Application-specific architectures for large finite element applications

by Valerie Taylor, 1991

Philip Levis
Stanford CS349G
Winter 2021
Fundamental Challenge

- A lot of the field of computer graphics struggles with the challenges of quantizing the continuous
  - Line are lines, but we have to turn them into pixels
  - Quantizing a curved surface into triangles
  - Simulating fluids, materials, sound

- Related to, but different, from high-performance computing (Taylor's dissertation)
  - High performance computing is about accurately predicting the physical world (chemical reactions, structural stress/strain, heat transfer, explosions, weather, etc.)
  - Graphics is about making something that looks convincing
Simulation

• Want to numerically (rather than analytically) simulate what will happen to a physical system

• Discretize the system to a set of data points
  - Example: simulate water as grid cells, each cell stores whether it has water, the water's velocity, the water's pressure

• Use equations to specify how cells affect one another, take time steps
  - Example: Navier-Stokes equations for fluid flow

\[
\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{\nabla p}{\rho} + \nu \nabla^2 u
\]

• movement of velocity field
• advection
• pressure
• diffusion
Discretizing the Continuous
Discretization

- Equations are continuous, use partial derivatives: \( \frac{\partial u}{\partial t} \)
- Need to quantize/sample the space, derivatives are differences between sample points
Three Approaches

- Eulerian
- Lagrangian
- Mesh
Beam Bending

This shows a simulation of how a beam bends.
Green is no stress.
Red is tension (stretching).
Blue is compression.

\[ \Delta x = \frac{W}{48EI} (x^4 - 4L^3x - 3x^4) \]

\( E \) is elasticity modulus
\( I \) is 2nd moment of area
Need for Simulation

• Equations like the cantilever beam under uniform load are closed form solutions because the system is so simple.

• What if there are highly varying forces, time-varying forces, and complex objects?
Finite Element Method

• Represent a complex object to a *finite* set of elements, defined by a mesh

• Define the state of each element by equations based on its state, neighboring elements, and external forces.

• The combination of the mesh relationships and equations create a sparse system of linear equations

• Using the finite element method involves solving these equations to simulate what happens

• Valerie Taylor's dissertation is about a hardware accelerator to solve them faster
Finite Element Method

- Let's consider a trivial 1D case for simplicity
- Want to estimate how this bar bends
Finite Element Method

- Let's consider a trivial 1D case for simplicity
- Want to estimate how this bar bends

\[ u_{0L} \quad e_0 \quad u_{0R} \quad u_{1L} \quad e_1 \quad u_{1R} \]

\[ u_{0R} \text{ and } u_{1L} \text{ are the same point, they define how the two segments are coupled.} \]
Shape Functions

- Shape functions define how you interpolate between sample points

\[ \phi_0(x) = \frac{x - x_1}{x_0 - x_1} \]

\[ \phi_1(x) = \frac{x - x_0}{x_1 - x_0} \]

\[ u_x = \phi_0(x)u_0 + \phi_1(x)u_1 \]
Shape Functions

• Shape functions define how you interpolate between sample points

\[ \phi_0(x) = \frac{x - x_1}{x_0 - x_1} \]
\[ \phi_1(x) = \frac{x - x_0}{x_1 - x_0} \]

\[ u_x = \phi_0(x)u_0 + \phi_1(x)u_1 \]

We can use this shape function to, for example, compute how a force at \( x \) is distributed at \( x_0 \) and \( x_1 \).
Combining Equations

- We have a series of equations that define the behavior of the elements, coupled at shared points.
- To determine the behavior of the entire object, combine these equations into a linear system and solve it.

\[ \begin{bmatrix} k_{00} & k_{01A} \\ k_{1A0} & k_{1A1A} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_{1A} \end{bmatrix} \]

\[ \begin{bmatrix} k_{00} & k_{01A} & 0 \\ k_{1A0} & k_{1A1A} + k_{1B1B} & k_{1B2} \\ 0 & k_{21B} & k_{22} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_{1A} + f_{1B} \\ f_2 \end{bmatrix} \]

\( k_{AB} \) is the coefficient for how A affects B
End Result

- We have a large, sparse set of linear equations
  - $N \times N$ for $N$ sample points
  - Each row has a number of non-zero fields equal to degree of that mesh node + 1

3D grid of dimension $N$
Has $N^3$ elements
Grid point $n$ has 7 non-zero values at
- $n$ (itself)
- $n-1$, $n+1$ (x-axis)
- $n+N$, $n-N$ (y-axis)
- $n+N_2$, $n-N^2$ (z-axis)
Often need to consider corners too: 27 non-zero values
Solving FEM

• Computing solutions for FEM problems boils down to solving these sets of equations

• Many aspects of the FEM computation scale $O(N)$, but the solver does not: $O(N^2), O(N^{3/2})$
  - The fraction of computational time consumed by the solver increases as the model grows: it quickly dominates execution time
Two Parts

- Two basic methods: direct and indirect
  - Direct: fixed set of steps, require entire matrix in memory
  - Iterative: each step gets closer to the solution, tradeoff in speed vs. accuracy, can be distributed (used in HPC for this reason)

- Two techniques for improving solver performance
  - Factor the matrix into a format that's easier to solve
  - Laying out the data so it can be accessed quickly by the solver
Two Parts

- Two basic methods: direct and indirect
  - Direct: fixed set of steps, require entire matrix in memory
  - Iterative: each step gets closer to the solution, trade-off in speed vs. accuracy, can be distributed (used in HPC for this reason)

- Two techniques for improving solver performance
  - Factor the matrix into a format that's easier to solve
  - Laying out the data so it can be accessed quickly by the solver
Factorization

\[ K\vec{u} = \vec{f} \]

Our starting problem: displacement is a function of the force and the stiffness matrix.

\[ \tilde{L}\tilde{L}^T\vec{u} = \vec{f} \]

Factor stiffness matrix \( K \) into \( L \) and its transpose. \( L \) is an upper triangular matrix, \( L^T \) is a lower triangular.

\[ \tilde{L}\vec{y} = \vec{f} \]

Solve for \( y \). This is fast because \( L \) is an upper triangular matrix, so just work up increasing number of terms.

\[ \tilde{L}^T\vec{u} = \vec{y} \]

Solve for \( u \). This is fast for the same reasons as solving for \( y \).
Our starting problem: displacement is a function of the force and the stiffness matrix.

Factor stiffness matrix $K$ into $L$ and its transpose. $L$ is an upper triangular matrix, $L^T$ is a lower triangular.

Solve for $y$. This is fast because $L$ is an upper triangular matrix, so just work up increasing number of terms.

Solve for $u$. This is fast for the same reasons as solving for $y$. 

Key step
Two Parts

• Two basic methods: direct and indirect
  - Direct: fixed set of steps, require entire matrix in memory
  - Iterative: each step gets closer to the solution, tradeoff in speed vs. accuracy, can be distributed (used in HPC for this reason)

• Two techniques for improving solver performance
  - Factor the matrix into a format that's easier to solve
  - Laying out the data so it can be accessed quickly by the solver
Matrix Representation

- Matrices are extremely sparse: don't want to represent them completely
  - Wastes memory
  - Sparse memory access, wastes memory bandwidth

- Want to be able to access vectors of values, not just individual memory words

- Tradeoff in design
  - Only store non-zero values: vectors are short, more indirection
  - Pad values: vectors are longer, less indirection
CMNS

- Column-Major Nonzero Storage (CMNS) scheme

\[ K = \begin{bmatrix}
  k_{11} & 0.0 & k_{13} & 0.0 & 0.0 \\
  0.0 & k_{22} & 0.0 & 0.0 & 0.0 \\
  k_{31} & 0.0 & k_{33} & 0.0 & k_{35} \\
  0.0 & 0.0 & 0.0 & k_{44} & 0.0 \\
  0.0 & k_{52} & k_{53} & 0.0 & k_{55}
\end{bmatrix} \]

\[ \overrightarrow{K_u}^T = [k_{11}, k_{31}, k_{22}, k_{52}, k_{13}, k_{33}, k_{53}, k_{44}, k_{25}, k_{35}, k_{55}] \]

\[ \overrightarrow{R}^T = [1,3,2,5,1,3,5,4,2,3,5] \]

\[ \overrightarrow{L}^T = [2,2,3,1,3] \]

- Good: dense representation
- Bad: random access requires traversing \( L \)
- Bad: fetching irregular numbers of elements
Multiplying with CMNS

\[ \vec{K}_u^T = [k_{11}, k_{31}, k_{22}, k_{52}, k_{13}, k_{33}, k_{53}, k_{44}, k_{25}, k_{35}, k_{55}] \]

\[ \vec{R}^T = [1,3,2,5,1,3,5,4,2,3,5] \]

\[ \vec{L}^T = [2,2,3,1,3] \]

\[ \vec{v} = K \vec{p} \]

index = 1
for column in 1..N:
    for entry in 1..L[column]:
        \[ v[R[index]] = V[r[index]] + K[index] * p[column] \]
    index++;
Vector Operation Costs

• Example: Cray Y-MP computer

• Fetching a vector of length $V$ takes $19+V$ cycles
  - E.g., a element connected to 24 others requires $V=24$
  - Vector of length 24 takes 43 cycles, overhead is 44.2% of time

• High performance computing is really about high performance: if FPUs fall idle, you are wasting time
Scatter-Gather I/O

- Hardware support for transforming between sparse and dense representations

- Suppose you want to multiply a vector $v$ by a sparse matrix that is represented in CMNS
  - Want to convert the dense column in CMNS into a sparse column so it can be directly multiplied by $v$

- Scatter-gather automates this movement
  - *Scatter* dense elements into a sparse representation
  - *Gather* sparse elements into a dense representation
HPC Today

- Highly parallel clusters (10,000+ cores)
- All simulations are distributed across many machines
- Bottleneck is communication and barriers rather than FPU utilization
- Ghost cells, iterative solvers
- Infiniband/MPI

Ghost cells exchanged between nodes
Valerie Taylor's Contributions