Scaling compressible flow solvers on the IBM Blue Gene/Q platform on up to 1.97 million cores

By I. Bermejo-Moreno, J. Bodart AND J. Larsson

1. Motivation and objectives

Progress in the numerical simulation of turbulent flows has always been tied to leading-edge advancements in high-performance computing technologies, both by benefiting from and pushing forward such technological developments. The driving parameter in turbulent flows is the Reynolds number, Re, which measures the relative importance of inertial (non-linear) forces over viscous (dissipative) forces acting on the fluid. At sufficiently high Re, which occurs in most natural and engineering flows, a spectral gap between the energy-containing and dissipation length scales of the flow exists. The resulting multi-scale nature often renders the turbulence problem intractable even for the simplest flow configurations due to the large number of degrees of freedom required to solve all (as in Direct Numerical Simulations, DNS) or the most relevant (as in Large-Eddy Simulations, LES) scales of the flow. Only by utilizing the most powerful supercomputers and developing better numerical methods can this spectral gap of length scales be increased towards more reliable predictive simulations of turbulent flows of practical interest.

Over the past year, the list of the top ten most powerful computer systems in the world has seen the emergence of several new platforms, all of them capable of peak performances in the order of petaFLOPS—i.e., $\mathcal{O}(10^{15})$ floating-point operations per second. One of these new architectures is the IBM Blue Gene/Q (BG/Q). At the time of writing this brief, four spots in that top-ten ranking of supercomputers are occupied by BG/Q clusters, two of which (nicknamed *Sequoia* and *Vulcan*) are deployed at the Lawrence Livermore National Laboratory (LLNL). Early access to the *Sequoia* and *Vulcan* clusters was granted for a short testing phase in the beginning of 2013 to assess the massively parallel scaling and performance capabilities of scientific codes developed under the Predictive Science Academic Alliance Program (PSAAP) funded by the U.S. Department of Energy. The PSAAP center at Stanford University focused on the predictive simulation of multi-physics flow phenomena inside scramjet engines for application to hypersonic flight, relying heavily on high-fidelity (DNS and LES) compressible flow solvers.

In this brief we present scaling and performance tests on the BG/Q platform of two such compressible flow solvers, *Hybrid* and *CharLES*^{x}, developed at the Center for Turbulence Research. These solvers are representative of two complementary numerical techniques for fluid flow simulations (and for solving partial differential equations, in general) and target a broad range of scientific and engineering applications regarding turbulent flows. *Hybrid* is a structured-mesh, finite-difference solver aimed at the direct numerical simulation of fundamental, canonical physical problems in simple geometrical configurations by using high-order numerical schemes; *CharLES*^{x} is an unstructured-mesh, finite-volume solver that trades high-order accuracy (typically limited to second order) for versatility in handling arbitrarily complex geometries and multi-physics modeling. A more detailed description of the results presented in this brief for the *Hybrid* code can be found in Bermejo-Moreno *et al.* (2013*a*).

Both codes have been previously deployed in several existing supercomputing architectures, showing good levels of parallel scalability. The main objective of this work is to assess how far the existing parallelization paradigms used by the two solvers can be pushed in this new BG/Q architecture with minimal modifications to the codes, rather than to perform an intensive code optimization targeted to this particular platform, that could compromise portability to future architectures. After introducing the BG/Q system and the performance tools used in Section 2, results are first presented for the *Hybrid* code in Section 3, followed by the *CharLES*^{x} code in Section 4. Concluding remarks and suggested future work are discussed in Section 5.

2. Testing environment: hardware and performance tools

BG/Q is based on a system-on-a-chip (SoC) building block (Haring *et al.* 2012), each consisting of eighteen 64-bit PowerPC A2 processor cores, sixteen of which are dedicated to computing, one serving the operating system and the remaining one reserved for redundancy. Two execution units per core are available: XU for integer, load and store operations; AXU for floating-point operations. At most, one instruction can be completed per cycle and thread. Two instructions can be completed per cycle per core, one from each of the two execution units. The clock frequency of each core is 1.6 GHz. A SIMD (single-instruction-multiple-data) quad-vector double precision floating point unit (QPX) is included per core. 4-way simultaneous hardware hyperthreading is available, bringing the peak performance per core to 12.8 GFLOPS. Memory-wise, each core includes a 16 KiB L1 D-cache with a 6-cycle latency, and a 4 KiB prefetch buffer (L1P), with a 24-cycle latency (see Chung *et al.* 2012). Each SoC integrates also a 800 MHz, 32 MiB eDRAM L2 cache, with an 84-cycle latency and connects all the cores with a full crossbar switch.

A 5D torus network provides communication with other chips through 2 GB/s links. Each node includes a chip and 16 GiB of DDR3 DRAM (1 GiB per core) with 346-cycle latency. A compute rack contains 1,024 compute nodes, 16,384 cores and 16 TiB RAM. The lightweight Compute Node Kernel (CNK) operating system is installed on the compute nodes, whereas dedicated Input/Output (I/O) nodes run Red Hat Enterprise Linux, handling the file system operations in a proportion of one I/O node per 128 compute nodes. The IBM XL compilers were used for all the tests reported in this study. *Sequoia* contains 96 racks (1,572,864 cores), whereas *Vulcan* includes 24 racks (393,216 cores). During the early access phase in which the study presented in this brief was performed, the *Sequoia* and *Vulcan* clusters were occasionally inter-connected to perform scalability tests on a 120-rack configuration, resulting in an unprecedented core count of 1,966,080, and a combined theoretical peak performance of 25 PFLOPS. In addition to the *Sequoia* and *Vulcan* clusters at LLNL, the *Vesta* cluster at the Argonne Leadership Computing Facility (ALCF) of the Argonne National Laboratory (ANL) was also utilized. This cluster, which is intended for development and testing, has 2 racks (32,768 cores).

Hiding the high L2-cache latency of BG/Q (84 cycles compared with the nearly 10 cycles found in other popular CPUs) is paramount to effectively increase the performance of memory-bound applications such as the *Hybrid* and *CharLES*^{x} codes in this platform and can, in principle, be achieved by means of hyperthreading, which allows multiple instruction streams (threads) to execute simultaneously per core: when one thread is idle (for example, waiting for data to be retrieved from cache/memory), the other thread(s) can still be executing instructions (at the maximum rate of two instructions per cycle,

one per execution unit, in each PowerPC A2 core of BG/Q, which allows up to four threads).

To measure performance and scalability in this study, the codes are instrumented with MPI timing functions which impose a minimal overhead allowing elapsed wall-clock time spent in different parts of the program execution to be measured. In this brief we focus on the global computation and MPI-communication times that dominate our applications. Averages over the number of time steps of execution are considered. The resolution on BG/Q of the MPI timing function is one nanosecond (several orders of magnitude smaller than the times being measured). In addition, the IBM High Performance Computing Toolkit (IHPCT) is also utilized, as it provides MPI timing, profiling information, and data from hardware counters (including derived performance measurements, such as FLOPS) through the Hardware Performance Monitor (HPM) library (Gilge 2013).

3. The Hybrid flow solver

The *Hybrid* code solves the compressible Navier-Stokes equations for a perfect gas on structured, Cartesian, stretched grids for the conserved variables of density, momentum, and total energy. The code was originally developed to study the fundamental canonical shock/turbulence interaction problem through a series of DNS on up to $2.5{\cdot}10^9$ grid points, achieving a Reynolds number of the incoming turbulence based on the Taylor microscale of $Re_\lambda \approx 70$ (see Larsson & Lele 2009; Larsson *et al.* 2013, for details). A solution-adaptive strategy that builds upon the work of Adams & Shariff (1996) and Pirozzoli (2002) is utilized to balance the opposite numerical requirements involved in resolving the turbulence (through low-dissipative schemes) and capturing shock waves while minimizing aliasing and numerical instabilities (by introducing extra numerical dissipation). This solution-adaptivity incorporates a split form of the convective terms that reduces the aliasing errors, allowing for stable computations. The stability of the scheme-switching was analyzed and proven in Larsson & Gustafsson (2008), and the effect of the shock-capturing errors on the computed turbulence was analyzed in Larsson (2010). The most common configuration of the *Hybrid* code is to use a 6th-order accurate central-difference scheme in most of the domain, and a 5th-order accurate weighted essentially non-oscillatory (WENO) scheme across all discontinuities (e.g., shock waves), marked by a sensor based on local dilatation and enstrophy. The accuracy of these schemes to solve the physics of shocks and turbulence was thoroughly assessed in Johnsen *et al.* (2010). Favored by the incompletely hyperbolic nature of the system of partial differential equations (PDEs) to be solved and by the already small time steps required to retain high accuracy, a fourth-order, four-stage Runge-Kutta explicit time integration is used.

The computational domain is decomposed at initialization into Cartesian blocks according to the number of executed MPI tasks, aiming to ensure load balance and minimize the MPI communication time by reducing the number of grid points in the block interfaces. At each stage of the time-stepping algorithm, the contribution of the convective terms (inviscid fluxes) is first calculated, followed by computation of viscous fluxes. Primitive variables (velocity, pressure, and temperature) and velocity gradients are then computed from the updated conserved quantities, followed by calculation of the shock sensor at each grid point to mark regions near shock waves (discontinuities) where the WENO scheme with Roe flux splitting will be used for the approximation of the inviscid fluxes in the subsequent time stepping stage.

The parallel communication is minimized by the use of a spatially explicit difference

operator. The code is written in C++ and uses MPI for parallelization (including all major I/O). Double-precision arithmetic is used, with a memory footprint in each block of 42 doubles per grid point, including ghost (halo) points. Exchange of halo-data is done through non-blocking and asynchronous communications with concurrent computation to hide the communication cost. As an example, primitive variables are computed from the conserved variables (updated in the Runge-Kutta time-stepping) first in a layer of 3 grid points adjacent to the block boundaries (for the 6th-order scheme used here). After this, the communication of both conserved and primitive variables is initiated (through a non-blocking `MPI_Isend`/`MPI_Irecv` pair). The primitive variables are then computed in all remaining grid points before the communication is verified to be completed. Similarly, the communication of the velocity gradients starts at the beginning of each stage of the Runge-Kutta time-stepping, before the calculation of inviscid fluxes, and ends before the computation of viscous fluxes, for which they are required.

In the scaling/performance tests that follow, we consider the 6th-order central-difference and the 5th-order WENO schemes independently, which can be useful for the determination of load-balancing strategies, since these numerical schemes have a rather different computational cost per grid point, whereas the communication cost is the same. Periodic boundary conditions are used and no shocks are present in the flow to ensure stability when using only the central-difference scheme.

3.1. *Weak scaling results*

In a weak scaling test, the time to solution is measured as the number of processors (cores) is increased, while keeping a constant problem size per processor (thus increasing the total problem size). Ideal scaling is obtained if the time to solution remains constant for all core counts.

3.1.1. *Maximum memory load with no hyperthreading*

The block size chosen for this weak scaling test is 128^3 points per core. The memory footprint of the *Hybrid* code for such a block size is approximately 830 MB. Note that the operating system and MPI overhead consume a fraction (generally increasing with the number of cores in use) of the remaining memory. Keeping this computational load per core constant, the number of cores is increased in successive runs from 32 to the 1,966,080 available when the *Sequoia* and *Vulcan* clusters were inter-connected. Thirty two cores correspond to the minimal core count involving inter-node communications. No optimized mapping of the physical torus network to the structured domain decomposition was utilized in these runs. The largest case, run on 120 racks, includes over 4.1 trillion points (20.5 trillion degrees of freedom), and a memory footprint surpassing 1.6 PB. All cases in this first test are run with the 6th-order central-difference numerical scheme.

Figure 1(a) shows the weak scaling results for this maximum-memory-load case. The horizontal axis represents the number of cores, in logarithmic scale. The left vertical axis represents the metric t_w (in microseconds) defined as a normalized time $t_w = \Delta^{wt} N_c/N_p$, where Δ^{wt} is the average wall-clock time needed to perform one time step, N_c and N_p being the number of cores used and the number of grid points (or control volumes), respectively. The total and communication times are plotted in solid circles and hollow squares, respectively, showing results averaged over the last five time steps of each run (averaging the last 50 time steps instead resulted in a difference of less than 1%). The weak scaling efficiency is defined as t_w^o/t_w, where t_w^o is the normalized time at the lowest core count considered when no hyperthreading is utilized.

The weak scaling for this case is nearly ideal up to 16,384 cores (1 rack). From 16,384

128^3 grid points per core 96^3 grid points per core

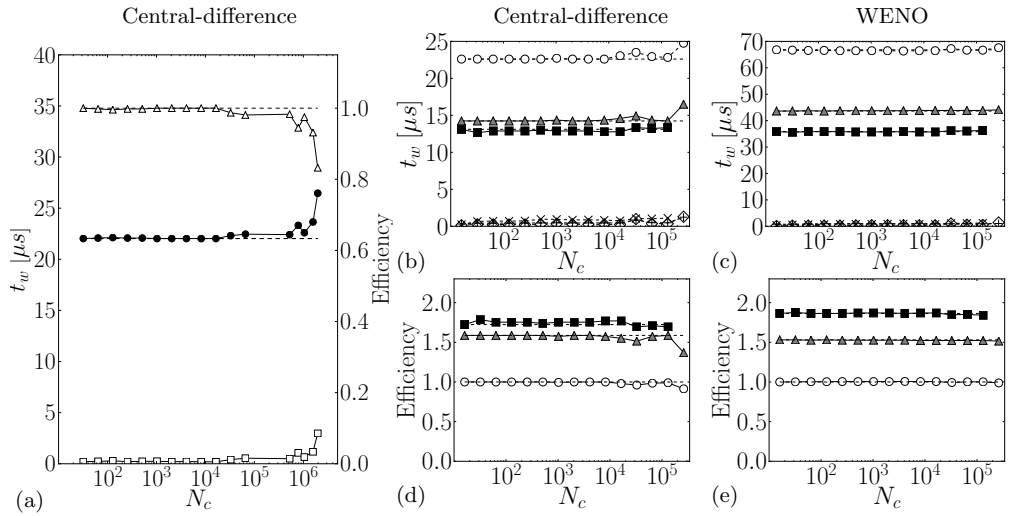

FIGURE 1. (a) Weak scaling results on the combined *Sequoia* and *Vulcan* BG/Q clusters, for a unit block size of 128^3 grid points per core, increasing the core-count (N_c) from 32 to 1,966,080: circles, total time; squares, communication time; triangles, efficiency (right ordinate axis). (b-e) Effect of hyperthreading on weak scaling results for 6th-order central-difference (b,d) and 5th-order WENO (c,e) on the *Vulcan* BG/Q cluster, for a unit block size of 96^3 grid points per core, increasing the core-count from 32 to 131,072: circles, no hyperthreading; triangles, 2 MPI tasks per core; squares, 4 MPI tasks per core. (b,d): average wall-clock time per time step divided by the number of grid points per core, t_w, with the contribution of MPI communications in diamonds, pluses and crosses, for 1, 2 and 4 MPI tasks per core, respectively. (d,e): weak scaling efficiency. Black dashed lines are drawn for reference to indicate the value achieved for the lower core-count at each specific configuration of numerical scheme and hyperthreading.

to 524,288 (32 racks) the percentage of time spent in MPI communications increases, and the efficiency decreases to 0.98. At 1,048,576 cores (48 racks) the efficiency drops to 0.94, recovering to 0.97 for 64 racks. This drop in efficiency was replicated in different tests performed at that scale and is probably the result of suboptimal communications associated with the block decomposition and MPI mapping layout for that case. For the full *Sequoia* cluster (96 racks and 1,572,864 cores) the efficiency is 0.93, whereas adding the 24 additional racks of *Vulcan* brings the efficiency to 0.83. We note that the interconnection of both clusters was experimental: normal operation of the clusters during the early access period in which these runs were performed, as well as the planned production runs, did not include the connection of *Sequoia* and *Vulcan*. Nevertheless, repeatability of all these runs was ensured by running on different occasions for which the two clusters were interconnected. The block decomposition resulting for this last case and the lack of an optimized MPI mapping layout to the torus network also likely contribute to that drop in scaling efficiency.

3.1.2. *Hyperthreading and numerical scheme*

The effects of hyperthreading on each numerical scheme (6th-order central-difference and 5th-order WENO) are evaluated next. The memory overhead imposed by the use of hyperthreading in these runs required lowering the memory footprint of the code by

reducing the unit block size to 96^3 grid points per core. The number of cores was varied from 32 to 131,072.

Results are shown in Figure 1(b-e). Note that WENO takes nearly three times longer on average than central-difference. The time spent in communications is the same for both schemes, since the number of ghost points is kept constant in all runs. As a consequence, WENO shows an improved parallel scaling. Hyperthreading provides a substantial improvement of the efficiency, resulting in speedups larger than 1.5, sustained when the number of cores is increased. The gain is more noticeable when changing from a single task per core (no hyperthreading) to two, but further increasing from two tasks per core to four still results in a noticeable gain, especially for the WENO scheme.

3.2. *Strong scaling*

In a strong scaling test, the total problem size is kept constant as the number of cores, N_c, increases. Ideal scaling is obtained if the speedup, $N_c^o t_w^o / t_w$, increases as the number of processors. The strong scaling efficiency is defined as $N_c^o t_w^o / N_c t_w$. N_c^o is the minimum core count in the test (typically limited by the minimum memory requirements for the problem size of choice).

Figure 2(a) shows the parallel speedup for a strong scaling test on a problem size of $8{,}192^3$ (approximately 550 billion) grid points. The memory footprint of the code for this grid size is approximately 185 TB. For such memory usage, the starting point of the strong scaling study corresponds to 16 racks (262,144 cores), for which the memory usage per core is approximately 707 MB. Successive runs are performed increasing the number of cores while keeping the problem size constant, until the full capacity of the *Sequoia* and *Vulcan* clusters combined is reached. Runs with one and two tasks-per-node are done at each core count to evaluate the effect of hyperthreading, except for the case with 1,048,576 cores, for which no hyperthreaded data could be retrieved during the time the clusters were available for this study. Only the central-difference scheme is used in this first strong scaling test. Good strong scaling for this case is observed, with substantial speedups achieved by means of hyperthreading.

Figure 2(b,e) shows additional strong scaling studies for a problem size of $2{,}048^3$ grid points, and increasing the number of cores from 4,096 to 131,072, comparing WENO and central-difference numerical schemes and considering the two available levels of hyperthreading (2 and 4 MPI tasks per node). Speedup and parallel efficiency results are consistent with our previous findings for both weak and strong scaling studies. Parallel efficiency is nearly linear for the WENO numerical scheme, as a result of an increased computational cost while keeping the communication time nearly identical to the central-difference scheme. The improvement provided by hyperthreading is slightly higher for the WENO scheme, and can be sustained for a larger core count, compared to the central-difference scheme. The largest gains in this range of core numbers are provided by using 4 MPI tasks per core, although the relative increase is consistently higher for 2 MPI tasks per node, as already noticed in the previous tests. For the minimum core count considered in these runs (4,096), 4-way hyperthreading using MPI was not possible due to the memory footprint introduced by MPI.

3.3. *Performance*

Performance tests were run on two racks (32,768 cores) of the *Vulcan* cluster, to involve inter-rack communications, and for two grid sizes ($2{,}048^3$ and $3{,}072^3$ points). The results are summarized in Table 1. The higher algorithmic complexity of the 5th-order WENO scheme results in a larger average number of instructions issued per cycle per

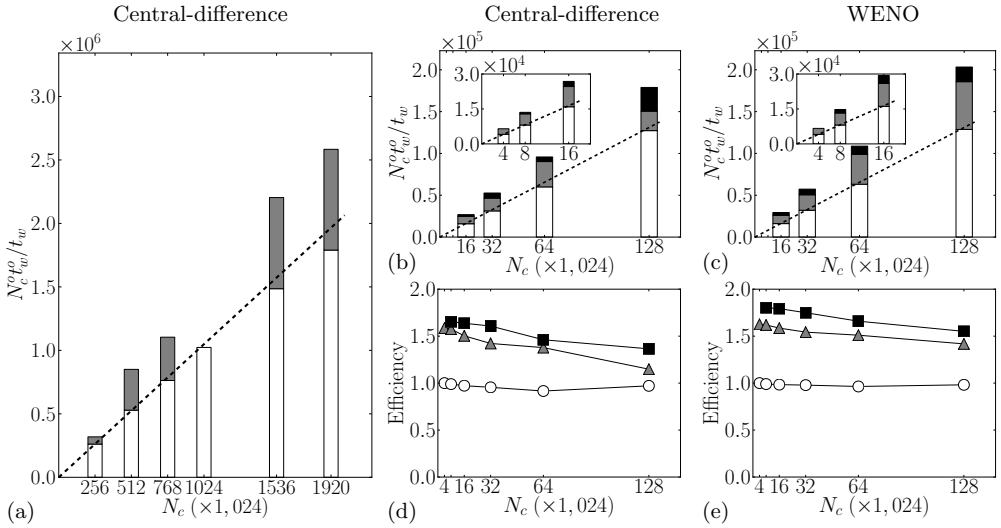

FIGURE 2. (a) Strong scaling results for a problem size of $8{,}192^3$ grid points, increasing the core-count from 262,144 to 1,966,080. (b-e) Effect of hyperthreading on strong scaling results for 6th-order central-difference (b,d) and 5th-order WENO (c,e) on a problem size of $2{,}048^3$ grid points, increasing the core-count from 4,096 to 131,072. (b,c) show the speedup (the inset zooms in the lower core-count range); (d,e) show the strong scaling efficiency. White bars: 1 MPI task per core; gray bars: 2 MPI tasks per node; black bars: 4 MPI tasks per node. Dashed black lines correspond to the linear scaling considering one task per core. N is the number of nodes. $N_c = 16N$ is the core-count.

core (IPCPC) in comparison with the 6th-order central-difference scheme. Independently from the grid size and for both numerical schemes, 2- and 4-way hyperthreading typically result in gain factors in the number of IPCPC of approximately 1.6 and 2, respectively, over cases with no hyperthreading. This is consistent with the ability of the hardware to issue two instructions per cycle instead of one, when more than one hardware thread is involved per core (see Section 2). The percentage of floating-point operations (F) is lower than 50% of the total in all cases, as expected for this kind of memory-bound application. F is consistently 9-15% higher for WENO runs, as a result of its higher algorithmic complexity. A larger grid size, for a constant core-count, increases F, whereas hyperthreading decreases it, more significantly for smaller grid sizes.

Regarding memory access, cache hits are shared as follows: 88% occur at L1 while about 10% at L2. DDR traffic is relatively high for the central scheme with 8.1 bytes per cycle (the maximum bandwidth is approximately 18 bytes per cycle on BG/Q) and largely decreases (to approximately 4) for the WENO scheme. This reduction is directly related with an equivalent memory access combined with a longer time-to-solution for the WENO scheme. The same effect is observed when hyperthreading is used, as it increases the memory traffic. Note that little use of the L1 prefetch is found by the compiler (1.6% or less). Hardware counters confirmed the high impact of L2 cache latency for our application. However, hyperthreading appears as an efficient way to partially hide memory latency for this application.

The global sustained levels of performance (%P) shown in Table 1 refer to the ideal 419.4 TFLOPS that can be delivered by the two BG/Q racks considered in these runs. On

N	T	IPCPC		F		DTC		%P	
		C6	W5	C6	W5	C6	W5	C6	W5
2048^3	1	0.33	0.42	33	42	4.3	2.2	2.1	3.5
	2	0.52	0.67	28	40	6.8	3.9	2.9	5.3
	4	0.69	0.80	21	36	6.6	4.2	2.9	5.7
3072^3	1	0.32	0.41	34	43	4.8	2.4	2.2	3.5
	2	0.52	0.64	32	42	7.6	3.7	3.4	5.4
	4	0.65	0.81	28	40	8.1	4.4	3.6	6.4

TABLE 1. Performance results for 6th-order central-difference (C6) and 5th-order WENO (W5) schemes, on two racks (32,768 cores) of BG/Q. N: number of grid points; T: number of threads per core; IPCPC: instructions completed per cycle per core; F: percentage of floating-point instructions; DTC: DDR traffic (node average) (Bytes/cycle); %P: percentage of ideal peak performance.

average, WENO shows a performance 1.9 times higher than central-difference. Significant improvements are achieved by hyperthreading for both numerical schemes, particularly when the grid size of the problem is increased and, therefore, memory locality decreased. Two-way hyperthreading provides the largest relative gain.

3.4. *Profiling*

Profiling results for runs with each numerical scheme on two racks of the *Vulcan* cluster are shown in Table 2. Runs with the central-difference scheme are dominated (42%) by the calculation of viscous terms: 12.5% to compute the viscous fluxes in each of the transverse directions (y and z) and 9% in the x direction. The convection terms account for 36% of the time, of which 26% is spent in the computation of inviscid fluxes. The computation of the velocity gradients in y and z took slightly above 3.5% of the time, whereas the gradients in x required about 3%. This discrepancy among the execution times found for operations in different coordinate directions is a direct consequence of the privileged data locality found for the x coordinate direction, which corresponds to the fastest-varying index in the arrays containing the data in memory for all the points within an MPI block. Setup and rearrangement of non-blocking MPI halo-data exchange accounts for 7% of the total time, and the intrinsic array multiplications and additions in the Runge-Kutta time-stepping update take less than 5% of the execution.

In contrast, when the WENO scheme is used, the interpolation required to obtain inviscid fluxes, together with the Roe flux splitting, dominate the profiling results, accounting for about 60% of the execution time. The other elements described above when using the central-difference scheme remain in practically the same order, with their percentages of total time significantly brought down due to a smaller relative contribution.

3.5. *Time-to-solution*

For a DNS of canonical shock-turbulence interaction to show a spectral gap between energy-containing (L) and dissipation (η) scales, it is estimated that a Taylor Reynolds number $Re_\lambda \approx 190$ must be reached. Such Re_λ implies a fourfold increase in the ratio L/η

	C6 (%)	W5 (%)
Convection	**36** (26+10)	60
Diffusion	**42** (12.5+12.5+9+8)	26
Gradients computation	**10** (3.5 + 3.5 +3)	6
Data re-arranging	**7**	5
Runge-Kutta update	**5**	3

TABLE 2. Relative timing in advancement of one time step. Addition between brackets for the central scheme refers to the substep discussed in the text.

with respect to the current state of the art (Larsson *et al.* 2013), and would then require about $9{,}464{\times}4{,}096^2 = 160{\cdot}10^9$ grid points and 140,000 time steps. From the strong scaling study reported in Figure 2(a) for central-difference scheme on a grid of comparable size and considering a conservative 100% increase in the computation time resulting from the subobtimal load-balancing of central-difference and WENO schemes based on prior simulations, the estimated total time to perform this simulation on *Sequoia* would be $2{\times}140{,}000{\times}(9{,}464{\times}4{,}096^2){\times}16{\cdot}10^{-6}/3600{\approx}\ 200{\cdot}10^6$core hours, which would take about 126 wall hours (≈5 days) on the full cluster. Doubling the grid resolution to a number of $18,928 \times 8,192 \times 8,192 = 1.27$ trillion points, with a corresponding $Re_\lambda \approx 300$ of the incoming turbulence, would still allow the use of hyperthreading, carrying an associated cost of $3.16{\cdot}10^9$ core hours, feasible in about 84 days of full utilization of *Sequoia*.

4. The CharLES^x flow solver

CharLES^x stems from an early version of the *CharLES* code originally developed by Frank Ham at the Center for Turbulence Research. The code implements a new modular architecture into an object-oriented paradigm, adding extensions for new physical models (such as wall and combustion models, real fluid, impedance boundary conditions), I/O, and visualization capabilities. It uses a finite-volume formulation to discretize the spatially filtered compressible Navier-Stokes equations for the conserved variables of mass, momentum, and total energy on unstructured meshes, thus allowing the solver to handle complex geometries. The inherited mesh-based blend of centered and upwind numerical schemes ensures stability and robustness of the differencing operators (see Khalighi *et al.* 2011). Similarly to the *Hybrid* code discussed in Section 3 and to the original *CharLES* code, *CharLES*^{x} implements a solution-adaptive methodology that combines a low-dissipation centered numerical scheme and an essentially non-oscillatory (ENO) second-order shock-capturing scheme. The latter is applied only near shock waves, identified by a shock sensor which is activated according to the criterion: $-\partial u_k/\partial x_k > \max(\sqrt{\omega_j\omega_j}, 0.05c/\Delta)$, where $\partial u_k/\partial x_k$, $\omega_j\omega_j$ and c are the local dilatation, enstrophy, and sound speed, respectively, and Δ is the control-volume size. Dynamic Smagorinsky and Vreman subgrid-scale models are implemented for explicit LES. The discretized equations are advanced in time using a three-stage, third-order explicit Runge-Kutta algorithm. The code has been applied to solve a variety of complex multi-physics flows such as those present in high-lift devices (Bodart & Larsson 2011, 2012), traveling-wave thermoacoustic Stirling heat engines (Scalo *et al.* 2013*b*,*a*), rocket combustion with real-fluid effects (Hickey *et al.* 2013; Hickey & Ihme 2013), shock-waves/turbulent-boundary-layers interactions (Bermejo-Moreno *et al.* 2011; Vane *et al.*

2013), and scramjet propulsion (Larsson *et al.* 2012; Bermejo-Moreno *et al.* 2013*b*). The code is written in C++ and uses MPI and MPI-I/O for parallelization.

4.1. *Performance*

Prior to attempting any parallel scaling studies on BG/Q, several optimizations at the core level of execution were implemented in the code, while adhering to the primary design criterion of cross-platform portability. The purpose was to achieve a more comparable fraction of the machine peak performance than what is achieved on more forgiving architectures for memory-bound applications (i.e., with the decreased latencies, particularly for the L2-cache, found in BG/Q). Memory access at the core level was optimized and the general formulation for the unstructured mesh operators was particularized through fixed-size stencils specified at the pre-processing stage for hexahedral control volumes to explicitly allow further compiler-based optimizations such as loop unrolling.

Similarly to the *Hybrid* code, performance characterization and global profiling were attained by means of the GNU utilities *gprof* and *cprof*. The IBM HPM library was utilized to track hardware counters and evaluate peak performance. The optimizations added to the solver were shown to improve the overall efficiency by a total measured speedup factor of 2.3. This resulted in a sustained performance of about 4% of the machine nominal peak, slightly below but comparable to the *Hybrid* solver. These percentages of peak performance are commonly found in BG/Q for memory-bound applications that rely heavily on large sparse-matrix/vector multiplications (Walkup 2012), such as the kind of CFD solvers presented here. Note that popular benchmarks used in high-performance computing, such as LINPACK, are capable of achieving much higher percentages of FLOPS performance (74.9% of the peak on BG/Q, for example), due to their different algorithmic nature (e.g., solving dense linear systems in the case of LINPACK). For applications based on sparse-matrix/vector multiplications, modifications of the data structure have proven to greatly improve performance on both NUMA and BG/Q architectures (Hejazialhosseini *et al.* 2012; Rossinelli *et al.* 2013). Such optimizations of the data structures were not targeted in the present work due to the substantial implementation cost required and the short time available for this study on the *Sequoia* cluster, but they will be investigated in future work, keeping cross-platform portability as a priority.

4.2. *Partitioning strategy for large multi-core architectures*

The use of unstructured meshes on distributed parallel architectures relies on a domain decomposition of the problem done at the pre-processing stage. Classical approaches in CFD codes rely on external libraries such as ParMETIS (Karypis & Kumar 1995) to perform partitioning. Such libraries can create partitions of large domains in parallel and no particular adjustment is needed when using hundreds of cores. However, partitioning on large clusters involving hundreds of thousands of cores becomes non-trivial since: (i) increasing the number of parallel tasks to perform the partitioning can lead to lower quality of the partition, and (ii) the heterogeneity of the memory layout of the architecture has to be taken into account in order to maximize the usage of the available bandwidth by the flow solver at run time.

Regarding the parallelization, we note that the partitioning of a computational domain does not usually require a large number of processors compared with what is needed for the execution of the flow solver (the required core-count typically differs by one or two orders of magnitude for a cluster like *Sequoia* and the problem sizes targeted on scientific production runs with *CharLES*^{x}). Such partitioning libraries are not necessarily

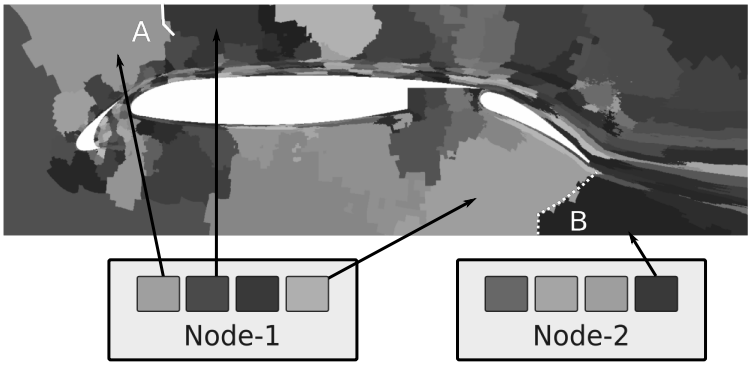

FIGURE 3. Partitioning over the McDonnell-Douglas 30P/30N multi-element airfoil on a multicore cluster architecture and associated subdomain physical location on the machine.

implemented with the possibility to run efficiently on million-core architectures and their execution on larger number of cores can lead to a lower partitioning quality. As a consequence, a common best-practice is to minimize the number of processors used to compute the partition for a given problem size as a pre-step to the flow solver execution.

The role of the memory-layout heterogeneity in the partitioning can be described using the sketch in Figure 3, where a global partitioning approach has been adopted. The discretized domain can be represented by its connectivity: vertices (control volumes) connected through edges (face of the control volumes). The partitioning problem is thus an "edge-cut" minimization that results in the lowest amount of inter-face communication between processors. To minimize the MPI communication needed during execution, the hierarchy of the memory layout of the system has to be taken into account. For a general multicore cluster architecture, intra-node communications are much faster than inter-node communications, resulting in a heterogeneous performance at the "ghost cell" exchange step. As depicted in Figure 3, subdomain boundaries can be associated with different network speeds, depending on whether neighboring subdomains were allocated on the same compute node or not. In the case of a torus network (e.g., five-dimensional on BG/Q), every dimension corresponds to a different network speed, and an optimized mapping should ideally consider this additional complexity. In *CharLES*^{x} we presently account for this heterogeneity in the memory layout by using a two-layer partitioning strategy, consisting of a first partitioning step done at the node level (inter-node) using ParMETIS followed by a second partitioning step at the core level (intra-node). The latter does not involve any inter-node communications and is achieved using the METIS serial graph partitioning library. By using this technique, locality is significantly improved among subdomains by ensuring that the edge-cut minimization is performed at the bottleneck level of the memory layout. The current two-layer partitioning approach can be extended to address any number of levels in the memory hierarchy of the system.

4.3. *Strong scaling on* Sequoia

In this work we consider the strong scaling of a problem involving 700 million control volumes. This is a modestly sized problem compared with what can be accomplished with the total number of cores ($1.57 \cdot 10^6$) available on the *Sequoia* system, in terms of memory capacity (see Section 3.1.1). However, such mesh size is representative of the scientific problems currently targeted with the *CharLES*^{x} code and takes into account

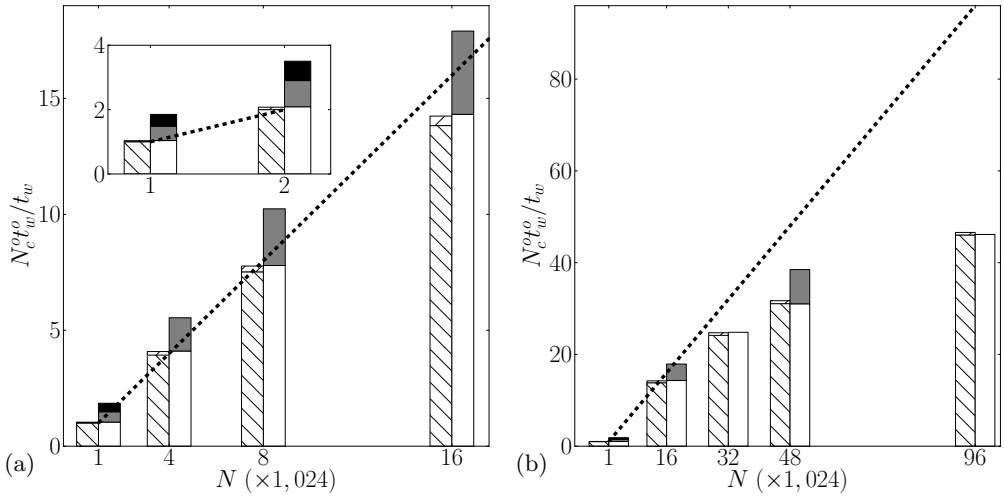

FIGURE 4. Strong scaling results for the *CharLES*^{x} code on the full *Sequoia* system: (a) zoomed view and (b) full extent covered. Tests executed with different compiler optimization levels and hyperthreading levels: backslash pattern, level 3 optimization with 1 MPI task per core; slash pattern, level 4 optimization with 1 MPI task per core; white bars, level 5 optimization with 1 MPI task per core; gray bars, O5 with 2 MPI tasks per core; black, O5 with 4 MPI tasks per core.

that unsteady compressible flow simulations in complex geometries typically require the integration of the discretized PDEs over a large period of physical time, and statistical convergence may require $\mathcal{O}(10^6 - 10^7)$ time steps. Therefore, the present case aims specifically at decreasing the time-to-solution by using such type of large-scale architecture on a moderately-sized problem, rather than at maximizing the size of the problem that can be solved. The minimum number of cores that can fit this problem in memory (allowing for hyperthreading) is given by a single rack of the *Sequoia* system (i.e., 16,384 cores), which is the starting point of this strong scalability study whose results are shown in Figure 4.

Multiple levels of compiler optimization (*O3*, *O4*, and *O5*) were tested, as well as several hyperthreading setups, ranging from one to four threads per core. Figure 4(a) shows the speedup obtained within the range of 1 to 16 racks (16,384 to 262,144 cores, respectively). The 99.8% parallel efficiency obtained when going from one to two racks confirms that the minimum number of cores considered here corresponds to an optimal parallel efficiency and ensures the meaningfulness of the overall scalability study. From 1 to 16 racks, the scalability is close to ideal. This range defines the optimal running conditions, using a minimum of about 2500 control volumes per core. Slightly better results are obtained with the *O5* mode, but the gain in this case does not exceed 3%, suggesting that more aggressive optimization levels beyond *O3* do not have a significant impact for the present form of the code on this platform. Furthermore, the memory-bound nature of the considered application makes it relatively insensitive to optimizations performed at the chip level. On the other hand, and similarly to *Hybrid*, the use of hyperthreading had a significant effect on improving performance, with a typical speedup gain of 1.5 (using 2 threads per core) and nearly 2 (using 4 threads per core). Several hyperthreaded cases attempted could not be successfully run in this testing phase of deployment of the *Sequoia* cluster, and we report only successful cases. When using more than 16 racks, the application keeps a positive scaling as shown on Figure 4(b), but with a parallel efficiency

Number of nodes	1024	2048	49152	98304
Writing speed (GB/s)	2.63	4.37	6.80	4.62

TABLE 3. Writing rates associated with the strong scaling study (*Lustre* filesystem).

decreasing from 90% on 16 racks to 48% on the full *Sequoia* machine (1,572,864 cores). This divergence from the ideal scaling curve remains reasonable, particularly for the modest size of the problem considered. Regarding the time-to-solution, the simulations using 96 racks are 3.2 times faster than those using 16 racks.

4.4. *I/O rates*

Table 3 presents writing rates measured when outputting solution files containing unstructured data from *CharLES*^{x}. We used the Parallel VTK Output (PVO) library (Krause 2012), which allows parallel writing of output files in the native XML formats of the Visualization Toolkit (VTK) library (Schroeder *et al.* 2006). As noted in Section 2, the *Sequoia* BG/Q system includes one I/O node per 128 compute nodes. A local reduction is thus performed at the inter-node level, and only a subset of MPI processes are involved in writing the output files. The rates obtained range from 3 to 7 GB/s, which is satisfactory when compared to the maximum rate of 30 GB/s measured when using optimal I/O patterns. No further optimizations were considered on this front as writing a snapshot of the full problem at a given time step takes less than 30 seconds, which is acceptable for the range of problems currently targeted with *CharLES*^{x}.

5. Conclusions and future work

The performance and scalability of two in-house compressible flow solvers, *Hybrid* and *CharLES*^{x}, were assessed on the IBM Blue Gene/Q *Sequoia* and *Vulcan* clusters at the Lawrence Livermore National Laboratory (LLNL) during an early access period prior to *Sequoia* becoming classified.

The main focus of the study for the structured-based solver, *Hybrid*, was to assess the scaling efficiency of the code up to the full capacity of 1.97 million cores made available by the temporary connection of the *Sequoia* (1.6 million cores) and *Vulcan* (393,216 cores) clusters, with minimal modifications to the code. The excellent scaling of the code, in both strong and weak scaling tests, is attributed to the explicit nature of the code, in both space and time, and the use of non-blocking MPI communication, with concurrent computation during the communication phase. The low memory footprint allows the direct use of hyperthreading through MPI, despite the non-negligible memory overhead induced by MPI when using $\mathcal{O}(10^6)$ tasks. Hyperthreading increased the performance between 50% and 80% depending on the numerical scheme considered. The low memory usage of the code led to the possibility of performing a simulation with $15{,}360 \times 16{,}384 \times 16{,}384 = 4.12 \cdot 10^{12}$ grid points, using nearly the full memory capacity of the combined *Sequoia*/*Vulcan* machine.

The unstructured-mesh flow solver *CharLES*^{x} implemented several optimizations to increase performance at the core-level as well as a new two-layer partitioning strategy for domain decomposition that works at the node level in a first step, followed by a core-level second partitioning step, in order to increase data locality and minimize MPI

communications during execution of the flow solver. Results from a strong scaling study targeting a more modest problem size (compared to those explored with the *Hybrid* code) were presented, also showing the advantage of hyperthreading for this solver, and a relatively small effect of the compiler optimization. The I/O rates when outputting solution files were found to achieve close to one fourth of the maximum rates obtained with optimal writing patterns.

A drawback of the numerical methods employed by the two solvers presented in this brief, and common to many existing CFD solvers, is their low ($\mathcal{O}(1)$) arithmetic intensity (AI) derived from the large sparse-matrix/vector multiplications that dominate the calculation of fluxes within the solver. Such low AI generally leads to memory-bound applications and consequently low percentages of peak performance (in terms of the rate of floating point operations). In the context of turbulence simulations in compressible flows, recent work by Hejazialhosseini *et al.* (2012) and Rossinelli *et al.* (2013), guided by the roof-line performance model Williams *et al.* (2009), have achieved high percentages of peak performance on Cray XE6 and BG/Q platforms, by means of extensive high-level optimization targeted to decrease total memory traffic through data re-ordering and increased cache locality. Some of these optimization techniques will be explored in future work.

Acknowledgments

The authors are grateful to Bob Walkup from the IBM Thomas J. Watson Research Center for valuable discussions on high-performance computing; to Blaise Barney, John Gyllenhaal, Scott Futral, David Fox, and Richard Hedges from the Lawrence Livermore National Laboratory for their technical support and advise in the use of the *Sequoia* and *Vulcan* clusters; to Ramesh Balakrishnan and Vitali Morozov at the Argonne Leadership Computing Facility (Argonne National Laboratory) for providing access to the *Vesta* cluster and for their help with code optimization; to Joseph W. Nichols for insightful comments on HPC; to Frank Ham for providing an early version of the *CharLES* code on which the *CharLES*^{x} extensions and architectural changes were implemented; to Ronan Vicquelin for his contribution to the initial developments of *CharLES*^{x}; and to Carlo Scalo and Jean-Pierre Hickey for their current developments and for useful discussions about the code.

The *Sequoia* and *Vulcan* IBM BG/Q systems at (LLNL) are funded by the Advanced Simulation and Computing (ASC) Program of the National Nuclear Security Administration (NNSA). Access to *Sequoia* and *Vulcan* clusters was provided through the Predictive Science Academic Alliance Program (PSAAP) at Stanford University, directed by Parviz Moin, whose support is gratefully acknowledged. The authors also thank Steve Jones (Stanford HPC Center) for coordinating access to the *Sequoia* and *Vulcan* clusters with LLNL. Access to *Vesta* was provided through the AURORA discretionary program. The *Hybrid* code was originally written by Johan Larsson when he was a Postdoctoral Fellow and Research Associate at the Center for Turbulence Research under the SciDAC program. Financial support from the Department of Energy is gratefully acknowledged.

REFERENCES

Adams, N. A. & Shariff, K. 1996 A high-resolution hybrid compact-ENO scheme for shock-turbulence interaction problems. *J. Comput. Phys.* **127**, 27–51.

Bermejo-Moreno, I., Bodart, J., Larsson, J., Barney, B., Nichols, J. & Jones, S. 2013*a* Solving the compressible navier-stokes equations on up to 1.97 million cores and 4.1 trillion grid points. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 62:1–62:10. New York, NY, USA: ACM.

Bermejo-Moreno, I., Larsson, J., Bodart, J. & Vicquelin, R. 2013*b* Wall-modeled large-eddy simulations of the HIFiRE-2 scramjet. In *Annual Research Briefs*, pp. 3–19. Center for Turbulence Research.

Bermejo-Moreno, I., Larsson, J., Campo, L., Bodart, J., Vicquelin, R., Helmer, D. & Eaton, J. 2011 Wall-modeled large eddy simulation of shock/turbulent boundary-layer interaction in a duct. In *Annual Research Briefs*, pp. 49–62. Center for Turbulence Research.

Bodart, J. & Larsson, J. 2011 Wall-modeled large eddy simulation in complex geometries with application to high-lift devices. In *Annual Research Briefs*, pp. 37–48. Center for Turbulence Research.

Bodart, J. & Larsson, J. 2012 Sensor-based computation of transitional flows using wall-modeled large eddy simulation. In *Annual Research Briefs*, pp. 229–240. Center for Turbulence Research.

Chung, I.-H., Kim, C., Wen, H.-F. & Cong, G. 2012 Application Data Prefetching on the IBM Blue Gene/Q Supercomputer. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing, SC12*, pp. 1–8.

Gilge, M. 2013 *IBM System Blue Gene Solution Blue Gene/Q Application Development*. International Business Machines Coorporation.

Haring, R. A., Ohmacht, M., Fox, T. W., Gschwind, M. K., Satterfield, D. L., Sugavanam, K., Coteus, P. W., Heidelberger, P., Blumrich, M. A., Wisniewski, R. W., Gara, A., Chiu, G. L.-T., Boyle, P. A., Chist, N. H. & Kim, C. 2012 The IBM Blue Gene/Q compute chip. *Micro, IEEE* **32** (2), 48–60.

Hejazialhosseini, B., Rossinelli, D., Conti, C. & Koumoutsakos, P. 2012 High throughput software for direct numerical simulations of compressible two-phase flows. In *Proceedings of the 2012 ACM/IEEE Conference on High Performance Computing SC12*.

Hickey, J.-P. & Ihme, M. 2013 Supercritical mixing and combustion in rocket propulsion. In *Annual Research Briefs*, pp. 21–36. Center for Turbulence Research.

Hickey, J.-P., Ma, P. C., Ihme, M. & Thakur, S. S. 2013 Large Eddy Simulation of Shear Coaxial Rocket Injector: Real Fluid Effects. In *43rd AIAA Fluid Dynamics Conference, 2013-4071*.

Johnsen, E., Larsson, J., Bhagatwala, A. V., Cabot, W. H., Moin, P., Olsson, B. J., Rawat, P. S., Shankar, S. K., Sjögreen, B., Yee, H. C., Zhong, X. & Lele, S. K. 2010 Assessment of high-resolution methods for numerical simulations of compressible turbulence with shock waves. *J. Comput. Physics* **229**, 1213–1237.

Karypis, G. & Kumar, V. 1995 Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing, SC95*.

Khalighi, Y., Nichols, J. W., Lele, S., Ham, F. & Moin, P. 2011 Unstructured large eddy simulation for prediction of noise issued from turbulent jets in various configurations. In *17th AIAA/CEAS Aeroacoustics Conference*.

Krause, D. 2012 Parallel VTK Output (PVO) Library. `https://github.com/kraused/pvo.git`.

Larsson, J. 2010 Effect of shock-capturing errors on turbulence statistics. *AIAA J.* **48(7)**, 1554–1557.

Larsson, J., Bermejo-Moreno, I., Bodart, J. & Vicquelin, R. 2012 Predicting the operability limit of the HyShot II scramjet using LES. In *Annual Research Briefs*, pp. 241–251. Center for Turbulence Research.

Larsson, J., Bermejo-Moreno, I. & Lele, S. K. 2013 Reynolds- and Mach-number effects in canonical shock-turbulence interaction. *J. Fluid Mech.* **717**, 293–321.

Larsson, J. & Gustafsson, B. 2008 Stability criteria for hybrid difference methods. *J. Comput. Phys.* **227**, 2886–2898.

Larsson, J. & Lele, S. K. 2009 Direct numerical simulation of canonical shock/turbulence interaction. *Phys. Fluids* **21**, 126101.

Pirozzoli, S. 2002 Conservative hybrid compact-WENO schemes for shock-turbulence interaction. *J. Comput. Phys.* **178**, 81–117.

Rossinelli, D., Hejazialhosseini, B., Hadjidoukas, P., Bekas, C., Curioni, A., Bertsch, A., Futral, S., S. J. Schmidt4, N. A. A. & Koumoutsakos, P. 2013 11 PFLOP/s Simulations of Cloud Cavitation Collapse. In *Proceedings of the 2013 ACM/IEEE conference on Supercomputing, SC13*.

Scalo, C., Lele, S. K. & Hesselink, L. 2013*a* Numerical investigation of a traveling-wave thermoacoustic Stirling heat engine. In *Annual Research Briefs*, pp. 165–178. Center for Turbulence Research.

Scalo, C., Lin, J., Lele, S. K. & Hesselink, L. 2013*b* Towards full-scale numerical simulations of a traveling-wave thermoacoustic Stirling heat engine. In *43rd AIAA Fluid Dynamics Conference, 2013-3208*.

Schroeder, W., Martin, K. & Lorensen, B. 2006 *The Visualization Toolkit*. Kitware.

Vane, Z. P., Bermejo-Moreno, I. & Lele, S. K. 2013 Simulations of a normal shock train in a constant area duct using wall-modeled LES. In *43rd AIAA Fluid Dynamics Conference, 2013-3204*.

Walkup, R. 2012 Application Performance Characterization and Analysis on Blue Gene/Q. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 2247–2280.

Williams, S., Waterman, A. & Patterson, D. 2009 Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* **52** (4), 65–76.