Accelerating CFD Simulations with GPUs

Patrice Castonguay
HPC Developer Technology Engineer
December 4th, 2012
Outline

- Computational Fluid Dynamics
- Why GPUs?
- Accelerating ANSYS Fluent (commercial CFD software)
  - Background
  - Novel parallel algorithms to replace sequential algorithms
  - Performance results
- Accelerating SD3D (research code developed at Stanford)
  - Background
  - Performance results
- Conclusions
Computational Fluid Dynamics
Computational Fluid Dynamics

- Simulation of fluid flows
- Large number of applications
Computational Fluid Dynamics

Navier-Stokes equations: coupled nonlinear partial differential equations which govern unsteady, compressible, viscous fluid flows

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} + \frac{\partial (\rho v)}{\partial y} + \frac{\partial (\rho w)}{\partial z} &= 0 \\
\frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho u^2)}{\partial x} + \frac{\partial (\rho uv)}{\partial y} + \frac{\partial (\rho uw)}{\partial z} &= -\frac{\partial p}{\partial x} + \frac{1}{Re} \left( \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} \right) \\
\frac{\partial (\rho v)}{\partial t} + \frac{\partial (\rho v^2)}{\partial x} + \frac{\partial (\rho uv)}{\partial y} + \frac{\partial (\rho vw)}{\partial z} &= -\frac{\partial p}{\partial y} + \frac{1}{Re} \left( \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} \right) \\
\frac{\partial (\rho w)}{\partial t} + \frac{\partial (\rho w^2)}{\partial x} + \frac{\partial (\rho uv)}{\partial y} + \frac{\partial (\rho vw)}{\partial z} &= -\frac{\partial p}{\partial z} + \frac{1}{Re} \left( \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} \right) \\
\frac{\partial (\rho e)}{\partial t} + \frac{\partial}{\partial x} \left( \rho u \left( e + \frac{p}{\rho} \right) \right) + \frac{\partial}{\partial y} \left( \rho v \left( e + \frac{p}{\rho} \right) \right) + \frac{\partial}{\partial z} \left( \rho w \left( e + \frac{p}{\rho} \right) \right) &= \\
- \frac{1}{Re Pr} \left[ \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \right] + \frac{1}{Re} \left[ \frac{\partial}{\partial x} (u \tau_{xx} + v \tau_{xy} + w \tau_{xz}) + \frac{\partial}{\partial y} (u \tau_{xy} + v \tau_{yy} + w \tau_{yz}) + \frac{\partial}{\partial z} (u \tau_{xz} + v \tau_{yz} + w \tau_{zz}) \right]
\end{align*}
\]
Computational Fluid Dynamics

- Partition the domain into large number of cells, and solve for fluid properties (density, velocity, pressure) inside each cell.
Why GPUs?
Why GPUs

Higher performance

Intel CPU:
- 8-core Sandy Bridge
  - fp32 performance: 384 Gflops
  - fp64 performance: 192 Gflops
  - Memory bandwidth: 52 GB/s

NVIDIA GPU:
- Tesla GK110
  - fp32 performance: 3935 Gflops \( \Rightarrow 10X \)
  - fp64 performance: 1311 Gflops \( \Rightarrow 6.8X \)
  - Memory bandwidth: 250 GB/s \( \Rightarrow 4.8X \)
Why GPUs

- Power efficiency
  - Traditional CPUs not economically feasible

Jaguar (3rd fastest supercomputer in Nov. 2011)
2.3 Petaflops @ 7 megawatts

7 megawatts \(\cong\) 7,000 Homes
Why GPUs

- Power efficiency
  - Traditional CPUs not economically feasible

Jaguar @ 2.3 Petaflops
7 megawatts
7,000 homes

Scaled to 100 petaflops

300 megawatts
300,000 homes
≈ Quebec City and its metropolitan area
Why GPUs

CPU
Optimized for Serial Tasks

GPU Accelerator
Optimized for Many Parallel Tasks

Higher computational power per watt
Why GPUs

Many scientific applications already benefit from GPUs

Relative Performance K20x vs. dual-socket Sandy Bridge
E5-2687w 3.10 GHz Sandy Bridge
Why GPUs

- Algorithms found in computer games are surprisingly similar to the ones found in scientific applications.
ANSYS Fluent
Large share of CFD software market

Finite volume method

Many different options: incompressible/compressible, inviscid/viscous, two-phase, explicit/implicit, …)

Most popular option: implicit incompressible Navier-Stokes solver
Incompressible Navier-Stokes

- Coupled non-linear PDEs
- Unknowns: $u,v,w$ and $p$ (density is constant)

**Mass:**
\[
\int_{\partial \Omega} \rho \mathbf{v} \cdot d\mathbf{S} = 0
\]

**Momentum x:**
\[
\int_{\partial \Omega} \rho u \mathbf{v} \cdot d\mathbf{S} - \int_{\partial \Omega} \mu (\nabla u) \cdot d\mathbf{S} + \int_{\Omega} \frac{\partial p}{\partial x} d\Omega = 0
\]

**Momentum y:**
\[
\int_{\partial \Omega} \rho v \mathbf{v} \cdot d\mathbf{S} - \int_{\partial \Omega} \mu (\nabla v) \cdot d\mathbf{S} + \int_{\Omega} \frac{\partial p}{\partial y} d\Omega = 0
\]

**Momentum z:**
\[
\int_{\partial \Omega} \rho w \mathbf{v} \cdot d\mathbf{S} - \int_{\partial \Omega} \mu (\nabla w) \cdot d\mathbf{S} + \int_{\Omega} \frac{\partial p}{\partial z} d\Omega = 0
\]
Incompressible Navier-Stokes

Solution procedure:

- Assemble \textit{Linear} System of Equations
- Solve Linear System of Equation, \( Ax = b \)
- Converged?
  - No: Accelerate this first
  - Yes: Stop

Runtime:

- \( \sim 33\% \)
- \( \sim 67\% \)
Incompressible Navier-Stokes

- Large sparse system of equations with 4x4 block entries

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22} \\
A_{31} & A_{33} \\
A_{42} & A_{44} \\
A_{53} & A_{55} \\
A_{611} & A_{64} \\
A_{66} & A_{66}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6
\end{bmatrix}
\]

- \( A_{ij} \) is 4x4 matrix
- \( x_i \) is 4x1 vector (\( x_i = [p_i, u_i, v_i, w_i] \))

In practice, millions of rows
Multigrid

- 2D Poisson equation, Jacobi smoother

![Initial error](image1)

![Error after 5 iterations](image2)

![Error after 10 iterations](image3)

![Error after 15 iterations](image4)

Convergence of error norm
Most simple iterative solvers are efficient at damping high-frequency errors, but inefficient at damping low-frequency error.

**Key idea:**
Represent the error on a coarser grid so that low-frequency errors become high frequency errors.
Algebraic Multigrid (AMG)

Example: Two-Level V-cycle

\[ A^f x^f = b^f \]

Pre-Smooth

Compute Residual

\[ r^f = b^f - A^f x^f \]

Restrict Residual

\[ Rr^f = r^c \]

Create

\[ A^c = RA^f P \]

Solve

\[ A^c e^c = r^c \]

Post-Smooth

Prolongate Correction

\[ x^f = x^f + P e^c \]
Algebraic Multigrid (AMG)

- Apply recursively

\[ A^f x^f = b^f \]

- Pre-smooth
- Coarsen
- Pre-smooth
- Coarsen
- Pre-smooth
- Coarsen
- 1-2 Unknowns

\[ A^c e^c = r^c \]

- Solve
- Prolongate Correction
- Post-smooth
- Prolongate Correction
- Post-smooth
- Prolongate Correction
- Post-smooth

\(~10^6\) Unknowns
ANSYS Fluent

- We have accelerated the AMG solver in Fluent with GPUs

- Some parts of the algorithm were easily ported to the GPU
  - Sparse matrix vector multiplications
  - Norm computations

- Other operations were inherently sequential
  - Aggregation procedure to create restriction operator
  - Gauss-Seidel, DILU smoothers
  - Matrix-matrix multiplication

- Need to develop novel parallel algorithms!
Aggregation

- How do we create the coarser levels?

- Graph representation of a matrix:

\[
\begin{bmatrix}
A_{11} & A_{12} & A_{13} & & A_{16} \\
A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\
A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\
A_{42} & A_{44} & A_{45} & A_{46} \\
A_{52} & A_{53} & A_{54} & A_{55} & \\
A_{61} & A_{64} & A_{66} & & \\
\end{bmatrix}
\]

\[w_{3,5} = f(A_{3,5}, A_{5,3})\]
Aggregation

How do we create the coarser levels?

\[ w_{i,j} = f(A_{i,j}, A_{j,i}) \]
Aggregation

In aggregation-based AMG, group vertices that are strongly connected to each other.
Aggregation

New matrix/graph:
Aggregation

- Want to merge vertices that are strongly connected to each other.
- Similar to a weighted graph matching problem.

\[ w_{i,j} = f(A_{i,j}, A_{j,i}) \]
Parallel aggregation

- Each vertex extends a hand to its strongest neighbor
Parallel aggregation

- Each vertex checks if its strongest neighbor extended a hand back
Parallel aggregation
Parallel aggregation

- Repeat with unmatched vertices
Parallel aggregation
Parallel aggregation
Parallel aggregation
Parallel aggregation
Parallel aggregation
Smoothers

- Smoothing step

\[ x^{n+1} = x^n + M^{-1}(b - Ax^n) \]

- \( M \) is called preconditioning matrix
- Here \( x \) is solution vector (includes all unknowns in the system)
Jacobi Smoother

For Jacobi, $M$ is block-diagonal

$$M = \begin{bmatrix}
D_1 & & \\
& D_2 & \\
& & \ddots \\
& & & D_{n-1} \\
& & & & D_n
\end{bmatrix}$$

Inherently parallel, each $x_i$ can be updated independently of each other

Maps very well to the GPU
Parallel Smoothers

- Smoothers available in Fluent (Gauss-Seidel & DILU) are sequential!
- For DILU smoother, $M$ has the form

$$M_{DILU} = (E + L)E^{-1}(E + U)$$

- $E$ is block diagonal matrix such that

$$\text{diag}(M_{DILU}) = \text{diag}(A)$$

$$\text{diag}(M_{DILU}) = \text{diag}(E + LE^{-1}U) = \text{diag}(A)$$

- Can recover ILU(0) for certain matrices
- Only requires one extra diagonal of storage
DILU Smoother

Construction of block diagonal matrix $E$ is sequential

\[
E_{11} = A_{11}
\]

\[
E_{22} = A_{22} - L_{21} E_{11}^{-1} U_{12}
\]

\[
E_{33} = A_{33} - L_{31} E_{11}^{-1} U_{13} - L_{32} E_{22}^{-1} U_{23}
\]

\[
E_{44} = A_{44} - L_{41} E_{11}^{-1} U_{14} - L_{42} E_{22}^{-1} U_{24} - L_{43} E_{33}^{-1} U_{34}
\]

Inversion of $M$ is also sequential (two triangular matrix inversions)

\[
\Delta = M^{-1}(b - Ax)
\]

\[
M_{DILU} = (E + L) E^{-1} (E + U)
\]
Coloring

- Use coloring to extract parallelism
- Coloring: assignment of color to vertices such that no two vertices of same color are adjacent
Coloring

- Use coloring to extract parallelism
- With $m$ unknowns and $p$ colors, $m/p$ unknowns can be processed in parallel
DILU Smoother

• Construction of block diagonal matrix $E$ is now parallel

\[
E_{11} = A_{11} \\
E_{22} = A_{22} \\
E_{33} = A_{33} - L_{31} E_{11}^{-1} U_{13} - L_{32} E_{22}^{-1} U_{23} \\
E_{44} = A_{44} - L_{41} E_{11}^{-1} U_{14} - L_{42} E_{22}^{-1} U_{24}
\]

• Inversion of matrix $M$ is also parallel
Parallel Graph Coloring – Min/Max

- How do you color a graph/matrix in parallel?
- Parallel graph coloring algorithm of Luby, new variant developed at NVIDIA
Parallel Graph Coloring – Min/Max

Assign a random number to each vertex
Parallel Graph Coloring – Min/Max

- Round 1: Each vertex checks if it’s a local maximum or minimum
- If max, color=dark blue. If min, color=green
Parallel Graph Coloring – Min/Max

- Round 2: Each vertex checks if it’s a local maximum or minimum.
- If max, color=pink. If min, color=red
Parallel Graph Coloring – Min/Max

- Round 3: Each vertex checks if it's a local maximum or minimum.
- If max, color=purple. If min, color=white
AMG Timings

- CPU Fluent solver: AMG(F-cycle, agg8, DILU, 0pre, 3post)
- GPU nvAMG solver: AMG(V-cycle, agg8, MC-DILU, 0pre, 3post)
AMG Timings

- CPU Fluent solver: AMG(F-cycle, agg8, DILU, 0pre, 3post)
- GPU nvAMG solver: AMG(V-cycle, agg2, MC-DILU, 0pre, 3post)
Flux Reconstruction (FR) Method

- Proposed by Huynh in 2007, similar to the Spectral Difference and Discontinuous Galerkin methods.
- High-order method: spatial order of accuracy is > 2.

![Graph showing computational cost vs. error for high-order and low-order methods.](image-url)
Flux Reconstruction (FR) Method

- Solution in each element approximated by a multi-dimensional polynomial of order $N$
- Order of accuracy: $h^{N+1}$
- Multiple DOFs per element

$N=2$
Flux Reconstruction (FR) Method

- Plunging airfoil: zero AOA, Re=1850, frequency: 2.46 rad/s
- 5th order accuracy in space, 4th order accurate RK time stepping
Flux Reconstruction (FR) Method

- Computations are **demanding:**
  - Millions of DOFS
  - Hundreds of thousands of time steps
  - More work per DOF compared to low-order methods

- Until recently, high-order simulations over complex 3D geometries were **intractable**, unless you had access to large cluster

- **GPUs to the rescue!**
GPU Implementation

- Ported entire code to the GPU

- FR and other high-order methods for unstructured grids map well to GPUs:
  - Large amount of parallelism (millions of DOFs)
  - More work per DOF compared to low-order methods
  - Cell-local operations benefit from fast user-managed on-chip memory

- Required some programming efforts, but was worth while…
Single-GPU Implementation

Speedup of the single-GPU algorithm (Tesla C2050) relative to a parallel computation on a six-core Xeon x5670 (Westmere) @ 2.9GHz
Applications

- Unsteady viscous flow over sphere at Reynolds 300, Mach=0.3
- 28914 prisms and 258075 tets, 4th order accuracy, 6.3 million DOFs
- Ran on desktop machine with 3 C2050 GPUs
Applications

- 3 GPUs: same performance as 21 Xeon x5670 CPUs (126 cores)
- 3 GPUs personal computer: ~$10,000, easy to manage

Iso-surfaces of Q-criterion colored by Mach number for flow over sphere at Re=300, M=0.3
Multi-GPU Implementation

Speedup relative to 1 GPU for a 6th order accurate simulation running on a mesh with 55947 tetrahedral elements

- No Overlap
- Communication Overlap
- Communication and GPU Transfers Overlap
Applications

- Transitional flow over SD7003 airfoil, Re=60000, Mach=0.2, AOA=4°
- 4th order accurate solution, 400000 RK iterations, 21.2 million DOFs
Applications

15 hours on 16 C2070s

157 hours (> 6 days) on 16 Xeon x5670 CPUs
Conclusions

- Most scientific applications have large amount of parallelism

- Parallel applications map well to GPUs which have hundreds of simple, power efficient cores

- Higher performance, higher **performance/watt**

- Presented two successful uses of GPUs in CFD
  - Linear solver in Ansys Fluent (hard to parallelize)
  - Research-oriented CFD code
Conclusions

- Future of HPC is CPU + GPU/Accelerator

- Need to develop new parallel numerical methods to replace inherently sequential algorithms (such as Gauss-Seidel, ILU preconditioners, etc.)

- Peak flops vs memory bandwidth gap still growing
  - Flops are “free”
  - Need to develop numerical methods that have larger flops/bytes ratio
Questions?

Patrice Castonguay
pcastonguay@nvidia.com