IF` statements are predicated using KernelC `select` operations.
- Reductions are performed in each cluster independently, and the LLC inserts the appropriate KernelC code to combine the local reductions before the kernel returns.

We have finished the first-cut implementation of the source-to-source translator, and an example code transformation is shown below In Figure 4.15.

4.6.3 Scalar

For the scalar part of the SVM code, the LLC processes all SVM API calls and replaces them with Merrimac hardware instructions, while preserving the rest of the scalar code. The Merrimac instructions are dispatched to the stream processor at run-time. To handle this, `asm` directives conforming to the scalar-stream interface of Subsection 4.3.3 are embedded in the scalar code, and are processed by the scalar MIPS core in the hardware simulator.

This transformation requires building the *control flow graph* (CFG) and then generating an *intermediate representation* (IR) from the SVM code. This is followed by a dependency analysis, where data dependency

```
typedef struct {
  kernel kernel;
  IOStream* io;
  int mul;
} MulKernel;

void
mulKernelWork(MulKernel* k) {
  int x;

  for (x = 0; x < k->mul; x++) {
    streamPush(k->io, &x);
  }

  while (!streamPeekEOS(k->io,
                        0)) {
    streamPop(k->io, &x);
    k->mul *= x;
  }
}
```

SVM

```
kernel mulKernel(iostream<int> io,
                 uc<int>& _UC_mul) {
  int x;
  int _CL_mul;
  uc<int> _UCount;

  _CL_mul= commclperm(ucid(), x,
                      _UC_mul);

  _CL_mul = idiv_iter(_CL_mul,
                      NUM_CLUSTERS);

  _CL_mul= commclperm(0, _CL_mul,
                      _UCount);

  loop_count(_UC_mul_count) {
    io << x;
  }

  loop_stream(io) {
    io >> x;

    _CL_mul = commclperm(ucid(),
                         x,
                         _UC_mul);

    _CL_mul = _CL_mul * x;
    _CL_mul = commclperm(0, _CL_mul);

    _CL_mul = commclperm(NUM_CLUSTERS-1,
                         _CL_mul,
                         NUM_CLUSTERS-1,
                         mul);
  }
}
```

KernelC

Figure 4.15: Example of SVM kernel Transformation

is implicitly communicated through the use of streams or blocks, and the sizes of these data conduits is extracted just before a kernel call.

The LLC also allocates the various resources of the stream processor: SDRs for streams being used by kernels, MARs for streams involved in memory operations and scoreboard slots for all Merrimac instructions. The micro-controller registers (uCRs) are allocated by the kernel scheduler. The code generation phase requires the following operations:

1. Loading kernels into microcode store
2. Converting SVM API calls to Merrimac Instructions:
   - Each SVM kernel run is replaced by a sequence of Merrimac instructions: `SETMAR`, `SETSDR`, `UCRREAD/UCRWRITE`, and `RUN_KRNL`

- for each predefined kernel, i.e memory operations, corresponding Merrimac instructions are generated.
- SVM `kernelWait` calls for dependencies are replaced by spin-locks.

These steps also require dependency analysis to extract the dependencies generated due to the resources allocated. This dependencies are encoded into the Merrimac instructions using `ADD_DEP_S` and `ADD_DEP_C` instructions.

Resource Allocation

SDR and MAR allocations are done in a round-robin fashion, starting from the first index. For each kernel run, increment the SDR number for each input/output stream, and increment the MAR number for each block used by the kernel run. The scoreboard slots are similarly allocated in a round-robin fashion.

Currently, all kernels are loaded into micro store at the beginning, assuming they can all fit into the micro store.

Status

The current algorithms for slot assignment, SDR/MAR allocation and microcode loading, were very easy to implement, but they are not optimized in terms of performance.

We have verified the correctness of the LLC implementation by running some test examples on the simulator. Recent changes made to the SVM specification have also been reflected to the LLC.

4.6.4 Legacy Program Support

Cycle-accurate simulation of the Merrimac stream processor has demonstrated that scientific applications can achieve extremely high arithmetic performance on such an architecture. Often sustaining half or more of the processor's peak performance entails using the on-chip memory hierarchy to stage most of the data required by the computation and to minimize the number of references to off-chip memories. While it is beyond doubt that data-intensive applications coded in StreamC/KernelC perform well, applications coded in Brook also appear to be able to achieve a similar result. It is widely known that any system that requires applications to be recoded in a new language will meet with resistance. Hence, for a machine to be successful, legacy program support is critical.

Compiler researchers have been studying the problems of extracting parallelism from traditional C and Fortran programs and exploiting locality of reference to utilize processor memory hierarchies for decades. Loop parallelization and cache blocking are standard components of today's compilers, and many existing algorithms could be used as a part of a compiler which targets a stream processor such as Merrimac.

We have begun research on making use of existing compiler analyses to target legacy applications to Merrimac, and on developing new algorithms to address the new compilation problems which arise due to the different nature of stream architectures. More specifically, this effort is starting out by using the affine framework developed by Monica Lam and her students [10, 14] to map computational kernels coded in standard C using loop and array constructs. These affine programs comprise a best-case of compilable parallel programs, as the compiler is able to do static analysis to gain a perfect understanding of the dependences and data flow. A large number of useful computational kernels are affine programs, including many from

```
for (i = 0; i < 1024; i++) {
    for (j = 0; j < 1024; j++) {
        C[i,j] = 0.0;
        for (k = 0; k < 1024; k++) {
            C[i,j] = C[i,j] + A[i,k] * B[k,j];
        }
    }
}
```

Figure 4.16: Matrix Multiply Example

```
for (i = 0; i < 96; i++) {
    for (j = 0; j < 96; j++) {
        g[i,j] = 0.0;
        for (u = 0; u < 5; u++) {
            for (v = 0; v < 5; v++) {
                g[i,j] = g[i,j] + f[i+u,j+v] * h[u,v];
            }
        }
    }
}
```

Figure 4.17: 2D Convolution Example

linear algebra and signal processing domains. Non-affine program constructs, for example data-dependent conditionals, will be mapped conservatively.

Affine Programs and Affine Analysis

An affine expression is simply a linear combination of variables, and an affine program in one in which all loop bounds and array indices are affine expressions of the surrounding loop variables. For example, Figure 4.16 shows a simple dense matrix-matrix multiplication code.
Similarly, Figure 4.17 shows a simple 2D convolution code.
Each of these programs is affine, and further, there are no data-dependent constructs or non-analyzable pointer expressions, resulting in programs which provide the compiler with perfect information and the ability to do perfect static analyses. Note that some programs, such as the Fast Fourier Transform code, while not being affine in their usual formulation, can be made affine by flattening one or more loops.

Targeting Merrimac

In the case of both conventional architectures as well as Merrimac, the plan would be to take these computational kernels which comprise the core of applications and map them to minimize their execution times. The surrounding code, which does mundane tasks such as reading input files, formatting the data into arrays, and

so on, would be handled in the same manner on both types of machines: it would be compiled by a standard compiler such as gcc, and would make little or no use of the hardware designed to accelerate numerically intensive computation (i.e.) the clusters on Merrimac, SSE on the Pentium line of processors, etc. On a conventional architecture, the affine computational kernel would be parallelized into multiple independent loop partitions if possible, or into partitions with the minimum number of synchronization operations both within and between them. Subsequently, these parallel loop partitions would be mapped to different processors in a parallel machine. To achieve good performance on each processor, the partitions would be blocked to fit its cache.

On Merrimac, the same partitioning and blocking concepts are used, but the situation is made much more complicated on account of the processor's architecture. In particular, the global on-chip cache of conventional processors has been replaced by the multi-lane SRF and the per-cluster LRFs which require explicit management by the compiler. Further, unlike the situation of targeting conventional parallel processors, in which each processor has an independent instruction stream and can handle arbitrary conditionals with ease, in Merrimac, the parallel clusters are SIMD, and the usual techniques used to partition parallelized loops introduce control flow statements which do not easily map to the machine.

Consider the simple example of a program containing the array expression `A[2*i+1]`. On a conventional architecture, the following run-time actions are performed for this expression.

1. The processor computes the array index expression `t1=2*i+1` for the current value of `i`; suppose `i=3`, then `t1=2*3+1=7`.

2. The processor maps the array index value to the memory address used to hold that data; supposing the array is composed of 4-byte words and its base address in memory is `A_base=1000`, the processor will compute `t2=A_base+t1*4=1028`.

3. The memory address 1028 is actually a global name for the piece of data, which the cache maps to an on-chip storage location via an associative lookup mechanism. The actual location of the piece of data on the chip is not decided until run-time when it is reactively fetched by the cache upon a miss, and subsequent references to that piece of data will map to the same cache location.

In Merrimac, none of these steps happen at run-time. The input data is laid out in the SRF in such a way that the kernel only needs to pop the next value, rather than explicitly compute the index of the next value. Further, there is no run-time mapping of data to on-chip storage locations as there is with a cache; instead, the SRF and LRFs are explicitly allocated by the compiler. Finally, not only must the SRF and LRF locations be allocated, but the communication required to move data between clusters must also be scheduled at compile time, unlike in a conventional processor in which the global on-chip cache handles all data movement.

This is just one example of the complicated task of targeting an application to Merrimac. To guide this research, some small applications like the matrix-matrix multiply shown above were hand-coded and simulated. Some of the necessary optimizations to achieve the best possible utilization of Merrimac are listed below.

- The size and shape of the blocks which are mapped to the SRF must be chosen to achieve a high ratio of computation to off-chip communication. This is the fundamental optimization which enables the levels of performance observed in stream processors: the data is staged in an on-chip storage rather than being repeatedly fetched from off-chip memory.

- The size and shape of the blocks must balance the idle time due to waiting on memory with the overhead of starting and stopping kernels in the clusters. For example, using deeper software pipelining in the SRF makes the blocks smaller, resulting in relatively more kernel startup overhead, but also allows better overlap of kernel execution with memory loads and stores, reducing the amount of time the clusters are idle waiting for memory.

- There is a tradeoff between replication of data in the SRF and communication via the inter-cluster switch. If a kernel is communication limited, as some signal processing kernels are, then it is advantageous to replicate the data in the SRF to avoid the need to communicate it between clusters. The downside to replication, though, is that it effectively reduces the available SRF capacity, which results in smaller blocks. The same problems alluded to above of increased kernel overheads also apply.

To date, this research effort has identified some of the new algorithms which are required to adapt existing compiler analysis tools to Merrimac in particular and stream processors in general, and is working on their solutions. Some simple programs have been hand-targeted to Merrimac to spur the process along and provide insight about the optimizations which are required to achieve good performance.

Relationship to Reservoir's R-Stream Compiler

As described above, Reservoir is also working towards a compiler which will target a C-like language to Merrimac and other stream processors. While this research and Reservoir's compiler development overlap to some extent, their basic goals are different enough to warrant independent efforts at the moment. It is possible, however, that at some point in the future this research will be folded into their production compiler. The primary differences are as follows.

- Reservoir is developing a production compiler to target a number of stream architectures, focusing first on tiled architectures such as RAW and secondly on Imagine- and Merrimac-like SIMD stream architectures. Their goal is to develop a commercial tool which will handle completely general applications and target multiple platforms.

- This research effort is specifically looking at the algorithms needed to target Merrimac- and Imagine-like architectures and is not attempting to produce an industrial-strength compiler, and thus is able to focus immediately on the specific problems of mapping to Merrimac.

4.7 Merrimac applications

During the last year, the Merrimac applications group focused on the performance of scientific applications on two types of streaming processors: the Merrimac processor and Graphics Processing Units (GPU), which are found in high end graphics cards.

On the Merrimac processor, we used the compiler and simulation tools available to port and fine-tune two applications: Gromacs (a molecular dynamics code for protein simulations) and CDP the turbulence simulation code from the Combustor simulation group. The goal was to evaluate various algorithms and hardware implementation of those codes, along with a complete schedule analysis. The role of conditional streams was found to increase performance significantly for highly unstructured data sets such as for Gromacs. For CDP, we implemented a WENO scheme to solve convection-diffusion equations. The progress

was somewhat limited due to delays in the development of compiling tools for Brook-to-SVM. Those limitations will be addressed during the coming year.

GPUs are a very attracting alternative to test the potential of streaming processors. Even though they are specialized for graphics applications, it is possible to program general scientific codes within certain restrictions. On a model problem in finite-element for solid mechanics, we demonstrated a speed-up of a factor greater than 10 over highly optimized code for latest generation Pentium processors. We have developed a programming environment, including compiler and run-time libraries, so that codes written in Brook can be run seamlessly on GPUs. This greatly simplifies the relative complexity of programming those cards and make GPUs available to the general scientific programmer without requiring any special knowledge about graphics card hardware and graphic languages. A full application, Gromacs, was ported to GPUs using those tools. The implementation and testing are almost complete now and we should obtain performance results during the fall quarter. This demonstrate both the potential of GPUs for scientific applications and also the relative ease for the programmer to port codes to this new platform.

The sections are as follows. The first two sections are on Gromacs and CDP on Merrimac. Then we present the Brook programming environment for GPUs. Finally we end with two applications on GPUS: Gromacs and a finite-element code for solid mechanics applications.

4.7.1 StreamMD

Molecular Dynamics is the technique of simulating detailed atomic models of molecular systems in order to determine the kinetic and thermodynamic properties of such systems. GROMACS [13] is an engine that performs molecular dynamics simulations by repeatedly solving Newton's equations of motion for all the atoms in a particular system. The simulations follow a discrete timescale, and are used to compute various observables of interest. Often, such simulations are used to design new materials or study biological molecules such as proteins, DNA. When simulating the latter, it is common to surround those molecules with a large number of water molecules. In that case a significant portion of the calculation is spent calculating long ranged interactions between water molecules and between the protein and the surrounding water molecules.

GROMACS is highly optimized to take advantage of mechanisms found in commercial high-end processors including hand optimized loops using SSE, 3DNow!, and Altivec instructions [12, 1, 11]. One of the numerical method employed by GROMACS for long ranged forces uses a cut-off distance approximation. All interactions between particles which are at a distance greater than r_c are approximated as exerting no force. This approximation limits the number of interactions calculated for the system from $O(n^2)$ to $O(n)$, where each particle (we will refer to this as the *central particle*) only interacts with a small number of neighboring particles. The list of neighbors for each particle is calculated in scalar-code and passed to the stream program through memory along with the particle positions. The overhead of the neighbor list is kept to a minimum by only generating it once every several time-steps. The accuracy of the calculation is maintained by artificially increasing the cutoff distance beyond what is strictly required by the physics. We have chosen this phase of the GROMACS algorithm to test a streaming implementation on Merrimac – StreamMD. This year we continued refining our implementation and also explored an additional 3 variants of the code. The best current implementation improves on last year's result by over 76%, and based on simulation results we estimate that it would outperform a 3.4GHz Pentium 4, produced using the same 90nm semiconductor technology as Merrimac, by a factor of 13.2. The full details of the algorithms used, trade-offs explored, and results obtained will be published in the proceedings of the SC'04 conference in November 2004 [4].

StreamMD interfaces with GROMACS through Merrimac's single global memory space. GROMACS provides the configuration data consisting of a position array containing nine coordinates for each molecule (because we are using water molecules these are the coordinates for each of the three atoms), the central molecule indices stream `i_central` (one element per molecule), and the neighboring molecules indices stream `i_neighbor` (one element per interaction). These streams index into the position input array and the force output array.

The basic flow of StreamMD is as follows:

1. Gather the positions of the interacting molecules into the SRF using Merrimac's hardware gather mechanism.
2. Run the force computation kernel over all interactions, reading the required data from the predictable SRF. The kernel's output is a stream of partial-forces and a stream of indices which maps each force to a molecule. Those partial forces are subsequently reduced to form the complete force acting on each atom.
3. Reduce the partial forces. This is achieved using Merrimac's scatter-add feature.

To attain high performance on Merrimac we must ensure our program has latency tolerance, parallelism, and locality. We explored several methods to try and maximize performance using a combination of replication, addition of dummy atoms, and Merrimac's *conditional streams* [8] and *indexed SRF* [6] mechanisms.

Latency Tolerance In all our implementations latency tolerance was achieved by strip-mining (blocking) the force computation kernel, and then software pipelining the resulting loop: while a kernel is busy processing a strip of the input data, the forces computed on the previous strip are reduced to memory and the position data belonging to the next strip is loaded. In this way the memory operations are overlapped with useful computation.

Locality The locality in StreamMD is a short term producer consumer locality (*i.e.* within a single kernel). It is the result of computing 9 full atom-atom forces while reading the positions of just two molecules in every kernel iteration, capturing intermediate results used for these force calculations, and the internal reduction of the partial forces of a central molecule within a cluster. This locality is entirely captured within the LRFs. The amount of arithmetic intensity changes with the implementation chosen as discussed below.

Parallelism StreamMD has abundant parallelism as each interaction can be calculated independently of all other interactions, and the forces reduced in a separate step. This parallelism is used to operate Merrimac's 16 clusters in SIMD fashion. This requires that all compute clusters perform the same operations every cycle.

The simplest implementation (*expanded* in Table 4.5) is to fully expand the interaction lists, so that the kernel reads 16 pairs of interacting molecules each iteration – one pair per cluster. Thus, each cluster reads in two interacting molecules and produces two partial forces. This requires 27 input words (18 words for the coordinates of the 6 atoms, and 9 words for periodic boundary conditions) and 18 output words (for the two partial forces). In addition three index streams must also be read, for a total of 48 words for the 234 operations of each interaction. The partial forces are reduced using the hardware scatter-add mechanism (Section 4.4.5).

A different implementation, which is still pure SIMD, is to use a fixed-length neighbor list of length L for each central molecule (*fixed* in Table 4.5). Now each cluster reads a central molecule once every L

iterations and a neighbor molecule each iteration. The partial forces computed for the L interactions of the central molecule are reduced within the cluster to save on output bandwidth. This configuration requires 9 words of input and 9 words of output every iteration (for the position and partial forces of the neighboring molecule), and 18 words of input and 9 words of output for every L iterations (for the central molecules position, boundary condition, and partial forces). Including the index streams we end up with $18 + 1 + \frac{27+2}{L}$ words per iteration per cluster. If we choose a fixed length of $L = 8$ the result is 22.6 words per iteration instead of 48 of the naive implementation. Since the number of neighbors for each molecule is typically larger than L, central molecules are repeated in `i_central` in order to accommodate all their neighbors. Furthermore, whenever a central molecule does not have a multiple of L neighbor molecules, some dummy neighbor molecules are added to `i_neighbor` to keep the fixed $L : 1$ correspondence (Figure 4.18).

p(0)	p(1)	p(2)	p(3)	$\cdots$	p(N)

Original position array `position`

0^1	0^2	1^1	1^2	1^3	2^1	$\cdots$	$\cdots$	N^1

Central molecule index stream `i_central`

p(0^1)	p(0^2)	p(1^1)	p(1^2)	p(1^3)	p(2^1)	$\cdots$	$\cdots$	p(N^1)

Gathered central position stream `c_positions`

p(0^1_1)	·	p(0^1_L)	p(0^2_1)	·	p(0^2_L)	p(1^1_1)	·	p(1^1_L)	$\cdots$

Gathered neighbor position stream `n_positions`

Figure 4.18: Central molecules are repeated and neighbors are padded for fixed length neighbor lists

The disadvantages of this scheme are the extra bandwidth required for reading the repeated molecules and the dummy values, as well as the extra computation performed on the dummy neighbors. This computation does not contribute to the solution but consumes resources. The overheads are typically not large for a reasonable value of L (between 8 and 32).

A third implementation option relies on Merrimac's inter-cluster communication, *conditional streams* mechanism [8], and the *indexable SRF* [6] (*variable* in Table 4.5). These mechanisms allow a SIMD architecture to process inputs and produce outputs at a different rate for each cluster. We use this to implement variable-length neighbor lists for each central molecule. Instead of reading a new central molecule once every L iterations, each cluster checks on each iteration whether a new molecule is required or not. To adhere to SIMD restrictions, the instructions for reading a new central position and the writing of the partial forces for the central molecule are issued on every iteration with a condition. Unless a new molecule is required the instructions have no effect. This leads to a slight overhead of unexecuted instructions within the kernel, but since they are not floating-point arithmetic instructions they have little detrimental effect on the overall kernel efficiency. The exact number of words required for an iteration depends on the number of neighbors per central molecule, but as a minimum 10 words of input are consumed and 9 words are produced for every iteration in order to read the neighbor coordinates and index, and write the partial force acting on that neighbor.

Results The arithmetic intensity and locality are tightly related and strongly dependent on the specific

implementation. Table 4.6 summarizes the arithmetic intensity as calculated above while explaining of the variants, based on the algorithm properties, as well as the measured operations per input/output word for the actual data set. We also calculated the arithmetic intensity more accurately based on the actual data set used (without actually running the code), and present it in Table 4.6 as well (these values appear in parentheses). The small differences between the theoretical and measured numbers attest to the ability of the compilation tools to efficiently utilize the register hierarchy of Merrimac. A more direct measure of locality is the number of operands supplied by each level of Merrimac's storage hierarchy. In all the implementation, other than *fixed*, over 95% of references were supplied from the local register files. For *fixed* this figure was 89%, but still only 3% of operands were fetched or stored from or to main memory.

Exception Type	Affected FUs	Static/Dynamic	Hardware Response
Illegal opcode	ALL	static	treat as NOP and return arbitrary result after 1 cycle
Illegal bus	ALL	static	return arbitrary value
Illegal stream ID	INOUT/JB/uC ALU	static	treat as arbitrary valid stream
Invalid LRF config	MULADD	static	treat as an arbitrary configuration
Illegal index	INOUT	dynamic	return arbitrary value
TRAP instruction	MULADD/uC ALU	dynamic	raise a TRAP flag and record PC, cluster ID, and FU number in the exceptions unit.
FP INVALID	MULADD	dynamic	produce NaNand record PC, cluster ID, and FU ID
FP OVERFLOW	MULADD	dynamic	produce $\pm\infty$ and record PC, cluster ID, and FU ID
FP UNDERFLOW	MULADD	dynamic	produce 0 and record PC, cluster ID, and FU ID
FP DVBZ	ITER	dynamic	produce $+\infty$ and record PC, cluster ID, and FU ID
MEM_SEG SEGMENT	MEM	dynamic	record AG and stream element number
MEM_FP INVALID	MEM	dynamic	produce $+\infty$ and record AG, stream element number, and AMO bank ID
MEM_FP OVERFLOW	MEM	dynamic	produce $+\infty$ and record AG, stream element number, and AMO bank ID
MEM_FP UNDERFLOW	MEM	dynamic	produce $+\infty$ and record AG, stream element number, and AMO bank ID

Table 4.1: Exception Events and Hardware Response (uC ALU refers to the micro-controller ALU)

Operation	NaN Produced By
$+$	$\infty + (-\infty)$
$\times$	$(\pm 0) \times (\pm\infty)$
$\frac{1}{\sqrt{x}}$	$\frac{1}{\sqrt{x}}$ when $x < 0$

Table 4.2: NaN Producing Operations

Logical	Physical	Description
0	uCR[63]	transfer register for scalar/stream communication
2	STRINST_{127-64}	Bits 127 – 64 of the stream instruction register
4	STRINST_{63-0}	Bits 63 – 0 of the stream instruction register
6	SCRBRDS	the score-board started status register
8	SCRBRDC	the score-board completion status register
16	SCNTRL	stream unit control register
10	KRNLEX_TR	the kernel `TRAP` exception register
11	KRNLEX_IV	the kernel `INVALID` exception register
12	KRNLEX_DBZ	the kernel divide-by-zero exception register
13	KRNLEX_OF	the kernel `OVERFLOW` exception register
14	KRNLEX_UF	the kernel `UNDERFLOW` exception register
15	AG0_SEG	AG0 segment violation exception register
16	AG0_IV	AG0 `INVALID` exception register
17	AG0_DBZ	AG0 divide-by-zero exception register
18	AG0_OF	AG0 `OVERFLOW` exception register
19	AG0_UF	AG0 `UNDERFLOW` exception register
20	AG0_ID	Scoreboard entry number for the instructions that executed on AG0
21	AG1_SEG	AG1 segment violation exception register
22	AG1_IV	AG1 `INVALID` exception register
23	AG1_DBZ	AG1 divide-by-zero exception register
24	AG1_OF	AG1 `OVERFLOW` exception register
25	AG1_UF	AG1 `UNDERFLOW` exception register
26	AG1_ID	Scoreboard entry number for the instructions that executed on AG1

Table 4.3: Stream Logical to Physical Register Mapping

Organization	Total Number of Registers per MULADD	Area in mm^2
Original	192	0.1008
5-ported	64	0.07995
Switched	96	$0.0567 + switcharea$

SVM	KernelC
`sqrt(x)`	`fsqrt_iter(x)`
`fabs(x)`	`trunc(x)`
`x % y`	`imod_iter(x, y)`
`(int)x / (int)y`	`idiv_iter(x, y)`
`(double)x / (double)y`	`fdiv_iter(x, y)`

Table 4.4: Arithmetic Operations Transformations from SVM to KernelC

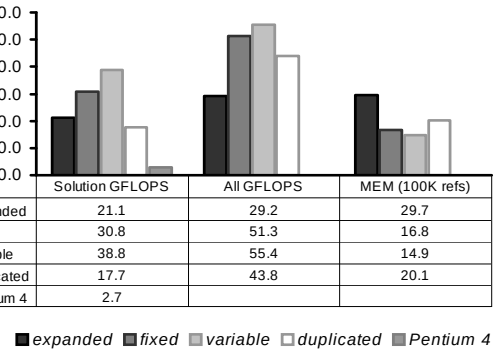
Name	Description
expanded	fully expanded interaction list
fixed	fixed length neighbor list of 8 neighbors
variable	reduction with variable length list
duplicated	fixed length lists with duplicated computation
Pentium 4	fully hand-optimized GROMACS on a Pentium 4 with single-precision SSE (water-water only)

Table 4.5: Variants of StreamMD

Variant	Calculated	Measured
expanded	4.9	4.9
fixed	10.3 (8.6)	8.6
variable	12.0 (9.9)	9.7
duplicated	8.0	7.2

Table 4.6: Arithmetic intensity of the various StreamMD implementations. Values in parenthesis are more accurate calculations accounting for the actual data set properties.

Figure 4.19 shows the performance results for the various implementations. The left-most bar group represents the rate of computation for floating-point operations directly contributing to the water-water interaction force calculation, and corresponds to the time-to-solution – a higher value indicates faster execution. The central group is the execution rate of all floating-point operations on the hardware. These include operations on dummy neighbors, kernel initialization, and other overheads. The right-most group shows the total number of memory accesses performed. While a higher total execution rate or lower memory reference count do not correlate with better algorithm performance we can clearly see the various trade-offs made between increased amount of computation and memory bandwidth requirements of the difference implementation choices. At this point we only estimate the performance on a conventional processor based on the wall-clock time of simulating the same data set for 1000 time-steps. We are assuming the actual force calculation accounts for 75% of the time.



	Solution GFLOPS	All GFLOPS	MEM (100K refs)
expanded	21.1	29.2	29.7
fixed	30.8	51.3	16.8
variable	38.8	55.4	14.9
duplicated	17.7	43.8	20.1
Pentium 4	2.7		

Figure 4.19: Performance of the different StreamMD implementations

The highest performance implementation is *variable* which outperforms *expanded* by 84%, *fixed* by 26%, *duplicated* by 119%, and GROMACS on a 3.4GHz Pentium 4 by a factor of 13.2.

Not only does Merrimac outperform a conventional processor fabricated with the same semiconductor process by a factor of over 13, Merrimac also performs 64-bit double-precision computations at full bandwidth as opposed to single-precision operations on the Pentium 4.

4.7.2 StreamCDP

StreamCDP is a transport-advection equation solver which is used within the jet engine simulation routines. This partial differential equation governs two very important scalars in the system: the mixture fraction and the reaction progress. Mixture fraction is a variable which ranges from 0 to 1, indicating the percentage of fuel in air. This is a conserved variable since neither air nor fuel is generated within the engine. The reaction progress variable indicates, in percentage, the progress of the reaction. Reaction progress however is not a conserved variable. We require both of these scalars to determine the state variables. It must be appreciated that the state variables are highly nonlinear functions of these scalars. This puts a very stringent restriction on the numerical methods, since even small spurious oscillations can result in drastically in-correct state variables.

Keeping this in mind, the WENO (Weighted Essentially Non-Oscillatory) is well-suited for this application (see Figure 4.20 for a flow chart). The way it works is to divide up the computational region into an unstructured mesh and calculate the scalars at the cell centers. The WENO scheme is a second order

scheme, which means that the value of the scalar at each time step depends on the scalar values of its neighbors and neighbors of neighbors. The information from neighbors appears in the form of flux interchanges. The WENO scheme makes a particular choice of flux based on the gradient information. The gradient can be calculated using an average over neighboring cell values or it can be an upwind gradient (see Figure 4.21 for an illustration of the gradient calculation). The algorithm selects a linear combination of both based on the fact that we want the solution to be non oscillatory. At each stage, we use the residual to check for convergence. When the residual is within a tolerance level, the algorithm terminates and we obtain the values of the scalar variable at the next time level for the whole grid.

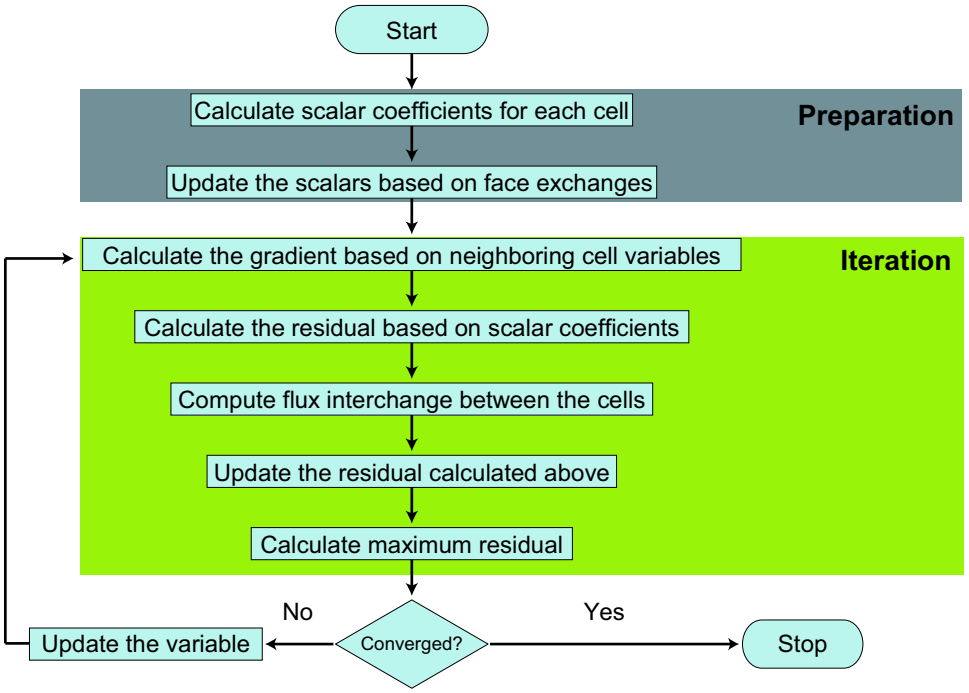


Figure 4.20: Flow chart of the WENO scheme.

There are many ways to organize the calculation. There are two basic steps in each iteration: flux exchange and residue collection. Both of these could be arranged either like a cell based calculation or a face based calculation. A cell based calculation loops over all cells and performs the calculation, similarly a face based calculation loops over all the faces.

Consider the cell based calculation. For flux interchange, we loop over all the cells and calculate the flux on each cell due to all its neighbors. In the next step, we loop over all the cells to calculate the residual in each.

Consider the face based calculation. For flux calculation, we loop over all the faces and calculate the flux exchange which takes place through that face. This is added to one cell and subtracted from the other associated with the face. Now, in the residue collection phase, we again loop over all faces and calculate the contributions to the residue due to the two cells on each side of the face.

The calculation must be organized so as to maximize the arithmetic operations with respect to data transfer. The ratio of arithmetic operations to data transfer of the cell based computations are dependent

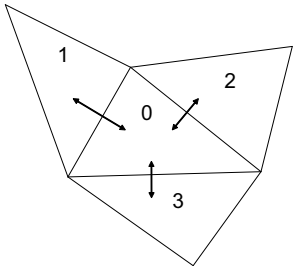


Figure 4.21: Gradient at cell 0 is a linear combination of three pre-determined vectors scaled by the value of the scalar variable in those cells. This is how the second order information enters the system.

on the number of neighbors of each cell. On the other hand, the face based computations have a constant ratio of arithmetic operations to data transfer. The Figure 4.22 illustrates this effect. The least number of neighbors a cell can have in a 3D grid is four. Thus from the figure we can infer that we must use *cell based* residue collection and *face based* flux interchange.

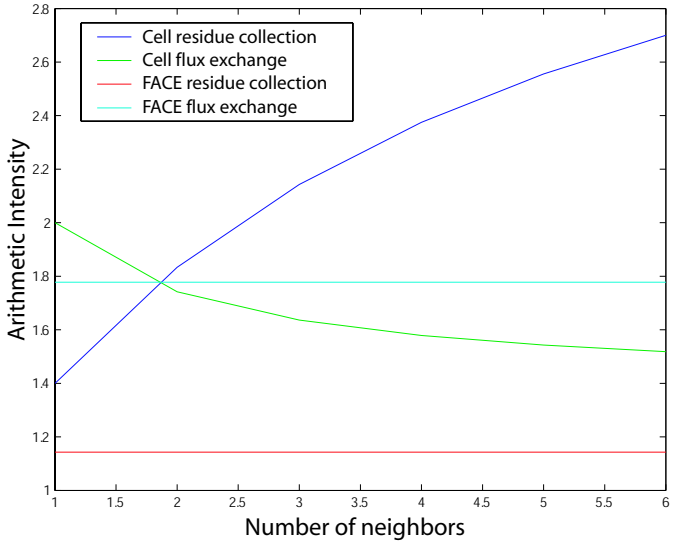


Figure 4.22: Arithmetic intensity of WENO kernels as a function of number of neighboring cells for cell based and face based schemes.

4.7.3 Brook for GPUs

During the first year of this project, we built a prototype of a new streaming programming environment titled Brook. Brook extends standard ANSI C to incorporate the idea of stream computing to produce code that is highly parallel and efficient for stream architectures. The Brook prototype defined many of the operations that we identified as fundamental for a streaming programming language. The second year of the project has been spent working with the compiler group to formalize the language into a mature, well defined, platform. The rapid success and adaptation of this new programming language has been tremendous. Working with

both application and compiler developers over the past 2 years has resulted in a stable, mature, and robust data parallel language that is ideally suited for streaming architectures. However, one of the main limitations in making Brook a mainstream parallel programming language is that current Brook compilers only target architectures that only exist in simulation. Many application developers where excited by the simulated performance numbers from the Merrimac simulators, however generating simulated results takes time and can limit progress in developing new streaming applications and algorithms.

This past third year of the project has been spent exploring the possibility of using our streaming programming model on other streaming platforms, specifically the commodity programmable graphics hardware. Today's GPUs (Graphics Processor Units) boast over 60 GFLOPS of compute performance and peak 36 GB/sec memory bandwidth. Furthermore, the programmable fragment portion of the graphics hardware mimics the stream model proposed by the Stream Virtual Machine. Programmable GPUs are commonplace in today's workstations and are an excellent target platform for Brook. By providing a Brook GPU implementation, we can (a) better understand the compilation process by addressing some of the challenges of mapping kernels and streams to the GPU, (b) continue to explore language additions and extensions to the streaming model from the perspective of graphics hardware, and (c) help motivate application writers by providing them with a streaming platform which they can see significant performance improvements today.

For this project, we implemented a Brook compiler and runtime system targeted to run on modern GPU hardware (Figure 4.23). This work was done separately from the Reservoir effort to take advantage of some of the expert GPU programming knowledge from the Stanford Graphics Lab as well as to provide an open-source solution which will be publicly available. The Brook compiler, called BRCC, is a source to source compiler. BRCC takes as input Brook code files, builds a parse tree, converts the Brook streams and kernels into GPU primitives, and generates customized C++ code. The resulting C++ code includes calls into the BRT (Brook Run Time) library, which implements the execution of kernels on the GPU, manages stream data resident in GPU memory, and supports the stream operators. The BRT supports both the NVIDIA and ATI hardware, which are the two leading graphics card vendors. The BRT interfaces with the hardware through DirectX and OpenGL, the two leading graphics APIs, and includes a fallback x86 CPU implementation. The system also runs on both Linux and Windows platforms.

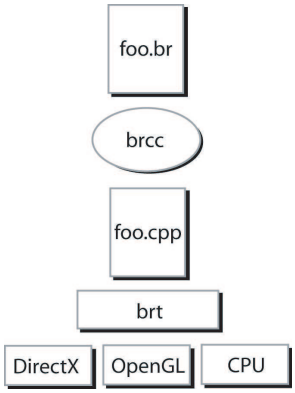


Figure 4.23: Brook compiler: path from Brook files to DirectX, OpenGL and CPU.

Implementing Brook on GPUs has allowed us to explore some additional language features that were specific to graphics hardware. Today's GPUs allow for indirect reads, or gathers, directly inside of kernels. We added this capability to the Brook GPU version through the use of standard C array syntax. In this

example (see Figure 4.24), we see that the stream "array" is passed as a gather argument to the kernel and the programmer is allowed to indirectly read from the stream. This functionality eliminated the need for many of the stream operators previously present in the Merrimac Brook implementation. The only operators that needed to remain were the streamRead (for loading stream data), streamWrite (for writing stream data), Domain (for reading from or writing to portion of a stream), and ScatterOp.

```
kernel void foo (float a<>, float t,
                 float array[],
             out float result<>) {
      result = array[a] + t;
}

float a<100>;
float c<100>;
float array<25>

foo(a,3.2f,array,c);
```

Figure 4.24: Example of an indirect read in Brook using streams and arrays.

We have published our findings in a paper presented at SIGGRAPH'04 titled, "Brook for GPUs: Stream Computing on Graphics Hardware." In this paper, we discuss the challenges of mapping Brook onto the GPU and how the system overcomes these difficulties through splitting kernels, restructuring stream data, and virtualizing the memory system. On Figure 4.25 are performance results for the SEGMV and SAXPY BLAS kernels, an image segmentation application, FFT, and a ray tracing application, normalized to an optimized CPU implementation running on a 3GHz Pentium 4.

We have made all of our source code and Brook applications for the GPU available at http://www.sourceforge.net/projects/brook. We have had over 6,700 downloads of the Brook software and over 220,000 page hits since the first software releasing in January 2004 thank in part to features in IEEE Computer, Slashdot, and GPGPU.org. By providing Brook with a streaming platform available today, we have not only been able to learn a lot about the streaming programming model but we have inspired an entire community of application developers to explore streaming as viable computing abstraction.

4.7.4 Gromacs on GPU

Understanding the structure and function of macromolecules has become the cornerstone of modern biology. One very useful tool that has evolved towards this goal is Molecular Dynamics (MD), which is a way to simulate the behavior of macromolecules at an atomistic level of detail. However the price to be paid for such a fine grained understanding is computational power. MD works by time stepping a system of differential equations that describes the interactions of collections of atoms. The time increment needed to obtain a reasonable solution is a few femtoseconds (10^{-15}s), while the processes of biological interest usually take anywhere from microseconds (10^{-6}s) to hours or even days. This means that one needs to compute at least a billion time steps before MD results become biologically relevant. Consequently there is a great demand for high speed MD code.

Towards this end, the use of stream computing for MD is very appealing. We have taken the source code of the widely used open source GROMACS software package and ported key portions of the code to Brook, a high level language designed for stream computing. Compared to last year, we made a very significant

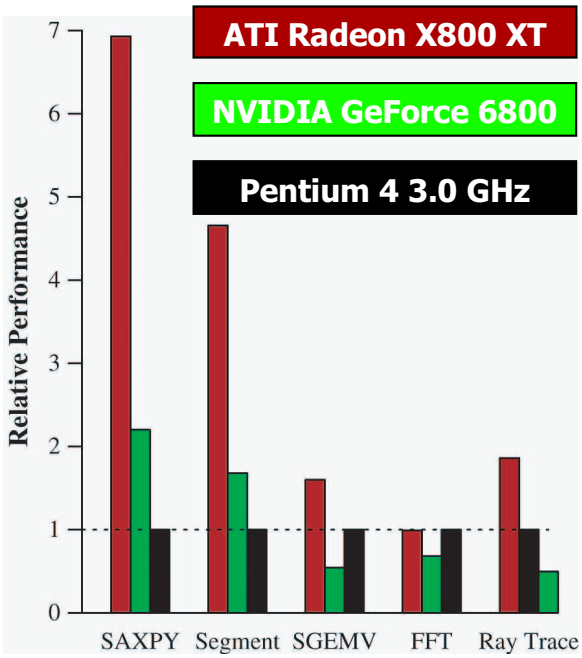


Figure 4.25: Performance results for benchmark applications.

progress. Last year we started with an implementation of the force calculation. Now we have implemented all the steps of the time stepping algorithm. The key advantage which makes this new version considerably faster compared to the previous one is that we now can perform several time steps entirely on the graphics card without having to transfer data back and forth between the CPU and the GPU. Therefore we are taking full advantage of the computing capabilities of the graphics hardware.

The rest of this section describes the changes that have been made to the GROMACS code and the results obtained so far.

Fig. 4.26 shows a flow chart of the main MD calculations in GROMACS. The most compute intensive part is the calculation of non-bonded forces between atoms. As a beginning, the GROMACS code for calculating the non-bonded forces between water molecules was ported from C to Brook. The coordinates are sent down to the GPU before the calculation and the forces are read back from the GPU afterwards (Fig. 4.27). This needs to be done for every time step. During benchmarks, it was found that considerable time was spent in reading back data from the GPU. In fact, this communication time was the bottleneck which decreased the performance on the GPU compared to the same code on the CPU.

In order to avoid bringing data back to the CPU every time step, we decided to implement the whole MD main loop on the GPU, except for the neighbor search step (see Fig. 4.28). Since the neighbor search step is usually performed only once every 10 steps, the communication bottleneck will effectively be eliminated. To do this, it is necessary to port the time stepping code (Verlet Integrator), the constraints code (SHAKE and SETTLE), handling of periodic boundary conditions (PBC), bonded forces (angles and dihedrals) and temperature and pressure coupling code to Brook.

As of this writing, porting of the integrator, PBC, SHAKE and SETTLE has been completed and tested. We are now able to simulate water in the microcanonical ensemble using this scheme and code optimization

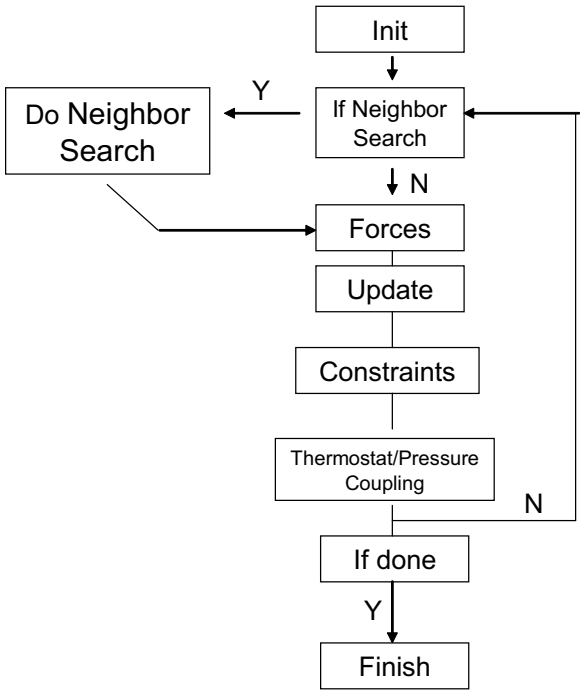


Figure 4.26: Schematic of the main MD loop of GROMACS.

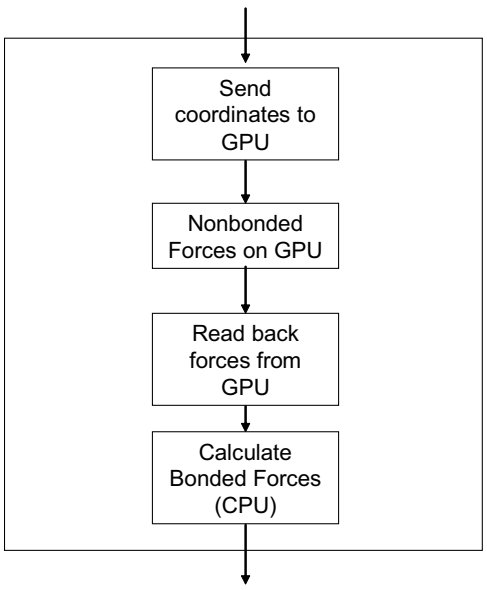


Figure 4.27: Modified force function for calculating only non-bonded forces on the GPU.

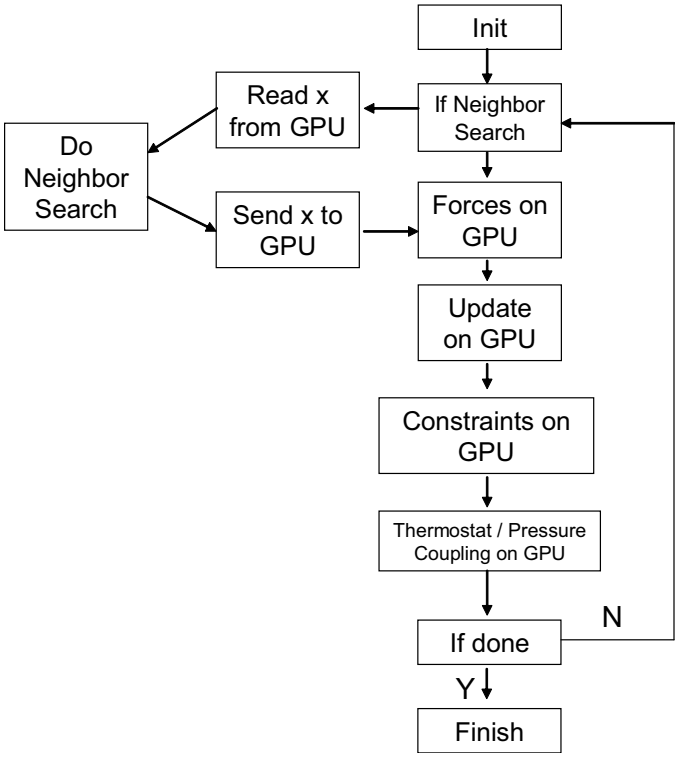


Figure 4.28: Target schematic for GROMACS on graphics hardware.

and benchmarking are under way. Once the bonded forces can be calculated on the GPU, we will be able to simulate systems containing both protein and water. We expect to see significant performance gains over the CPU when this is completed. The current status of the project is summarized on Figure 4.29.

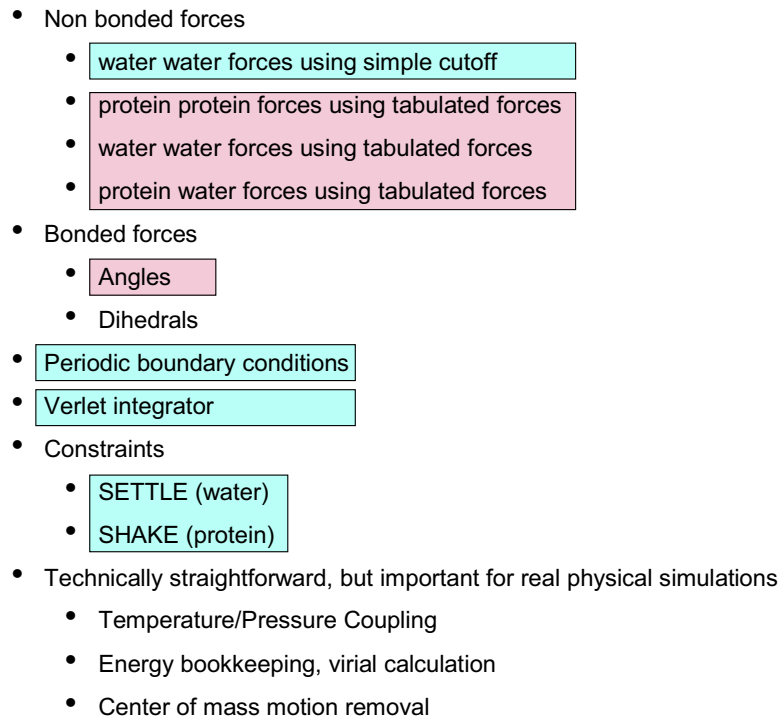


Figure 4.29: Current status of the Gromacs on GPU project. Highlighted in blue are the tasks which have been completed and tested. In brown are the tasks which have been completed but require further testing and validation. Tasks without colored boxes are still under development.

Shake Algorithm in Brook

To port Gromacs to GPUs, we had to modify many algorithms so that they can be run efficiently on streaming processors. We now describe an illustrative example which is the shake algorithm. It is a 2-stage algorithm used widely in molecular simulation. It is used to impose on the system constraints, such as bond length, which enable choosing larger time steps.

In this algorithm, the Verlet integrator is first used to calculate the motion of the atoms in the system assuming a complete absence of the rigid bond forces. The positions of the atoms at the end of this stage do not conserve the distance constraint required by the rigid bond and a correction is necessary. Second, the deviation in the length of a given rigid bond is used retrospectively to compute the constraint force needed to conserve bond length. See Figure 4.30. The constraint force can be shown to be

$$\boldsymbol{G}_{ij} \approx \frac{\mu_{ij}(d_{ij}^2 - |\boldsymbol{d}'_{ij}|^2)}{2\Delta t^2 \boldsymbol{d}_{ij}^0 \cdot \boldsymbol{d}'_{ij}} \, \boldsymbol{d}_{ij}^0$$

where $\boldsymbol{G}_{ij}$ is the constraint force, μ_{ij} is the reduced mass of the two atoms connected by the bond, $\boldsymbol{d}_{ij}^0$ and $\boldsymbol{d}_{ij}'$ are the original and new bond vectors, d_{ij} is the constrained bond length and Δt is the Verlet integration time step. The shake algorithm calculates the constraint force $\boldsymbol{G}_{ij} = -\boldsymbol{G}_{ji}$ that conserves the bond length d_{ij} between the i and j atoms.

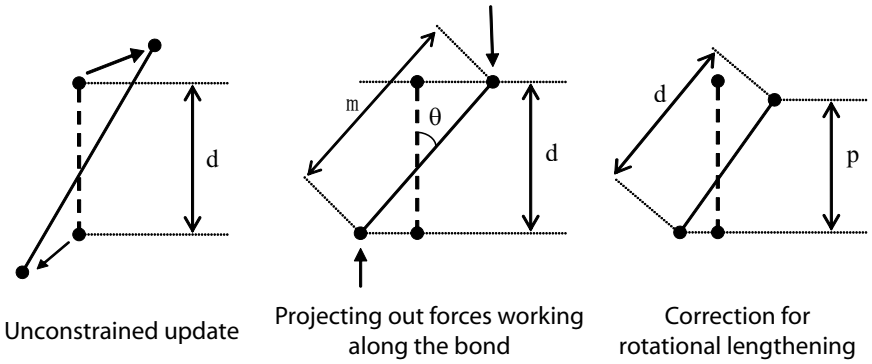


Figure 4.30: The three position updates needed for one time step. The dashed line is the old bond of length d, the solid line is the new bond.

The shake algorithm works on a set of coordinates, updating them sequentially as they participate in bonds. In this original form, however, the updates cannot be made parallel. To parallelize, the set of bonds (connectivity between atoms) were separated out into multiple interaction lists, such that each atom participates in only one interaction per list. This makes the interaction lists essentially decoupled. The main advantage of this approach is that, due to the decoupling of bonds, every atom is updated only once per list and therefore the algorithm becomes parallelizable. It may seem, at first, that this process is tedious or that we could end up with a high number of lists with very few bonds, thereby making it infeasible. It turns out that, using this approach, the maximum number of lists that we need is bounded by the number of bonds that are possible at every atom. This has an upper bound of 4 per atom (for example carbon) and therefore we will need a maximum of 4 lists. Since the molecular structure is fixed, this separation into different lists needs to be done only once. Figure 4.31 illustrates this process.

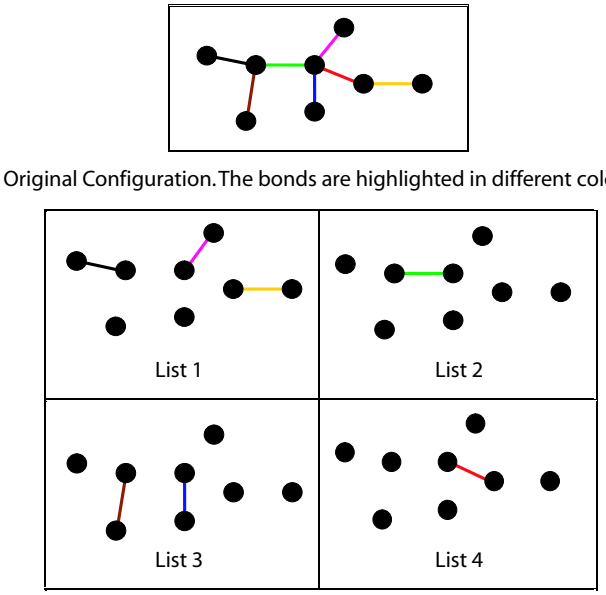


Figure 4.31: The original list of bond lengths to be constrained is split into multiple lists which can be processed in parallel by streaming processors.

This algorithm has been implemented and tested on the CPUs. Currently we are in the process of testing and benchmarking on the GPU. The integration with the remaining GROMACS code is complete. The brook version works well and provides accurate results of up to 4 places after the decimal (5 significant digits). We expect a significant speedup once the testing on GPU is complete.

The inner loops for water-protein and protein-protein are also being ported into brook. These inner loops are similar to the water-water force calculation completed last year. Development effort for these inner loops is now complete and the testing and integration is in progress.

4.7.5 Finite Element method for solid mechanics

Up to now we have focused on applications in fluid dynamics and molecular modeling. Those applications have very different computing requirements. In fluid dynamics, finite difference methods, for example, lead to problems with small connectivity. Typically each mesh node is connected to 6 other nodes. The arithmetic intensity vary but is usually relatively low. Molecular modeling codes offer different characteristics with huge connectivity between nodes, which now correspond to atoms. Typically atoms interact with all the atoms within a sphere of 7 Angstrom which may include up to 50 atoms. The arithmetic intensity was observed to be low to medium.

Another class of applications of great importance to DoE and NNSA is computational mechanics and in particular the simulation of materials undergoing impulsive loading such as in the case of detonations and impacts (see Figure 4.32). Those applications are often based on a finite element discretization. Contrary to the other applications, the number of data per control volume in the mesh can be quite large. Each control volume contains several Gauss points at each of which we need to compute the deformation gradient (9 components), the stress (9 components) and the tangent to the stress (81 components). From those

quantities the response of the material can then be computed. This application example is quite different from the fluid dynamics applications where typically the number of unknowns at each node is between 6 and 10 or from molecular dynamics where we only need to store the Cartesian coordinates of an atom. In order to demonstrate the streaming technology on GPUs, we considered the case of assembling the finite-element matrix for a neo-hookean material which belongs to the class of hyperelastic materials.

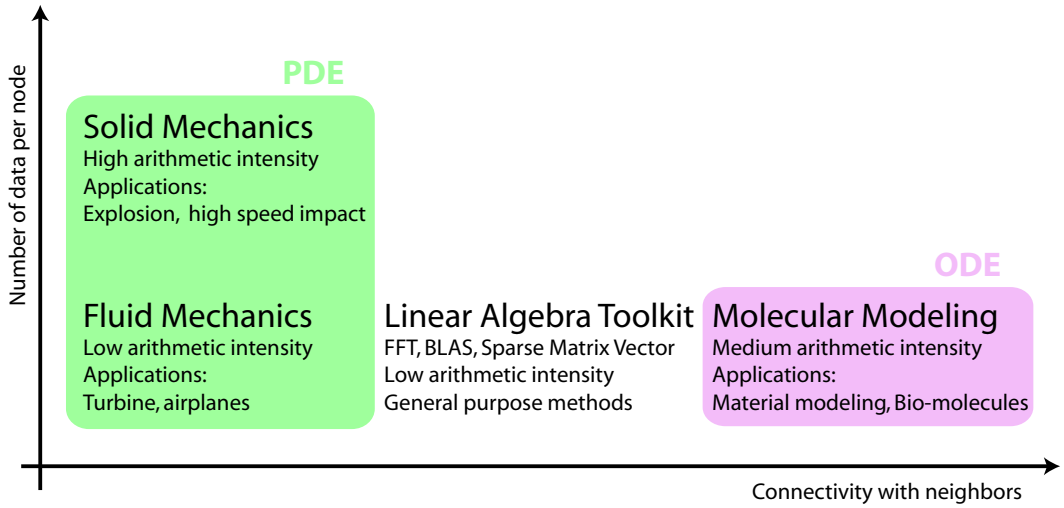


Figure 4.32: Map of various applications implemented so far on Merrimac and GPUs.

The results shown on Figure 4.33 for various problem sizes show that GPUs can outperform CPUs by a factor of more than 14. The overhead in using the graphics card is also visible for small size problems where the ATI performance curve asymptote to around two seconds. The cross over point is at 620 elements. Of course any practical problem will have a very large number of elements, up to millions of elements. Due to the limitations of GPUs in terms of memory (currently up to 512M) this type of calculation will have to be performed using a domain decomposition approach.

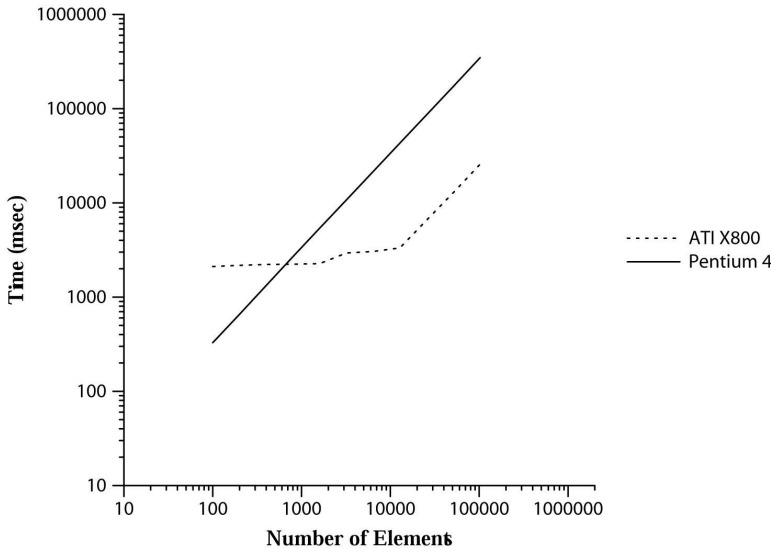


Figure 4.33: GPU performance result for finite element matrix assembly in computational mechanics.

Bibliography

[1] Advanced Micro Devices, Inc., One AMD Place, P.O. Box 3453, Sunnyvale, California, USA. *3DNow! Technology Manual*, March 2000. Order number 21928G/0.

[2] Jung Ho Ahn, Mattan Erez, and William J. Dally. Scatter-add in data parallel architectures. In *Proceedings of the Ninth Symposium on High Performance Computer Architecture*, February 2004.

[3] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonté, J-H A., N. Jayasena, U. J. Kapasi, A. Das, J. Gumm araju, and I. Buck. Merrimac: Supercomputing with streams. In *Supercomputing*, Phoenix, Arizona, November 2003.

[4] Mattan Erez, Jung Ho Ahn, Ankit Garg, William J. Dally, and Eric Darve. Analysis and Performance Results of a Molecular Modeling Application on Merrimac. In *SC'04*, Pittsburgh, Pennsylvaniva, November 2004.

[5] Merrimac Group. Merrimac specifications document. Technical report, CVA Group, Stanford University, 2004.

[6] Nuwan Jayasena, Mattan Erez, Jung Ho Ahn, and William J. Dally. Stream register files with indexed access. In *Proceedings of the Tenth International Symposium on High Performance Computer Architecture*, Madrid, Spain, February 2004.

[7] U.J. Kapasi, W.J. Dally, S. Rixner, P. Mattson, J.D. Owens, and B. Khailany. Efficient conditional operations for data-parallel architectures. *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, pages 159–170, December 2000.

[8] Ujval J. Kapasi, William J. Dally, Scott Rixner, Peter R. Mattson, John D. Owens, and Brucek Khailany. Efficient Conditional Operations for Data-parallel Architectures. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 159–170, December 2000.

[9] Ujval J. Kapasi, Scott Rixner, William J. Dally, Brucek Khailany, Jung Ho Ahn, Peter Mattson, and John D. Owens. Programmable stream processors. *IEEE Computer*, August 2003.

[10] A. Lim. *Improving Parallelism and Data Locality with Affine Partitioning*. PhD thesis, Stanford University, 2001.

[11] Motorola, Inc. *AltiVec Technology Programming Interface Manual*. Motorola, Inc, 1999.

[12] Shreekant (Ticky) Thakkar and Tom Huff. The Internet Streaming SIMD Extensions. *Intel Technology Journal*, (Q2):8, May 1999.

[13] David van der Spoel, Aldert R. van Buuren, Emile Apol, Pieter J. Meulenhoff, D. Peter Tieleman, Alfons L. T. M. Sijbers, Berk Hess, K. Anton Feenstra, Erik Lindahl, Rudi van Drunen, and Herman J. C. Berendsen. *Gromacs User Manual version 3.1*. Nijenborgh 4, 9747 AG Groningen, The Netherlands. Internet: http://www.gromacs.org, 2001.

[14] M. E. Wolf. *Improving Locality and Parallelism in Nested Loops*. PhD thesis, Stanford University, 1992.

CITS Roster (2003-2004)

Stanford Faculty:

- Alonso, Juan (T[a], I)
- Dally, William (M)
- Darve, Eric (M)
- Durbin, Paul (T)
- Fedkiw, John (M)
- Hanrahan, Pat (M)
- Jameson, Anthony (T)
- Lele, Sanjiva (T)
- Moin, Parviz (C)
- Pitsch, Heinz (C, I)
- Rosenblum, Mendel (M)

Research Staff:

- Apte, Sourabh (C)
- Binu, Mathew (M)
- Ham, Frank (C)
- Herrmann, Marcus (C)
- Iaccarino, Gianluca (C,T)
- Kalitzin, Georgi (T)
- Kassinos, Stavros (T)
- Kim, Sangho (I)
- Raman, Venkatramanan (C)
- Schluter, Jorg (I)
- Van der Weide, Edwin (T)
- Wu, Xiaohua (C,I)

External Affiliates:

- Davis, Roger (UC Davis)
- Mahesh, Krishnan (Univ. Minnesota)
- Wray, Alan (NASA)
- Barth, Tim (NASA)

Stanford Students:

- Ahn, Jung (M)
- Blanquart, Guillaume (C)
- Buck, Ian (M)
- Das, Abhishek (M)
- Desjardins, Olivier (C)
- Erez, Mattan (M)
- Gopinath, Arathi (T)
- Gummaraju, Jayanth (M)
- Gupta, Amit Kumar (M)
- Hartke, Paul (M)
- Hosseini, Kaveh (T)
- Ihme, Matthias (C)
- Iourokina, Ioulia (T)
- Jayasena, Nuwan (M)
- Ji, Minsuk (T)u
- Kang, Seongwon (C)
- Kapasi, Ujval (M)
- Knight, Tim (M)
- Kunz, Robert (M)

[a]Legend:
C: Combustor Group
I: Integration Group
M: Merrimac Group
T: Turbomachinery Group

Stanford Students (cont'd):

- Labonte, Francois (M)
- Lantz, Robert (M)
- Laskowski, Greg (T)
- Lee, Ki Hwan (T)
- Losasso, Frank (M)
- Nagaranjan, Santhanam (T)
- Rangasayee, Vidya (M)
- Sahu, Pradipta (T)
- Schaffer-Black, David (M)
- Shunn, Lee (C)
- Tamer, Zaki (T)
- Vaidyanathan, Vishal (M)
- Wall, Cliff (C)