GPU Computing for Computational Science

Jonathan Cohen

Senior Research Scientist

NVIDIA Research

NVIDIA

State of Computational Science

Legacy codes in wide use

Computational science community is cautiously exploring GPUs

Major issue: How to handle legacy code?

- Accelerate? Rewrite?
- Is it worth it? (lots of ink spilled about this…)

Wildcard: What about radically different approaches?

Conceptual Roadmap

- Option 1: Accelerate
- Option 2: Rewrite
- Option 3: Rethink

- “But my code already runs on dual core. Why can’t I just recompile?”

Why a GPU isn’t just a CPU with 100x more cores

How do you measure speed?

“How fast can you do one thing?”

(latency)

vs.

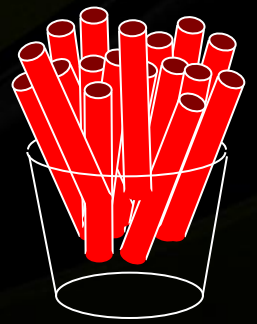
“How much can you do per second?”

(throughput)

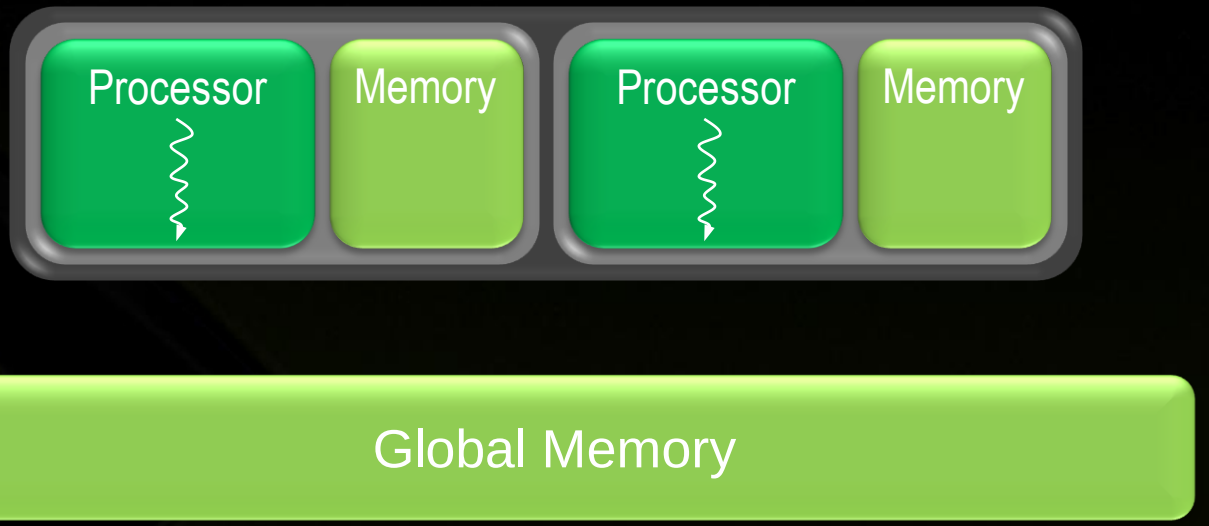
Throughput = Parallelism

Latency: sip through a really big straw

Throughput: Use 100 small straws

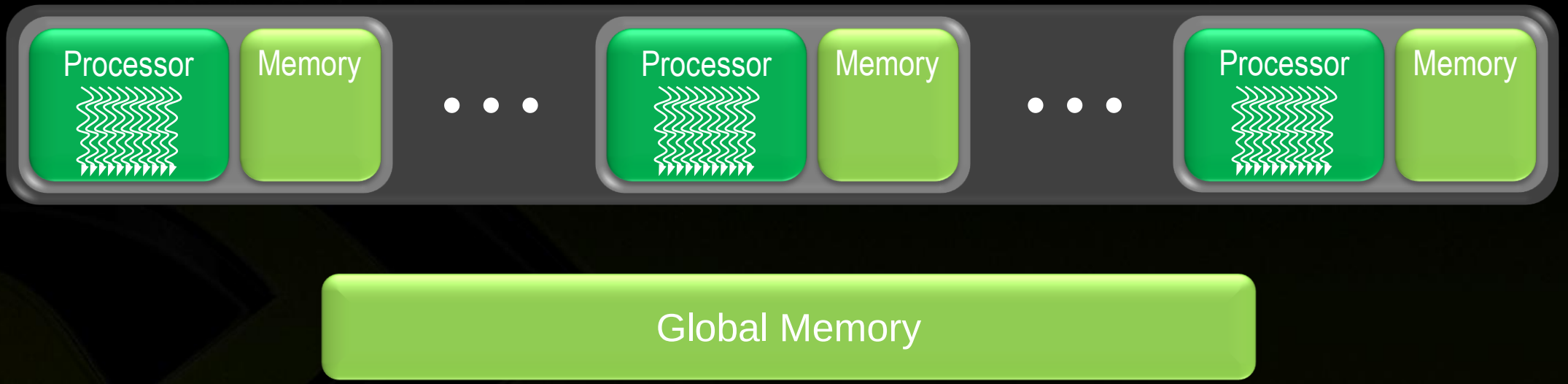


Multicore CPU: Run ~10 Threads Fast



- **Few processors, each supporting 1–2 hardware threads**
- **On-chip memory/cache near processors**
- **Shared global memory space (external DRAM)**

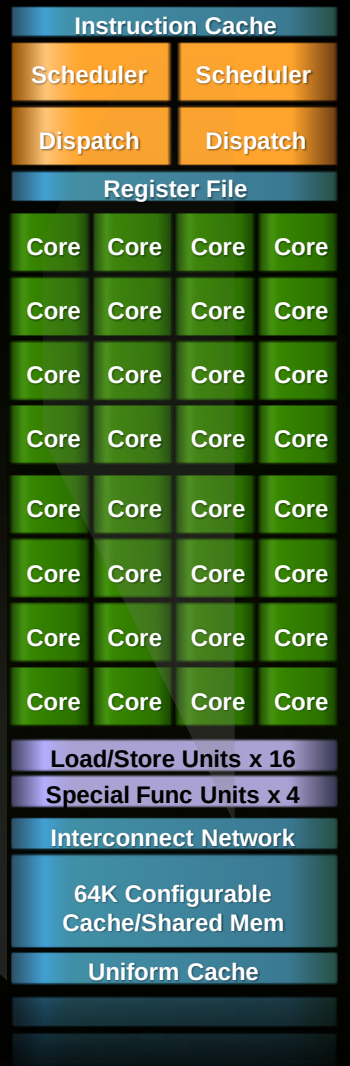
Manycore GPU: Run ~10,000 Threads Fast



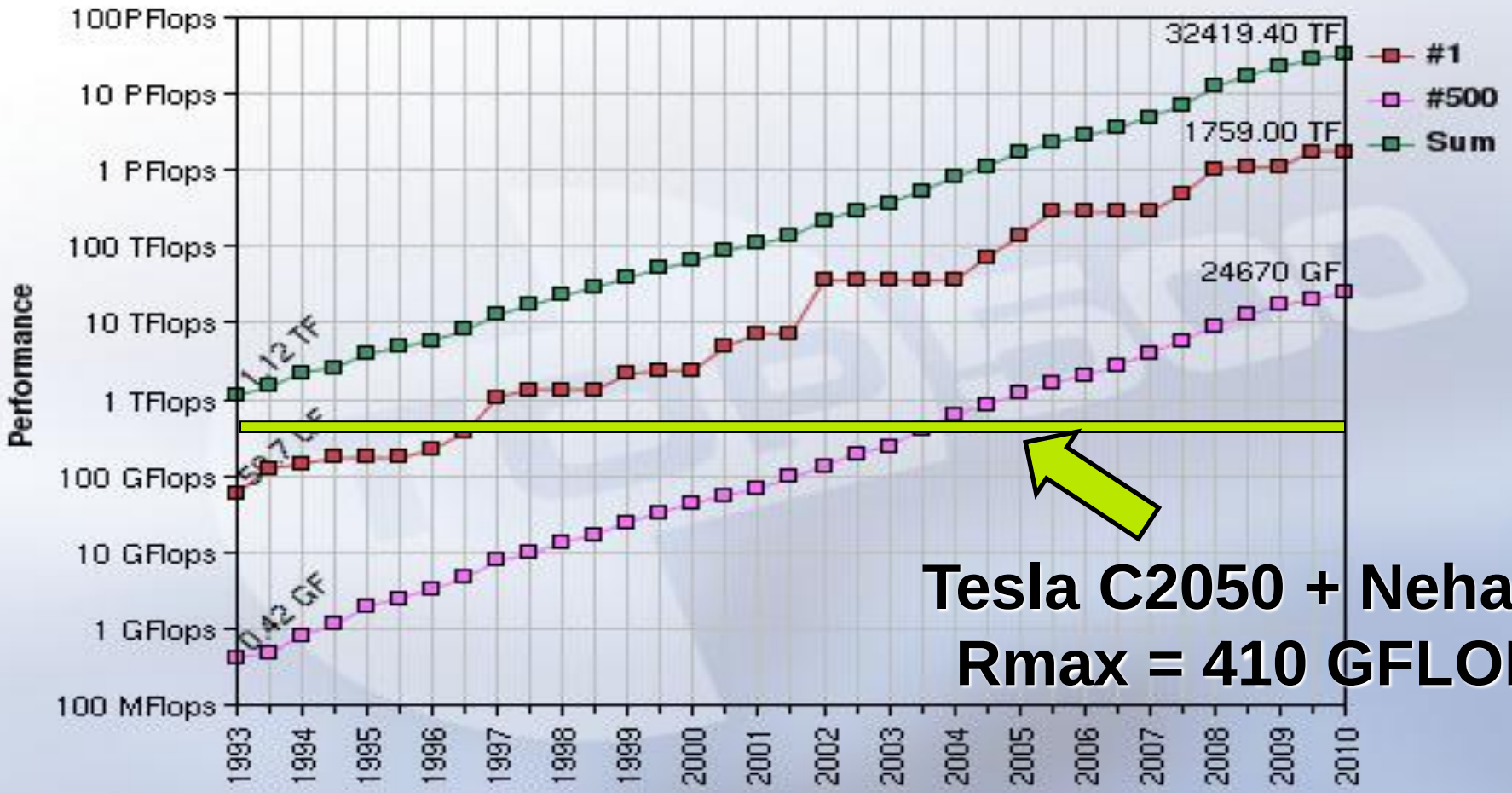
- **Hundreds of processors, each supporting hundreds of hardware threads**
- **On-chip memory/cache near processors**
- **Shared global memory space (external DRAM)**

NVIDIA “Fermi” Parallel Computing Architecture

- Designed for throughput
- Up to 512 Cores
- Singe Precision: >1 TFLOPS
- Double Precision: ~0.5 TFLOPS



Performance Development



Tesla C2050 + Nehalem
Rmax = 410 GFLOPS

What Computational Science Breakthroughs Happened in 2003?

- Universe of protein structures mapped
- Discovery that 80% of tropopause height increase due to human activity – global warming “fingerprint”
- First comprehensive analysis of Y chromosome
- Analysis of WMAP data establishes age of universe, curvature, Hubble’s constant
- High resolution simulations of 1994 Northridge Quake
- Human Genome Project published first complete version

This was not the stone ages!

What does this mean for computational science?

Option 1: Accelerate

Case Study: FEAST from TU Dortmund

Finite Element Analysis and Solution Tools

Complex FE code for CFD and Structural Mechanics

Dominik Göddeke et al. accelerated using GPUs

FEAST-GPU Approach: High level of abstraction

- Minimally invasive co-processor integration
- Identify and isolate "accelerable" parts of a computation
- Chunks must be large enough to amortize co-processor drawbacks (PCIE, change of data layout, etc.)

Portions of this slide courtesy Dominik Göddeke

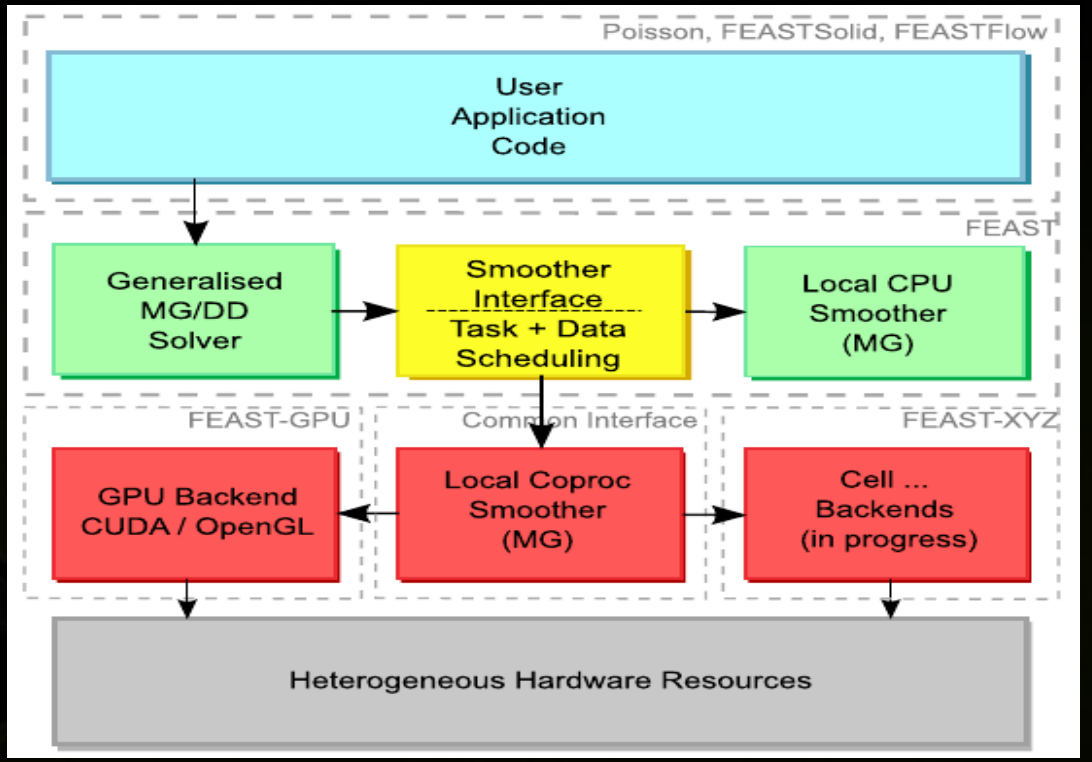
FEAST-GPU Design Philosophy

FEAST-GPU Goal:

- Integrate several co-processors into existing large-scale software package...
- *...without modifying application code*

- NOT mapping single application to GPU / GPU Cluster

Balance acceleration potential and acceleration effort



Portions of this slide courtesy Dominik Göddeke

FEAST-GPU Integration Results

Opteron 2214, 4 nodes

GeForce 8800 GTX

CUDA backend

18.8 M DOF

$$\begin{pmatrix} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ B_1^T & B_2^T & C \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ p \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ g \end{pmatrix}$$

Accel. fraction R_{acc}: 75%
Local speedup S_{local}: 11.5x
Theoretical limit S_{max}: 4x
Global speedup S_{total}: 3.8x

fixed point iteration
solving linearised subproblems with
global BiCGStab (reduce initial residual by 1 digit)
Block-Schurcomplement preconditioner
1) approx. solve for velocities with
global MG (V 1+0), additively smoothed by
for all Ω_i: solve for $\mathbf{u_1}$ with **local MG**
for all Ω_i: solve for $\mathbf{u_2}$ with **local MG**
2) update RHS: $\mathbf{d_3} = -\mathbf{d_3} + \mathbf{B^T}(\mathbf{c_1}, \mathbf{c_2})^T$
3) scale $\mathbf{c_3} = (\mathbf{M_p^L})^{-1}\mathbf{d_3}$

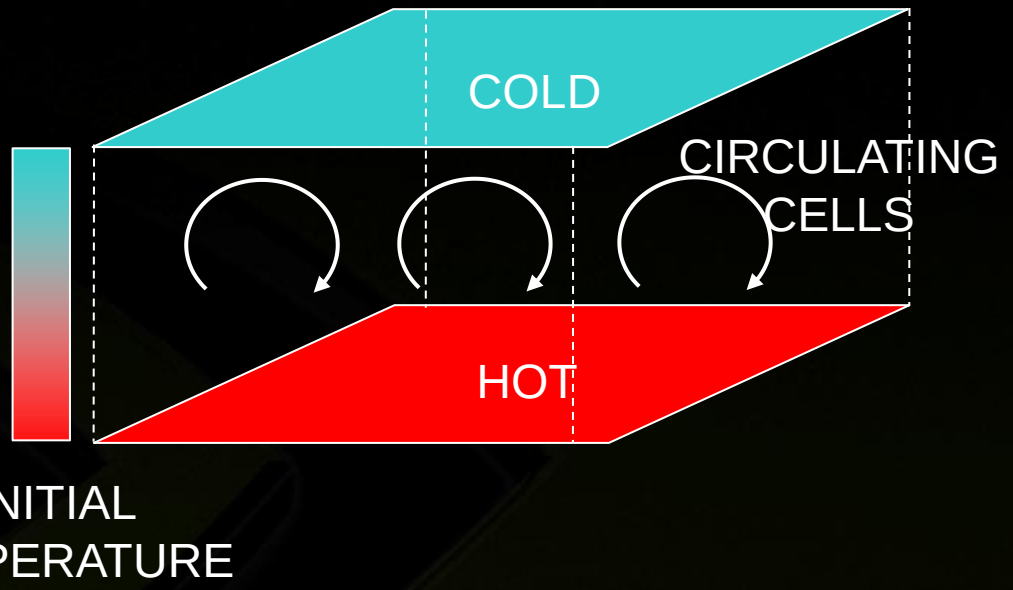
Portions of this slide courtesy Dominik Göddeke

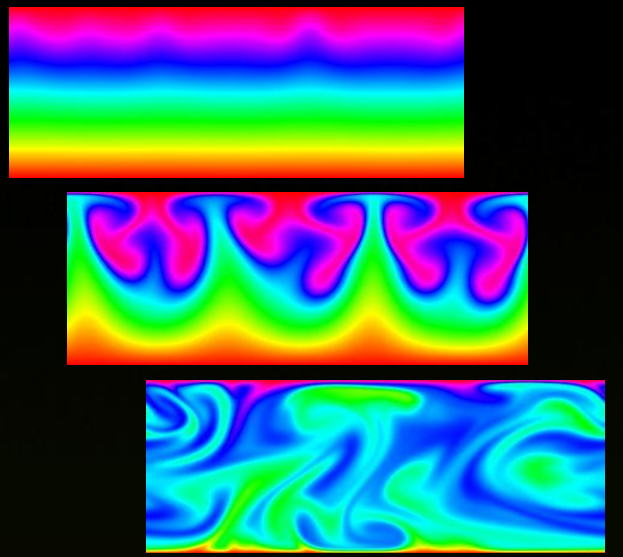
Option 2: Rewrite

- If you were to attempt a rewrite:
  - Good overall design?
  - What data structures / algorithms to use?
  - Is it worth the effort – does 10x improvement on bottlenecks translate into 10x improvement for entire system?
- Major Take-away: Avoid All Serial Bottlenecks
- In particular: Avoid All PCIE Transfers

  => Move Everything to GPU

Rayleigh-Bénard Benchmark





- "OpenCurrent": 2nd order Finite Volume Cartesian fp64 CUDA code - entire code runs on GPU
- Transition from stratified (top) to turbulent (bottom)
- Validated / benchmarked non-linear problems against published results & existing Fortran code

Benchmark Methodology

- Fortran code
  - Written by Jeroen Molemaker @ UCLA
  - 8 Threads (via MPI and OpenMP) on 8-core 2.5 GHz Xeon
  - Several oceanography pubs using this code, ~10 years of code optimizations. Code is small & fast.
- Per-step calculation time varies due to convergence rate of pressure solver
- Record time once # of v-cycles stabilizes
  - Point relaxer on GPU – 1 FMG + 7 v-cycles
  - Line relaxer on CPU – 1 FMG + 13 v-cycles
- See Cohen & Molemaker, ParCFD 2009

Benchmark Results – early 2009

- CUDA (1 Tesla C1060) vs. Fortran (8-core 2.5 GHz Xeon)
- As "apples-to-apples" as possible ($ and manpower)
  - Equal price nodes (in 2009: ~$3k)
  - Skilled programmers in each paradigm

Resolution	CUDA time/step	Fortran time/step	Speedup
64 x 64 x 32	24 ms	47 ms	2.0x
128 x 128 x 64	79 ms	327 ms	4.1x
256 x 256 x 128	498 ms	4070 ms	8.2x
384 x 384 x 192	1616 ms	13670 ms	8.5x

HOOMD blue

NVIDIA

- **MD Code from Joshua Anderson *et al.***
- **Designed to run on GPU(s) or CPU(s)**

Integration
- NVT (Nosé-Hoover)
- NPT
- Langevin Dynamics
- NVE

Bond forces
- harmonic
- FENE

Angle forces
- harmonic
- CGCMM

Dihedral/Improper forces
- harmonic

Simulation types
- 2D and 3D
- Replica exchange

Snapshot formats
- MOL2
- DCD
- PDB
- XML

Pair forces
- Lennard Jones
- Gaussian
- CGCMM
- Morse
- Table (arbitrary)
- Yukawa

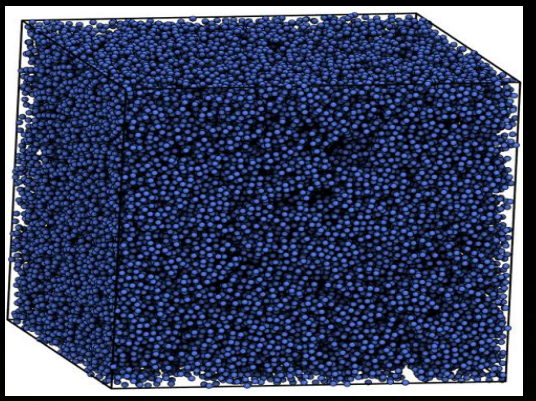
Many-body forces
- EAM (*coming soon*)

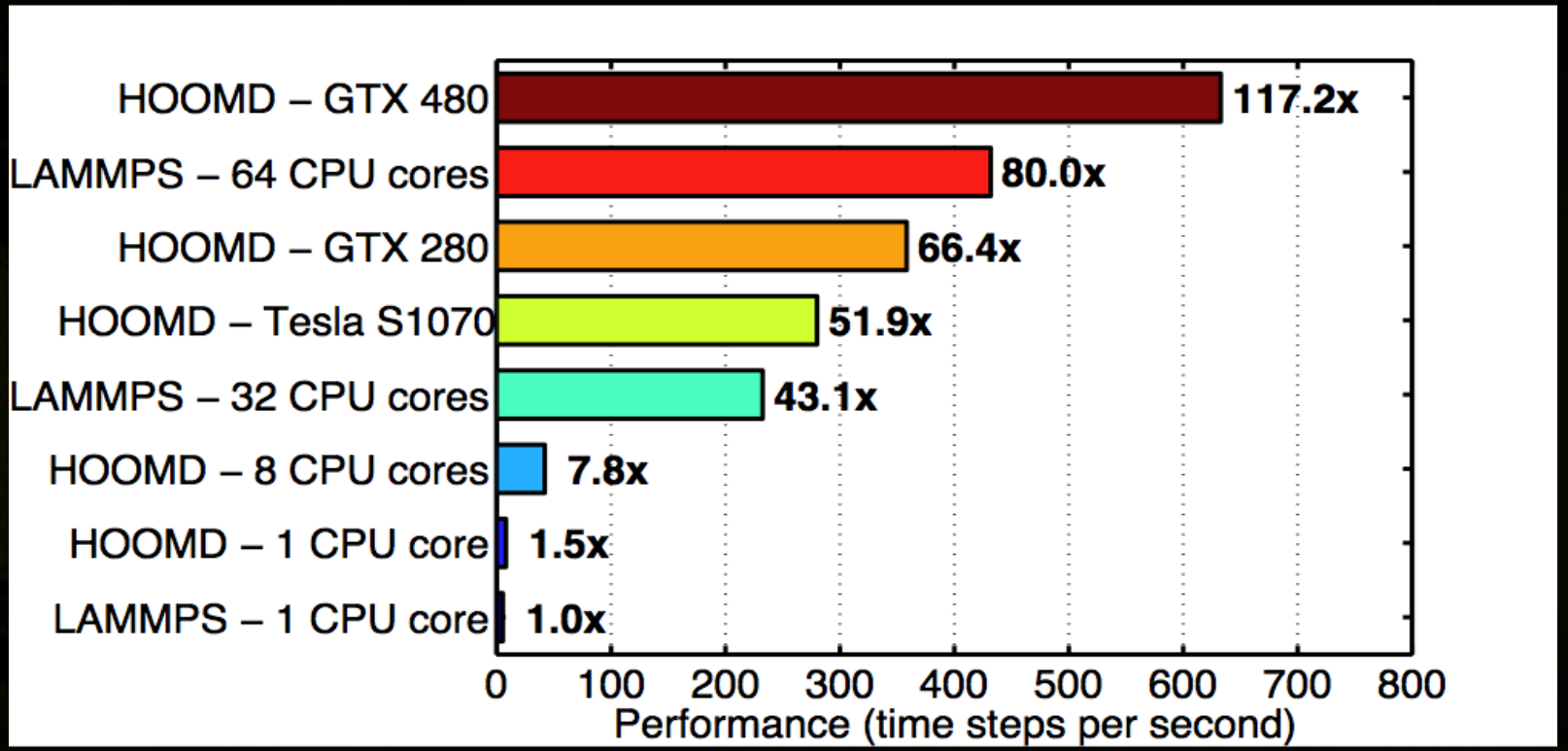
Hardware support
- All recent NVIDIA GPUs
- Multi-core CPUs via OpenMP

Portions of this slide courtesy Joshua Anderson

HOOMD-blue Benchmark

- 64,000 particle Lennard-Jones fluid simulation
- representative of typical performance gains





*CPU: Intel Xeon E5540 @ 2.53GHz

Portions of this slide courtesy Joshua Anderson

Option 3: Rethink the Numerics

- Numerical methods + programming languages + compilers + architectures + programming paradigms = co-evolution
  - Popular methods are easy to express in popular languages, run well on popular hardware
  - Not a coincidence!
- New architectures = opportunity for new numerics
- We overlooked approaches because they were impractical … maybe no longer true
- Paradigm shifts upend conventional wisdom

Example:
Nodal Discontinuous Galerkin Methods

Work from Tim Warburton & Andreas Klöckner *et al.* @ Brown & Rice

Solve conservation laws over unstructured grids

$$u_t + \nabla \cdot F(u) = 0$$

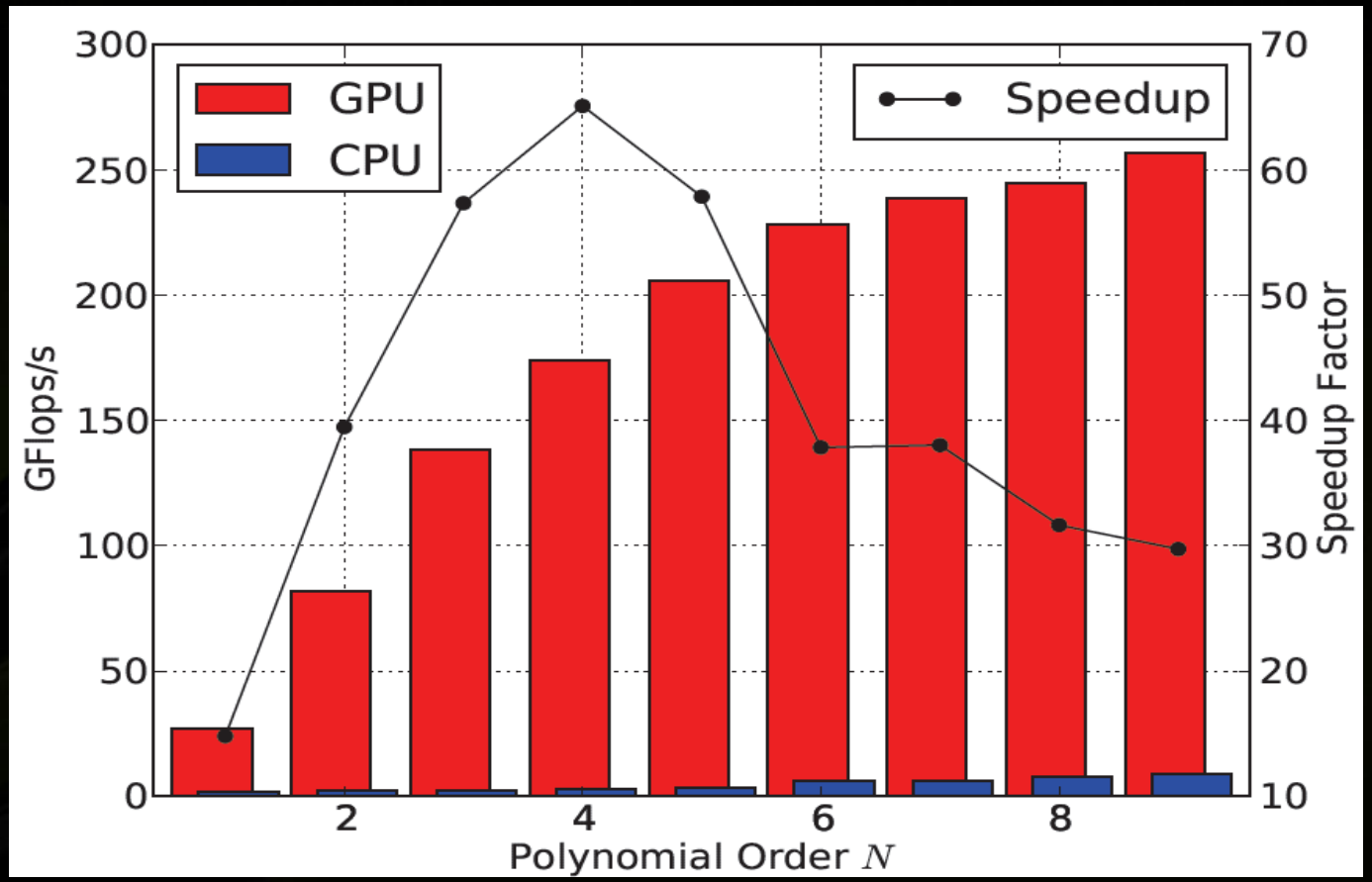
DG on GPUs: Why?

- GPUs have deep memory hierarchy
  - The majority of DG is local (matrix structure)
- Compute Bandwidth >> Memory Bandwidth
  - DG is arithmetically intense.
  - Adopt "FLOPS are free" philosophy

Portions of this slide courtesy Andreas Klöckner

Early DG Results

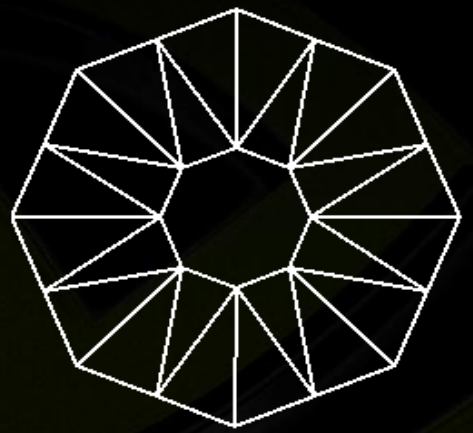
Nvidia GTX280 vs. single core of Intel E8400 - Maxwell's Equations

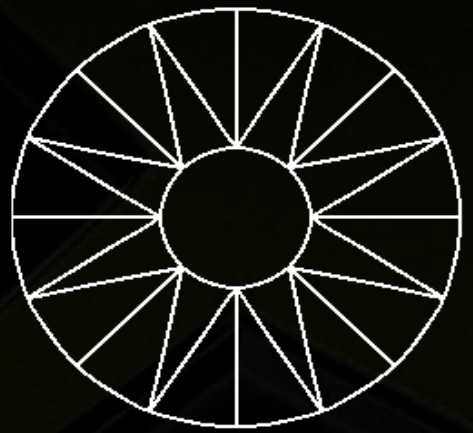


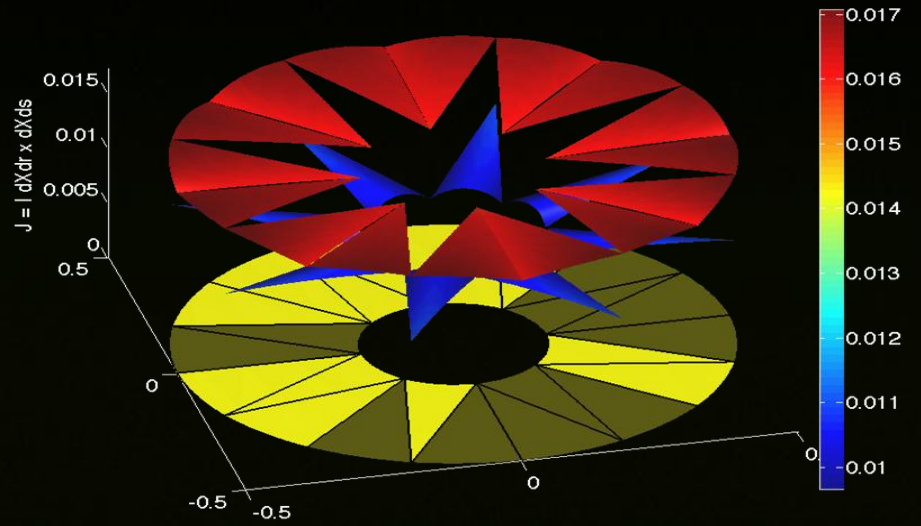
Portions of this slide courtesy Andreas Klöckner

Curvilinear DG: Needs Jacobians

$$J = \left| \frac{\partial \mathbf{x}}{\partial r} \times \frac{\partial \mathbf{x}}{\partial s} \right| \text{ appears in inner products: } (u,v)_T = \iint_{\hat{T}} u(r,s)v(r,s)J(r,s)\,dr\,ds$$







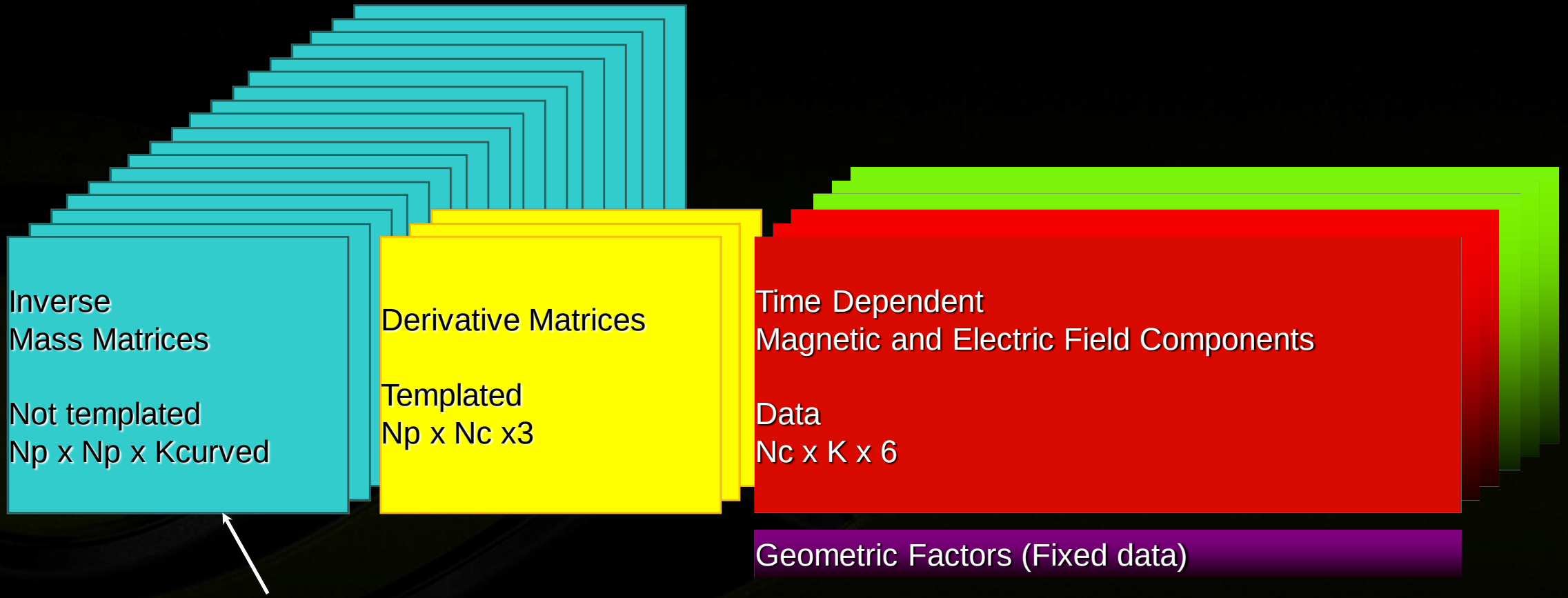
FEM Mesh

Curvilinear Mesh

Piecewise polynomial determinant of the Jacobian plotted vertically

Portions of this slide courtesy Tim Warburton

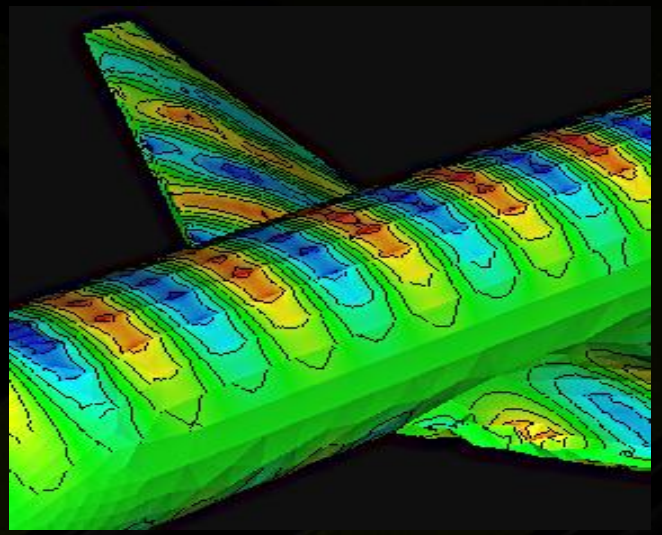
Curvilinear DG Matrix Structures



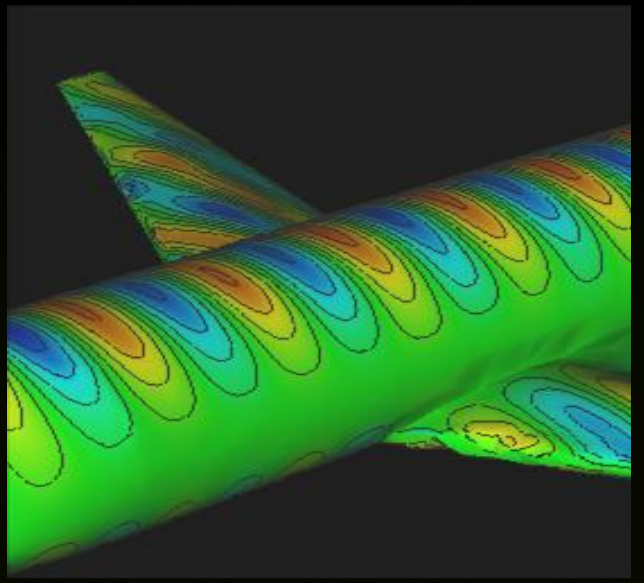
Each curved element requires its own mass matrix (lots of memory)
Compute on the fly on GPU since FLOPS are free

Portions of this slide courtesy Tim Warburton

Maxwell Equation Solution with Curvilinear DG



Paneled



Curvilinear

Portions of this slide courtesy Tim Warburton

Accelerate, Rewrite, or Rethink?

- **Accelerate** Legacy Codes
  - Use CUBLAS / CUFFT / thrust / matlab / cusp / PGI-Accelerator / etc.
    => good work for domain scientists (minimal computer science required)
- **Rewrite** New Codes
  - Opportunity for clever algorithmic thinking
    => good work for computer scientists (minimal domain knowledge required)
- **Rethink** Numerical Methods
  - Potential to transform science
    => Interdisciplinary: requires CS *and* domain insight
    => Exciting time to be a computational scientist!

Thanks

- Andreas Klöckner
- Tim Warburton
- Dominik Göddeke
- Joshua Anderson