GPU Computing with CUDA
Lecture 9 - Applications - CFD

Christopher Cooper
Boston University

August, 2011
UTFSM, Valparaíso, Chile

Outline of lecture

- Overview of CFD
  - Navier Stokes equations
  - Types of problems
  - Discretization methods
- “Conventional” CFD
- Port CFD codes to CUDA
- Efforts
- Example problem: implicit heat transfer

CFD - Introduction

- Numerical modeling of fluid systems
- Navier-Stokes equation: momentum conservation

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

- Type of problems:
  - Incompressible
  - Compressible (non-viscous approximation)
  - Shallow water
  - Biphasic flows....

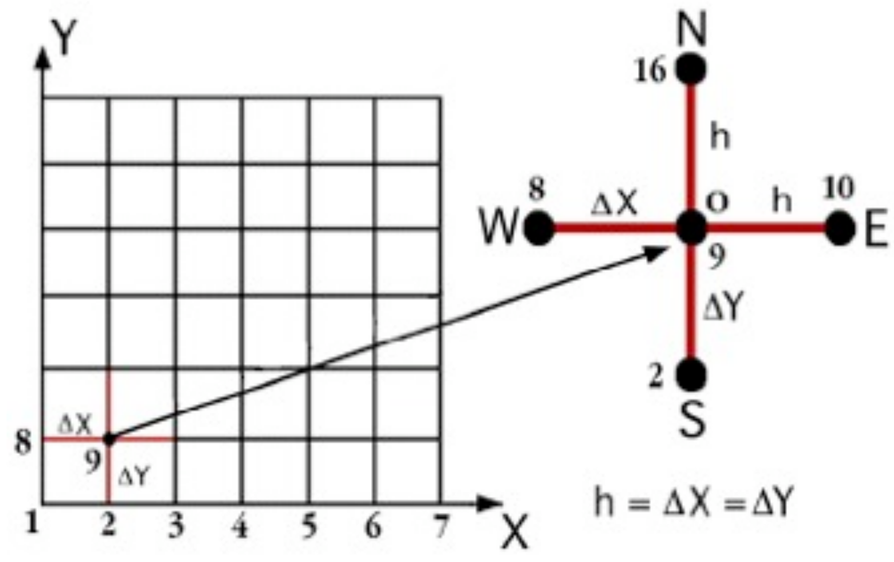
CFD - Introduction

- Earliest: Richardson (1910)
  - Human computers
  - Quickest averaged 2000 operations a week
- CFD development tied with computers!
  - 50s-60s: use of digital computers, finite difference methods
  - 70s: finite element methods, spectral methods
  - 80s: finite volume methods
  - 90s: application to diverse industries

CFD - Main discretization methods

- Finite difference

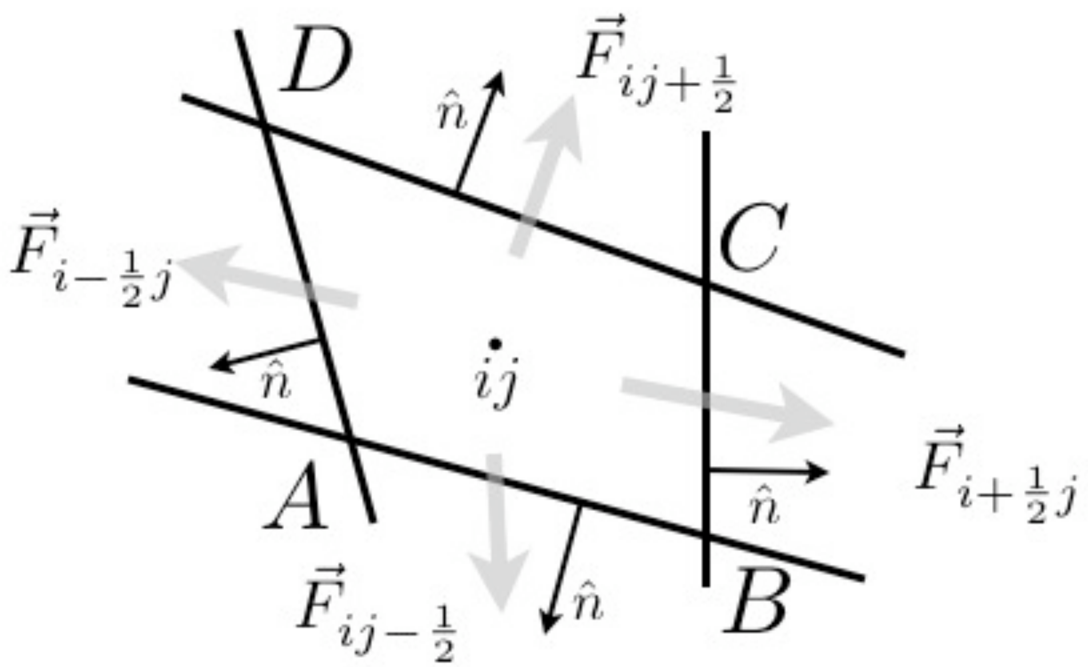
$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x}$$



- Finite volume

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} = 0$$

$$\frac{\partial}{\partial t} \int_{\Omega} \vec{U} \mathbf{d}\Omega + \oint_{\partial\Omega} \vec{F} \hat{n} \mathbf{ds} = 0$$



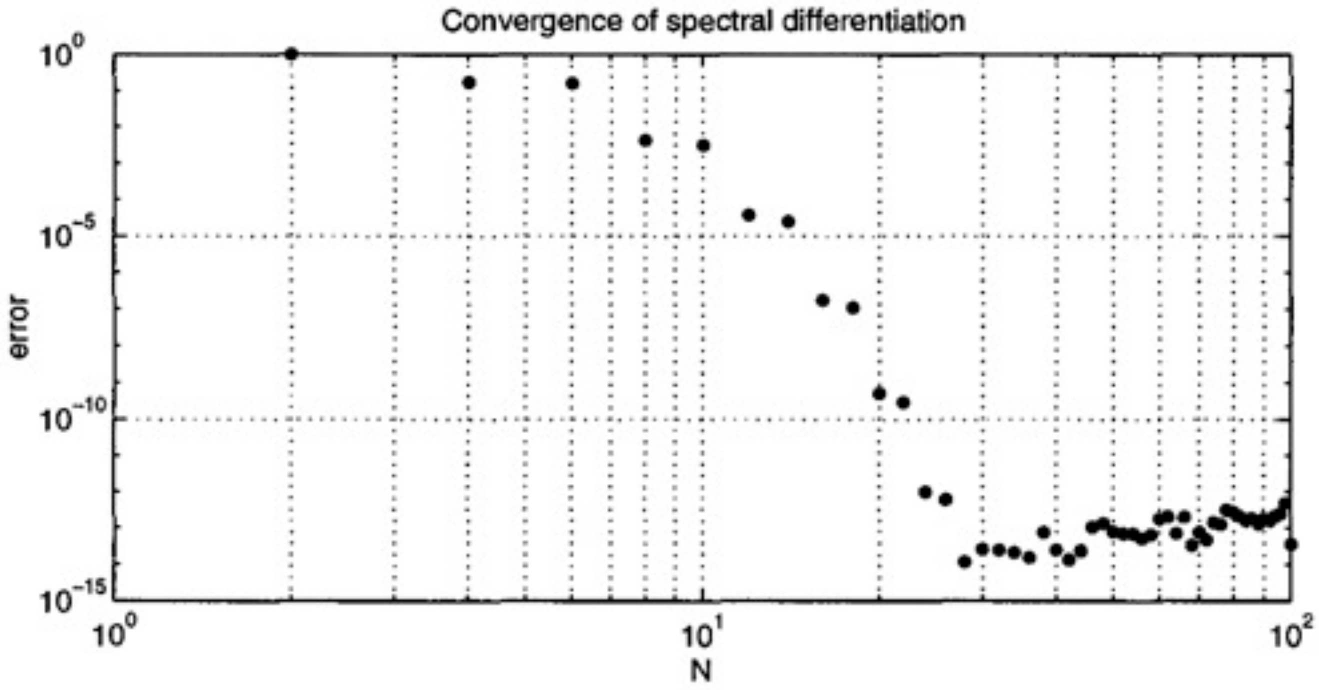
CFD - Main discretization methods

- Finite element method

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx.$$

- Spectral methods

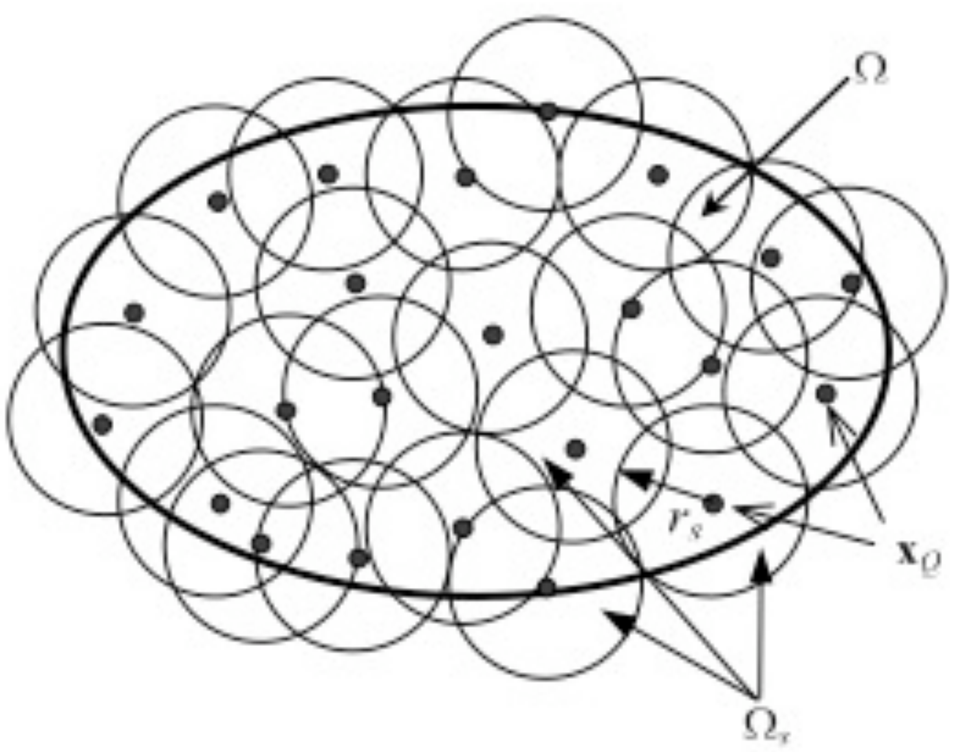
$$\widehat{\frac{\partial u}{\partial x}} = ik\hat{u}$$



CFD - Main discretization methods

- Mesh free methods
  - Smoothed Particle Hydrodynamics
  - Vortex methods
  - Radial Basis Functions
  - ...

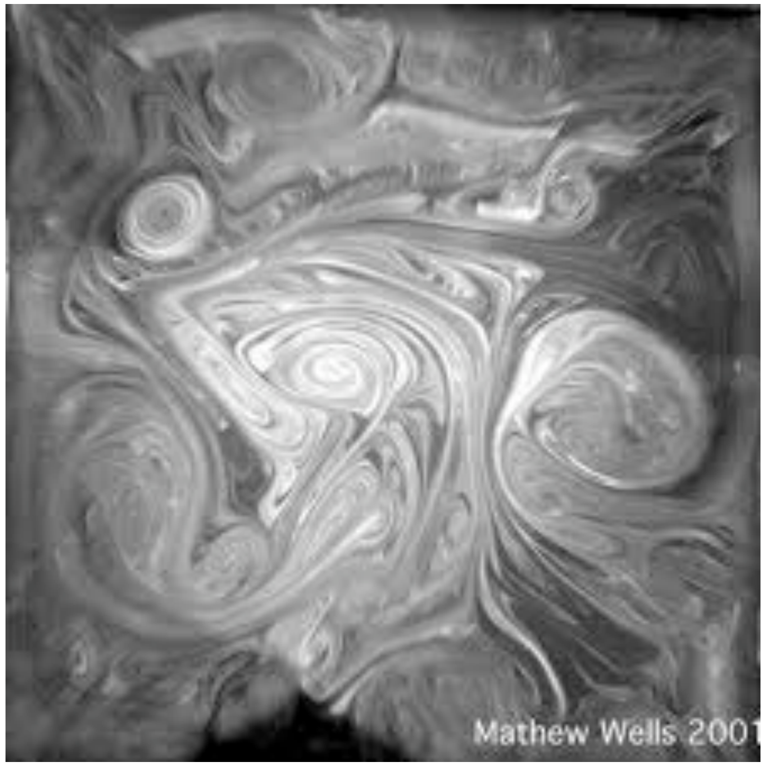
$$\frac{\partial u}{\partial x} = \sum_{i=0}^{N} \alpha_i \frac{\partial \phi_i}{\partial x}$$



CFD - Fluid Modeling

- Fluid flow is a multi-scale phenomena
  - We need Re^3 mesh points to reproduce all scales!
  - Turbulence modeling
  - Approximate turbulence effects

$$Re = \frac{VD}{\nu}$$



Conventional CFD

- Unstructured grids
  - Unstructured sparse matrices
- Incompressible $\nabla \cdot \mathbf{u} = 0$
  - Projection methods
- Implicit
  - Linear solvers
- Modeled turbulence
  - Reduced number of points

Conventional CFD

- CFD is a tough problem for the GPU:
  - Memory bound problems
- Also, needs to convince people
  - Old legacy codes
  - How to port old codes to the GPU?
- On the other hand, CFD codes are
  - SIMD
  - Single precision
  - Large data sets

Porting a code to GPU

- Option 1: accelerate the existing code
- Option 2: Rewrite code from scratch
- Option 3: Rethink algorithms

Potential acceleration

Next slides credits: J. Cohen - NVIDIA

Option 1: Accelerate existing code

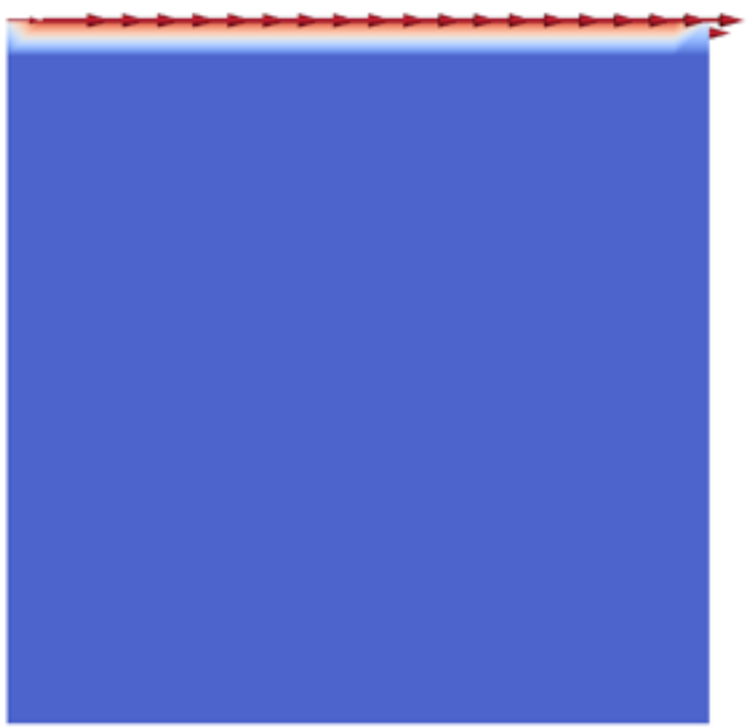
- Easiest way
- Probably not huge speedup
- Libraries like Cusp or CUFFT may be useful

Option 1: Accelerate existing code - SpeedIT

- SpeedIT (OpenFOAM)
  - Ported linear solvers to GPU
  - Supports multi-GPU

Mesh	Speedup
20x20	-100x
96x96x96	2.4x
128x128x32	2.0x

Xeon X5650 CPU
M2050 GPU



Ville Tossavainen (Seeinside Ltd.)

Option 1: Accelerate existing code - FEAST

- FEAST (Finite Element Analysis and Solution Tools)
  - High level abstraction approach
  - Isolate "accelerable" parts of code
  - Ports solver to GPU: Multigrid

Acceleration fraction: 75%
Local speedup: 11.5x
Global speedup: 3.8x

Opteron 2214 4 nodes CPU
GTX 8800 GPU

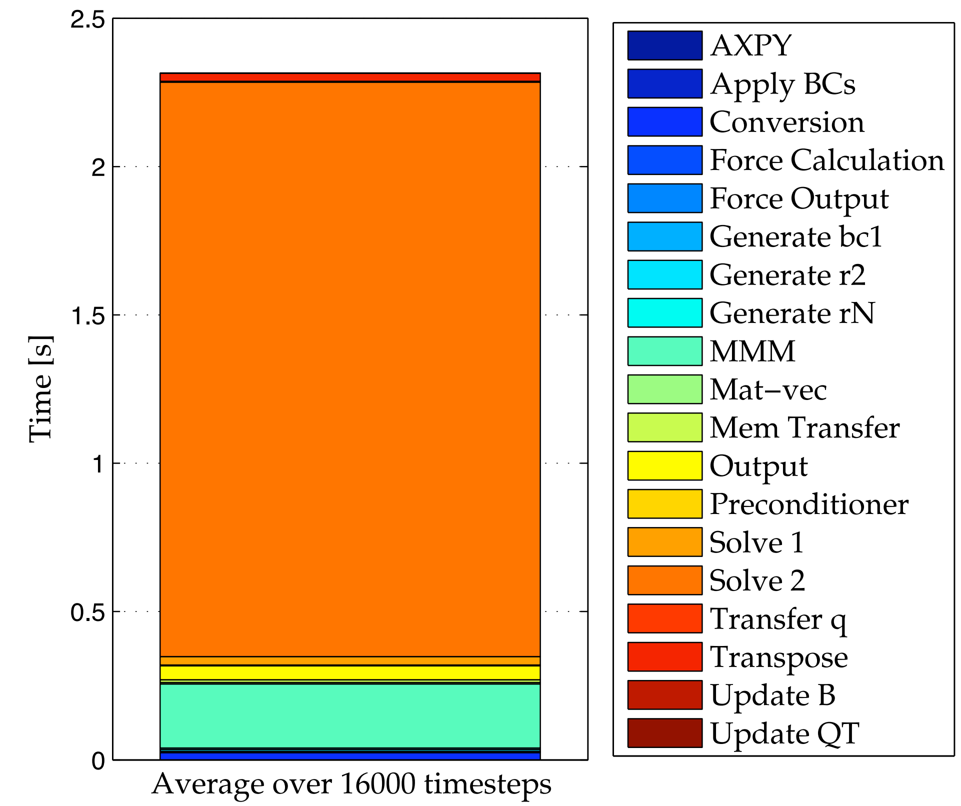
Strzodka, Goddeke, Behr (2009)

Option 2: Rewrite whole code

- First need to think about
  - What is the total application speedup that you can get
  - How does rewrite compare to accelerator approach
  - Good design
  - What global optimizations are possible

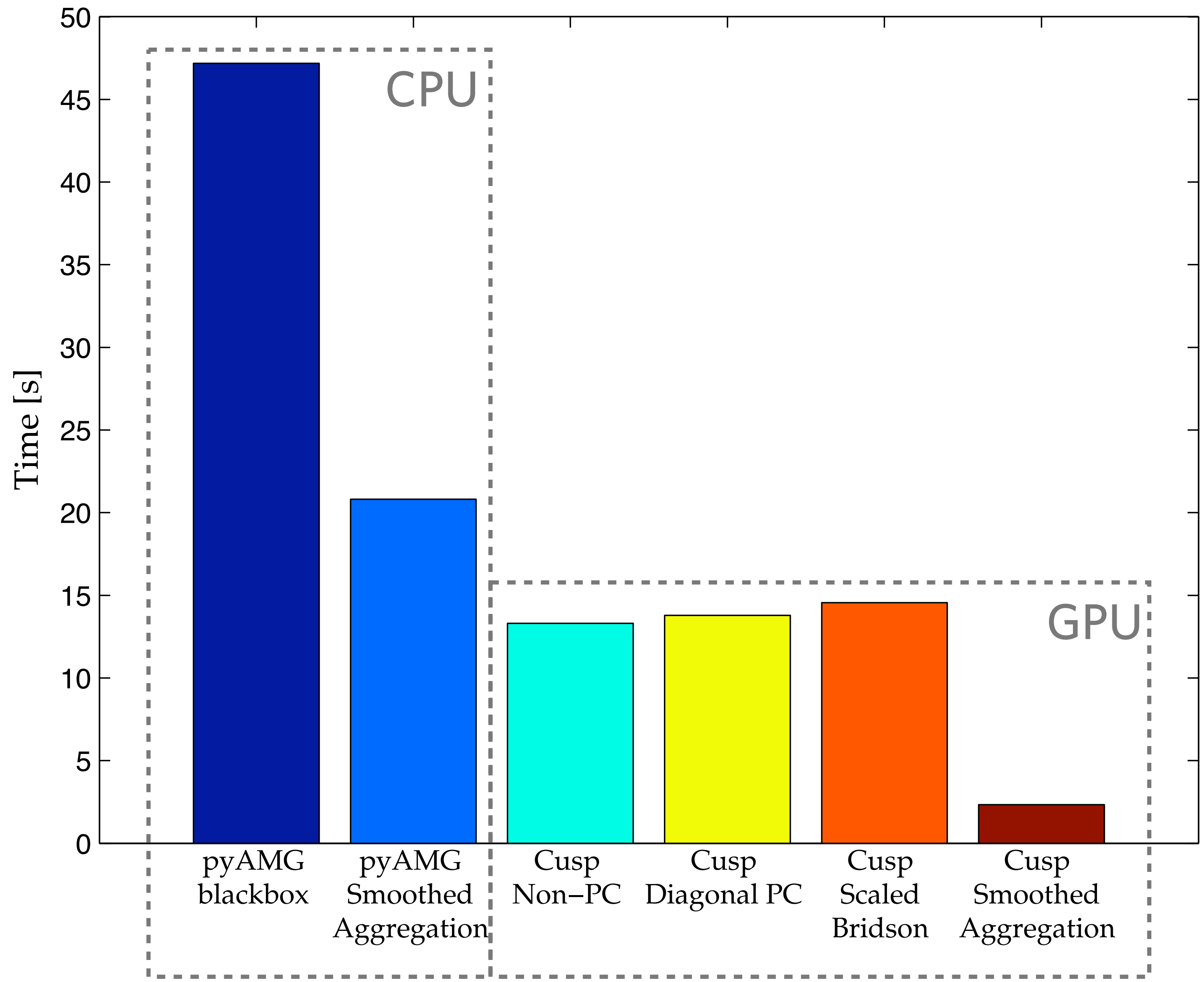
Option 2: Rewrite whole code - cuIBM

- Immersed Boundary Method on GPU (cuIBM)
  - Finite difference code with immersed boundary no slip condition
  - 2 linear systems: implicit diffusion and projection
  - Reported speedup: 7x



Layton, Krishnan, Barba 2011

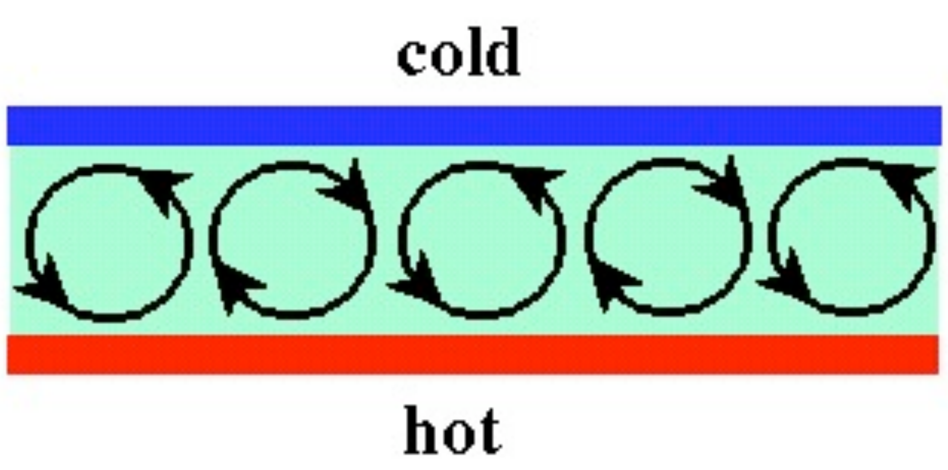
Option 2: Rewrite whole code - cuIBM



With good pre-conditioner, GPU is 9x faster, not much difference in other cases (best is 1.6x faster)

Option 2: Rewrite whole code - Open Current

- Developed by Jonathan Cohen in NVIDIA
- Compared a highly optimized CPU code and GPU code
  - CPU: Fortran, 8-core 2.5 GHz Xeon (8 thredas with MPI and OpenMP)
  - GPU: CUDA, Tesla C1060
- Solved the Rayleigh-Bernard with a finite difference code



Option 2: Rewrite whole code - Open Current

Resolution	CUDA time/step ms	Fortran time/step ms	Speedup
64x64x32	24	47	2.0x
128x128x64	79	327	4.1x
256x256x128	498	4070	8.2x
384x384x192	1616	13670	8.5x

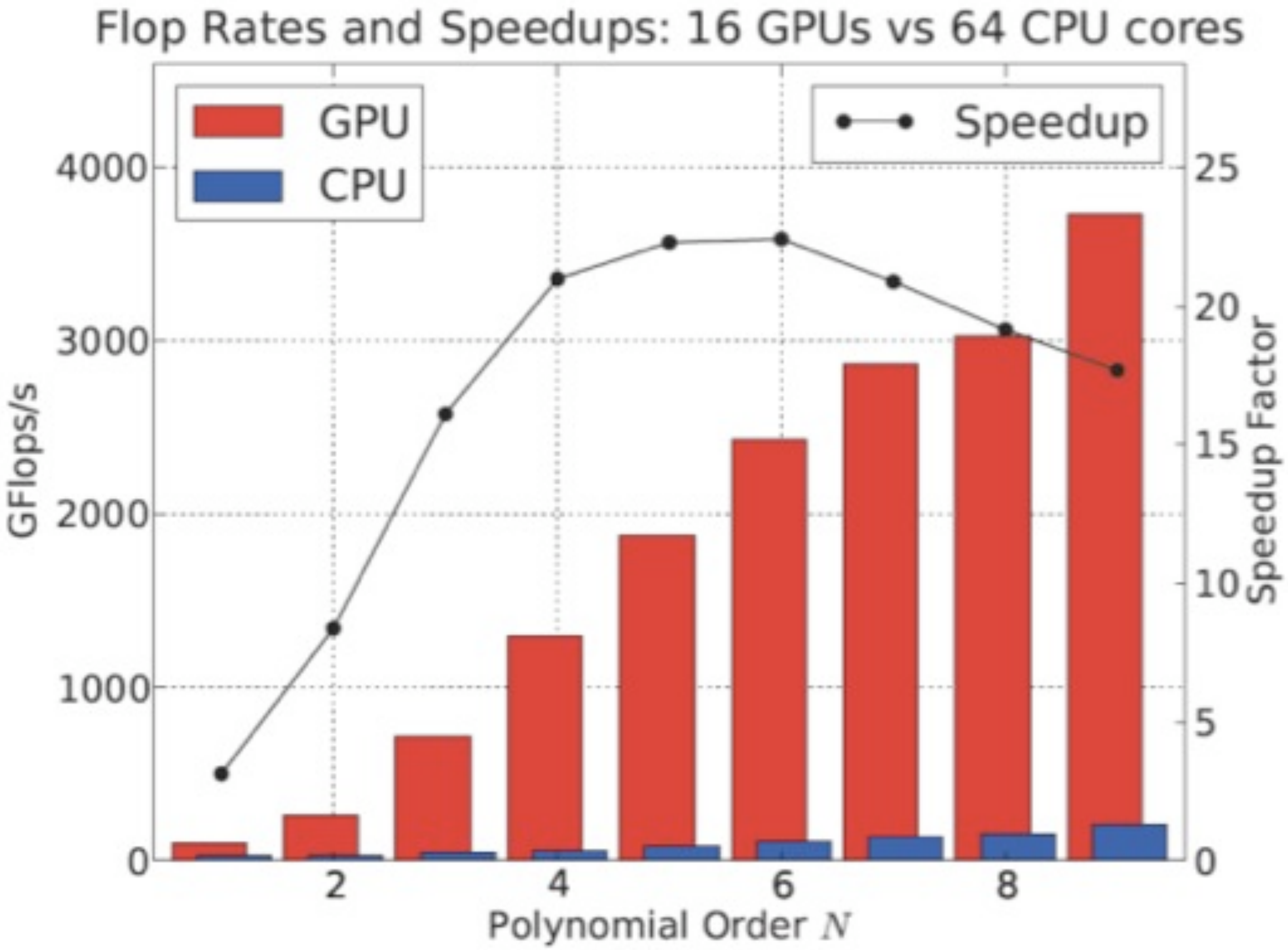
Option 3: Rethink numerical algorithms

- Most time consuming alternative!
- Maybe new architectures require new numerics
- Find methods that map well to the hardware
  - Maybe we overlooked something in the past because it was impractical

Option 3: Rethink numerical algorithms - DG

- Discontinuous Galerkin Methods
  - Arithmetically intensive
  - Mainly local
- Klockner et al. used DG to solve conservation laws

GPU T1060
CPU: Xeon E5472



Implicit heat equation solver

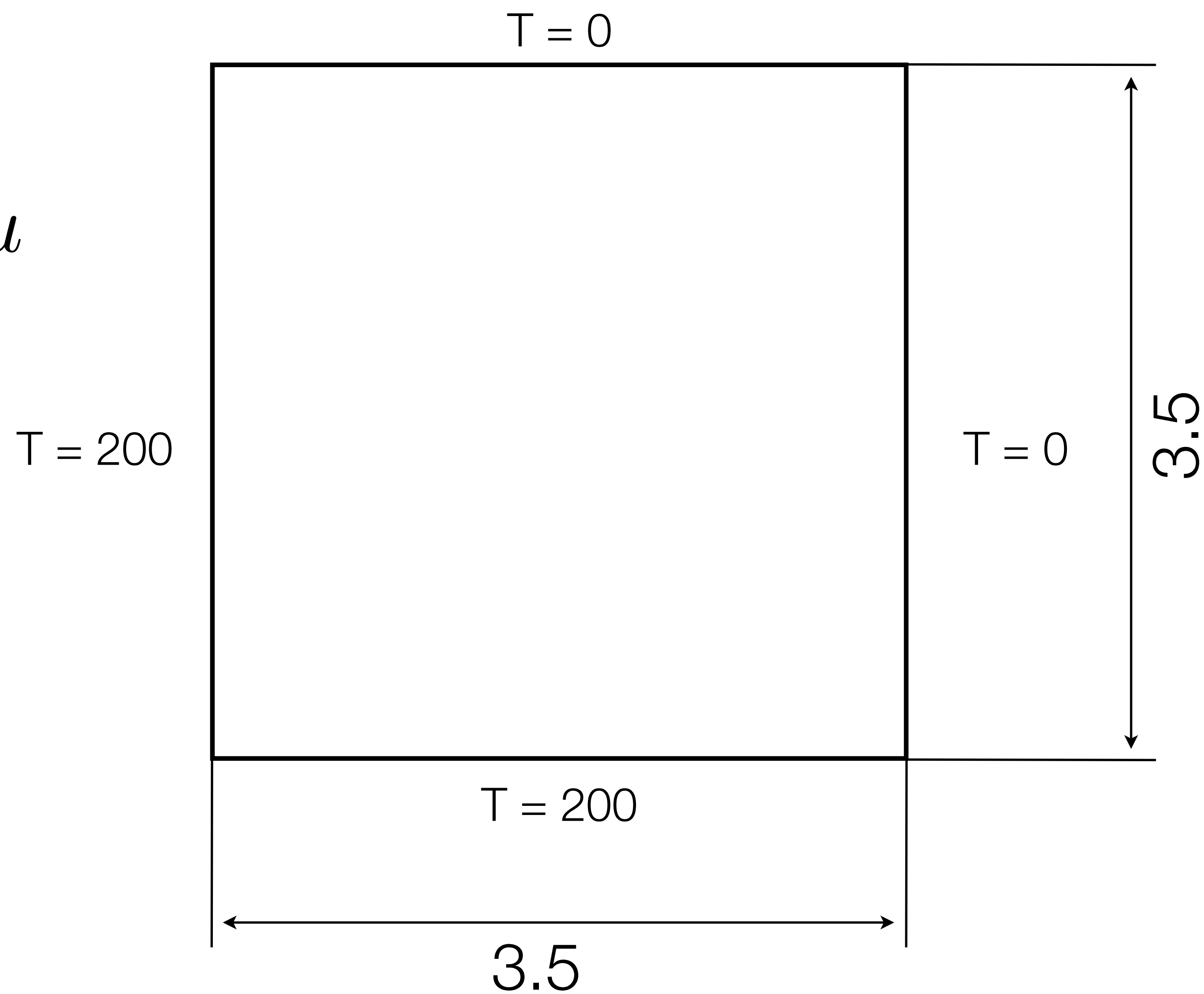
- Conventional CFD usually is dominated by Poisson type solvers
  - Projection methods
  - Implicit solvers to avoid stability constraints
- Heat equation with Crank-Nicolson $\frac{\partial u}{\partial t} = \alpha \nabla^2 u$
  - No stability constraint!

$$\frac{\alpha k}{h^2} < 0.5$$

Implicit heat equation solver

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u$$

α = 0.645
k = 1e-5
N = 128



$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{k} = \frac{\alpha}{2}\left(\frac{u_{i,j+1}^{n}+u_{i,j-1}^{n}+u_{i+1,j}^{n}+u_{i-1,j}^{n}-4u_{i,j}^{n}}{h^2} + \frac{u_{i,j+1}^{n+1}+u_{i,j-1}^{n+1}+u_{i+1,j}^{n+1}+u_{i-1,j}^{n+1}-4u_{i,j}^{n+1}}{h^2}\right)$$

Implicit heat equation solver

$$au_{i,j-1}^{n+1} + au_{i,j+1}^{n+1} + au_{i-1,j}^{n+1} + au_{i+1,j}^{n+1} + bu_{i,j}^{n+1} = RHS_{i,j}$$

$$a = -\frac{\alpha k}{2h^2} \qquad\qquad b = 1 - 4a = 1 + \frac{2\alpha k}{h^2}$$

$$RHS_{i,j} = u_{i,j}^{n} + \frac{\alpha k}{2h^2}\left(u_{i,j-1}^{n} + u_{i,j+1}^{n} + u_{i-1,j}^{n} + u_{i+1,j}^{n} - 4u_{i,j}^{n}\right) - BC^{n+1}$$

Implicit heat equation solver

$$[A]\mathbf{u}^{n+1} = \mathbf{RHS}$$

$$[A] = I + \frac{\alpha k}{2h^2} \cdot [\text{Poisson}]$$

$[A]$ size (N-2)^2 x (N-2)^2

$\mathbf{RHS}$ size (N-2)^2

$\mathbf{u}^{n+1}$ size (N-2)^2