Implementing Fast Parallel Linear System Solvers In OpenFOAM based on CUDA

Daniel P. Combest and Dr. P.A. Ramachandran
and Dr. M.P. Dudukovic

Chemical Reaction Engineering Laboratory (CREL)
Department of Energy, Environmental, and Chemical Engineering. Washington University, St. Louis, MO.

Optimization, HPC, and Pre- and Post-Processing I Session.
6th OpenFOAM Workshop Penn State University. June 15th 2011

Objectives

Objective

- To introduce a tool to integrate linear system solvers on the GPU into OpenFOAM

Topics

- Introduction to the GPU and CUDA
- Integrating CUSP into OpenFOAM
- Some preliminary results
- Important considerations when using this tool
- Next steps
- Take home messages

Introduction to The GPU and CUDA

What exactly is CUDA?
Defined as: Compute Unified Device Architecture. I.e. a parallel computing architecture used in graphics processing units (GPU), developed by Nvidia.

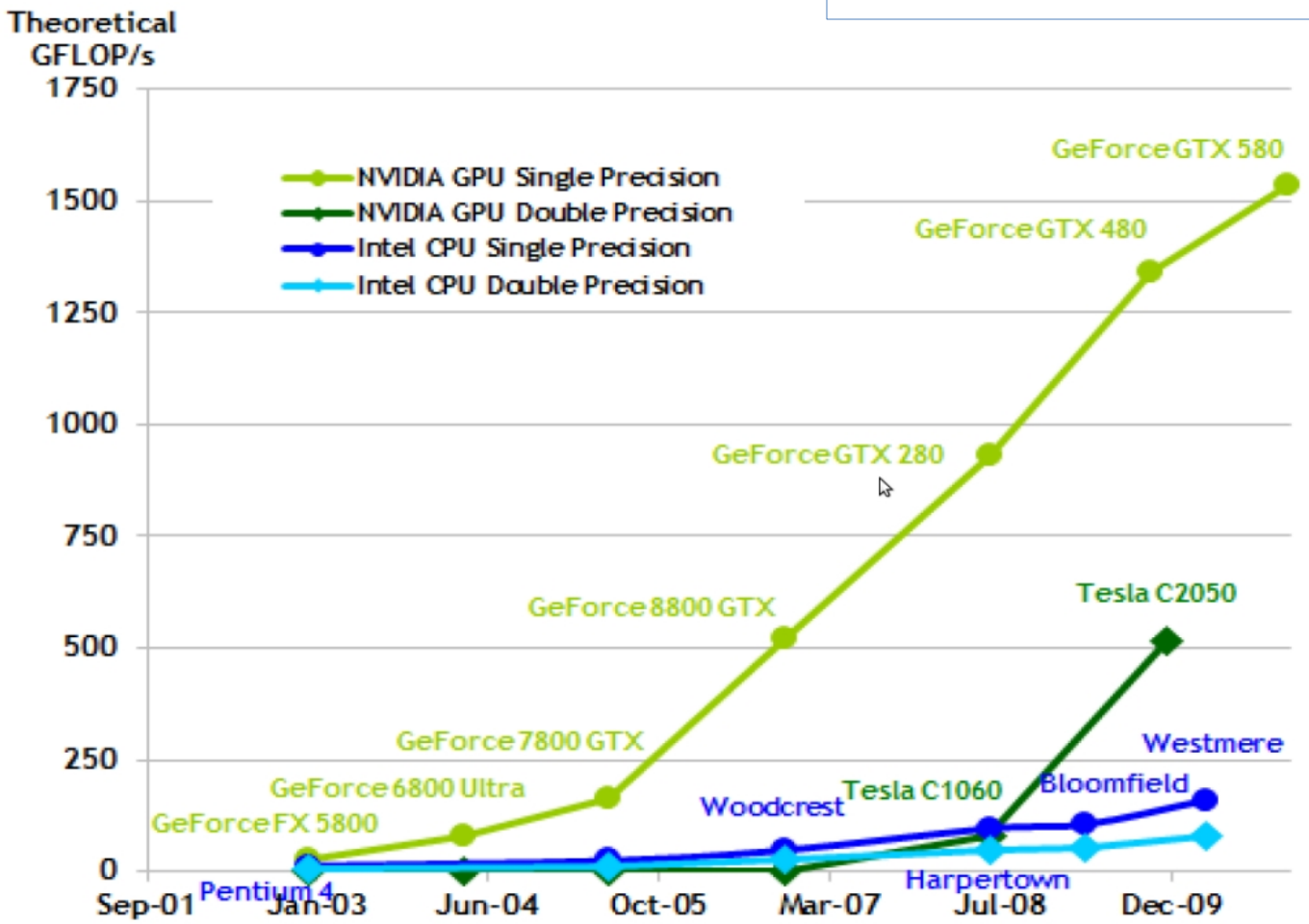
Introduction to The GPU and CUDA

What exactly is CUDA?
Defined as: Compute Unified Device Architecture. I.e. a parallel computing architecture used in graphics processing units (GPU), developed by Nvidia.

What is CUDA C/C++?
A language that provides an interface so that parallel algorithms can be run on CUDA enabled Nvidia GPUs

Introduction to The GPU and CUDA

GPU v.s CPU Calculations



CPU-GPU Comparison of Floating-point operations per second [1]

Introduction to The GPU and CUDA

Why are we interested?

Larger problems require more computing resources (LES, coupled physics)

GPUs are fast when used properly

They are relatively cheap

Introduction to The GPU and CUDA

Why are we interested?

Larger problems require more computing resources (LES, coupled physics)

GPUs are fast when used properly

They are relatively cheap

Where can GPUs be applied?

Where parallel algorithms live

- Linear algebra i.e. sparse matrix math

Introduction to The GPU and CUDA

Why are we interested?

Larger problems require more computing resources (LES, coupled physics)

GPUs are fast when used properly

They are relatively cheap

Where can GPUs be applied?

Where parallel algorithms live

- Linear algebra i.e. sparse matrix math

Why don't we compile everything to work on the GPU?

Only programs written in CUDA language can be parallelized on GPU. **So we cannot just recompile OF.**

Integrating CUSP into OpenFOAM

cusp-Library http://code.google.com/p/cusp-library/

“Cusp is a library for **sparse linear algebra** and **graph** computations on CUDA. Cusp provides a flexible, **high-level** interface for manipulating sparse matrices and solving sparse linear systems.”[2]

Provided Template Solvers:

- (Bi-) Conjugate Gradient (-Stabilized)
- GMRES

Matrix Storage

- CSR, COO, HYB, DIA

Provided Preconditioners

- Jacobi (diagonal) preconditioners
- Sparse Approximate inverse preconditioner
- Smoothed-Aggregation Algebraic Multigrid preconditioner

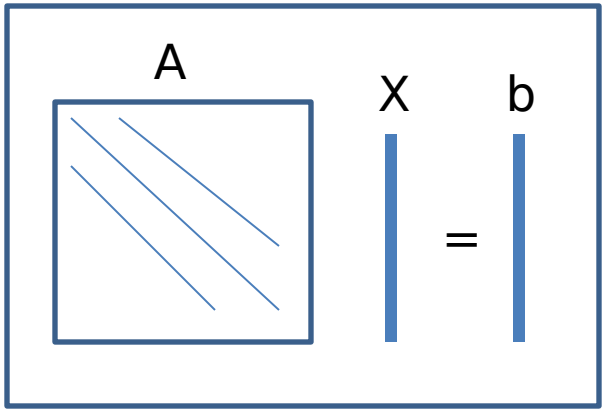
Integrating CUSP into OpenFOAM

http://code.google.com/p/thrust/

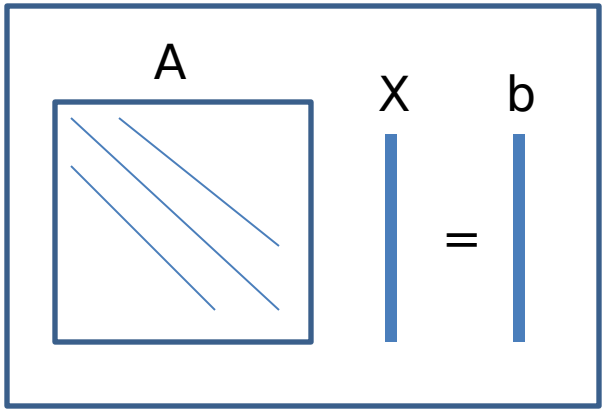
“Thrust is a CUDA library of parallel algorithms with an interface resembling the C++ Standard Template Library (STL). Thrust provides a flexible **high-level**interface for GPU programming that greatly enhances developer **productivity**.
“ [3]

Integrating CUSP into OpenFOAM

OpenFOAM solve(…);



Cusp-based solver on GPU

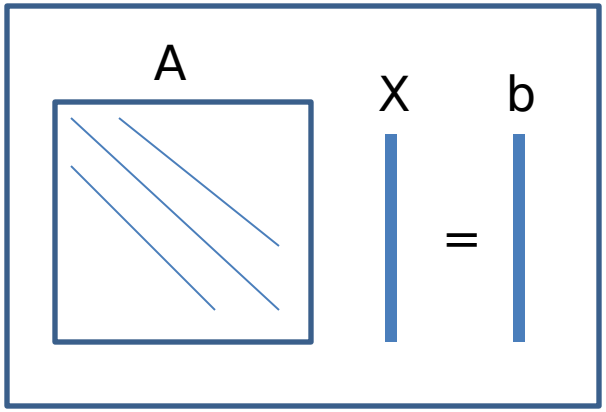


Thrust Methods

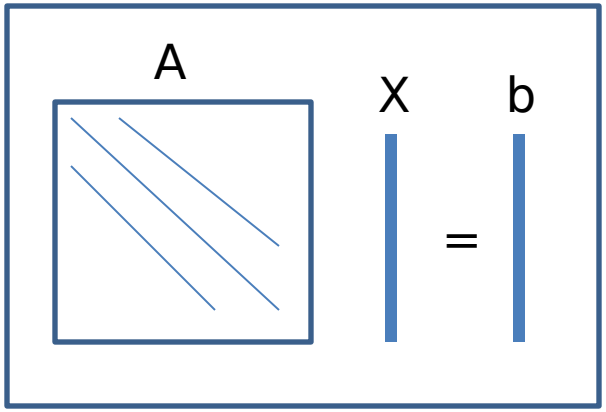
cusp Methods

Integrating CUSP into OpenFOAM

OpenFOAM solve(…);



Cusp-based solver on GPU



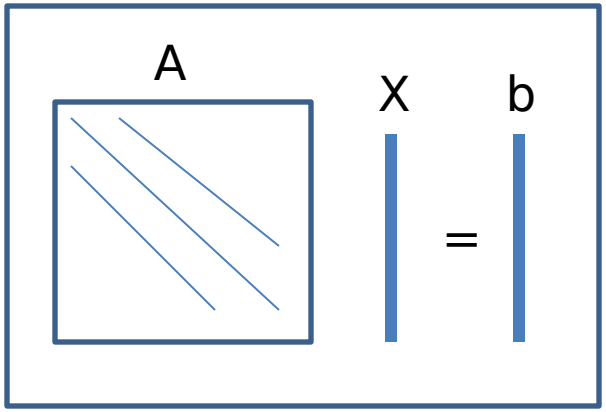
Thrust Methods

lduMatrix is converted to COO Using thrust::copy() in C++

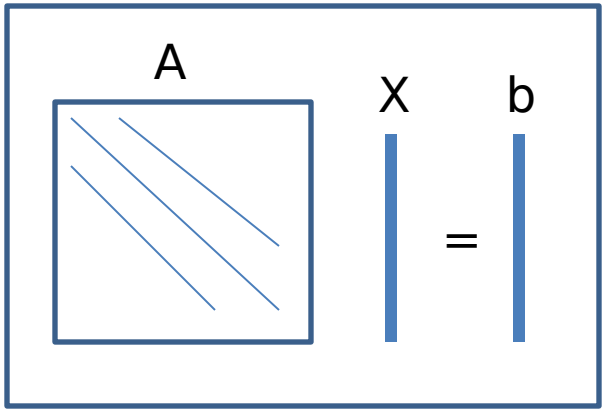
cusp Methods

Integrating CUSP into OpenFOAM

OpenFOAM solve(…);



Cusp-based solver on GPU



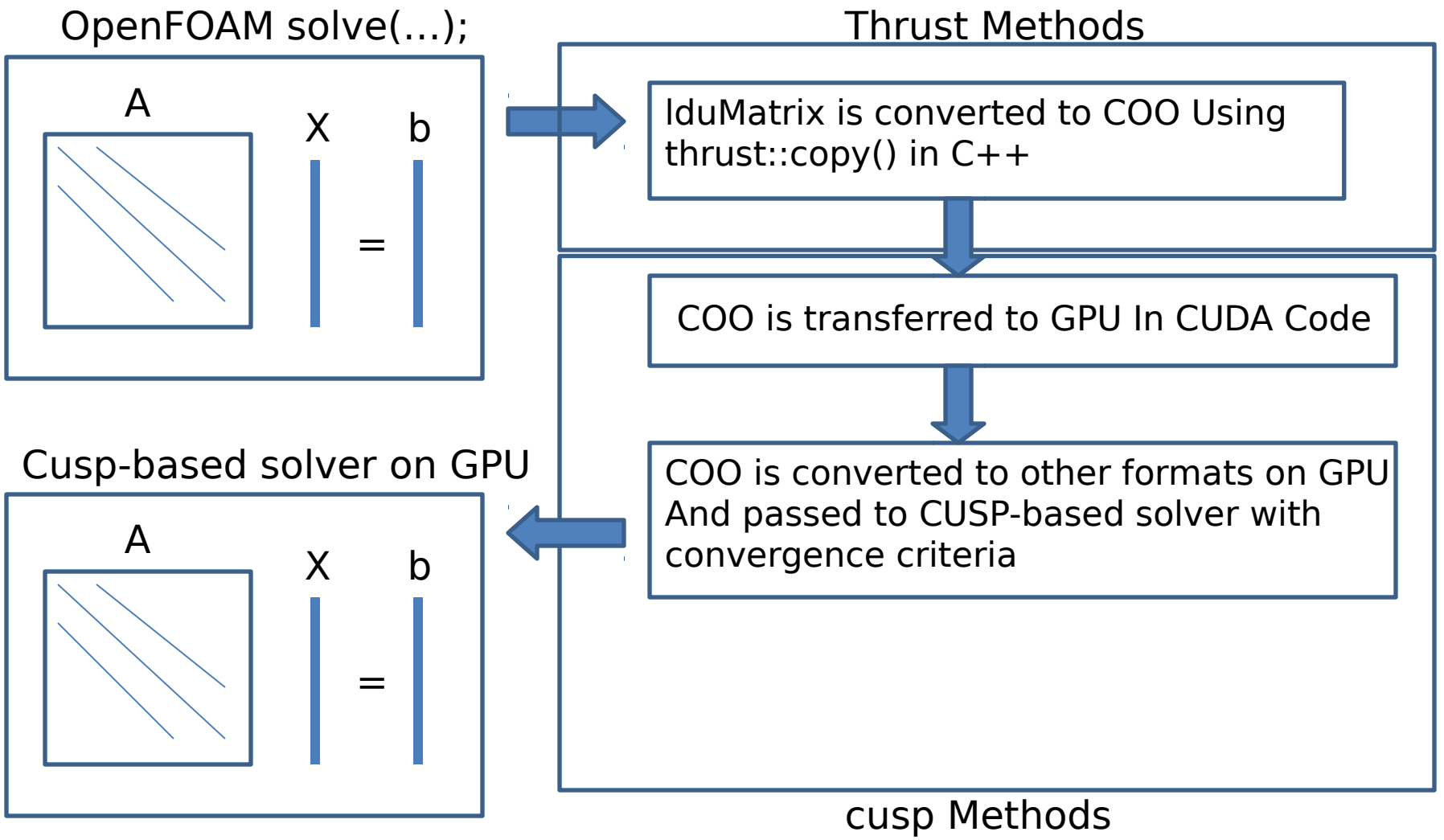
Thrust Methods

lduMatrix is converted to COO Using thrust::copy() in C++

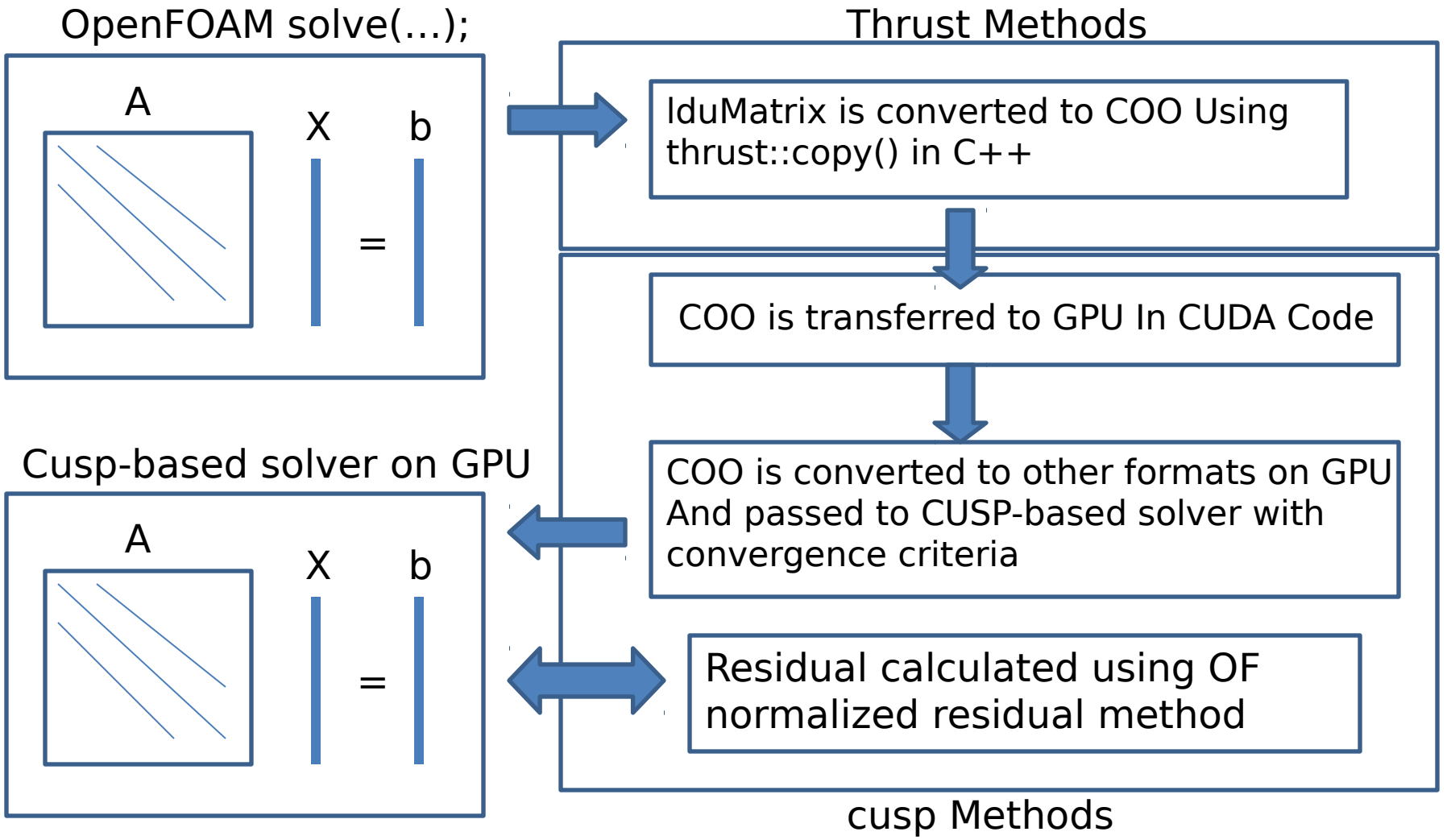
COO is transferred to GPU In CUDA Code

cusp Methods

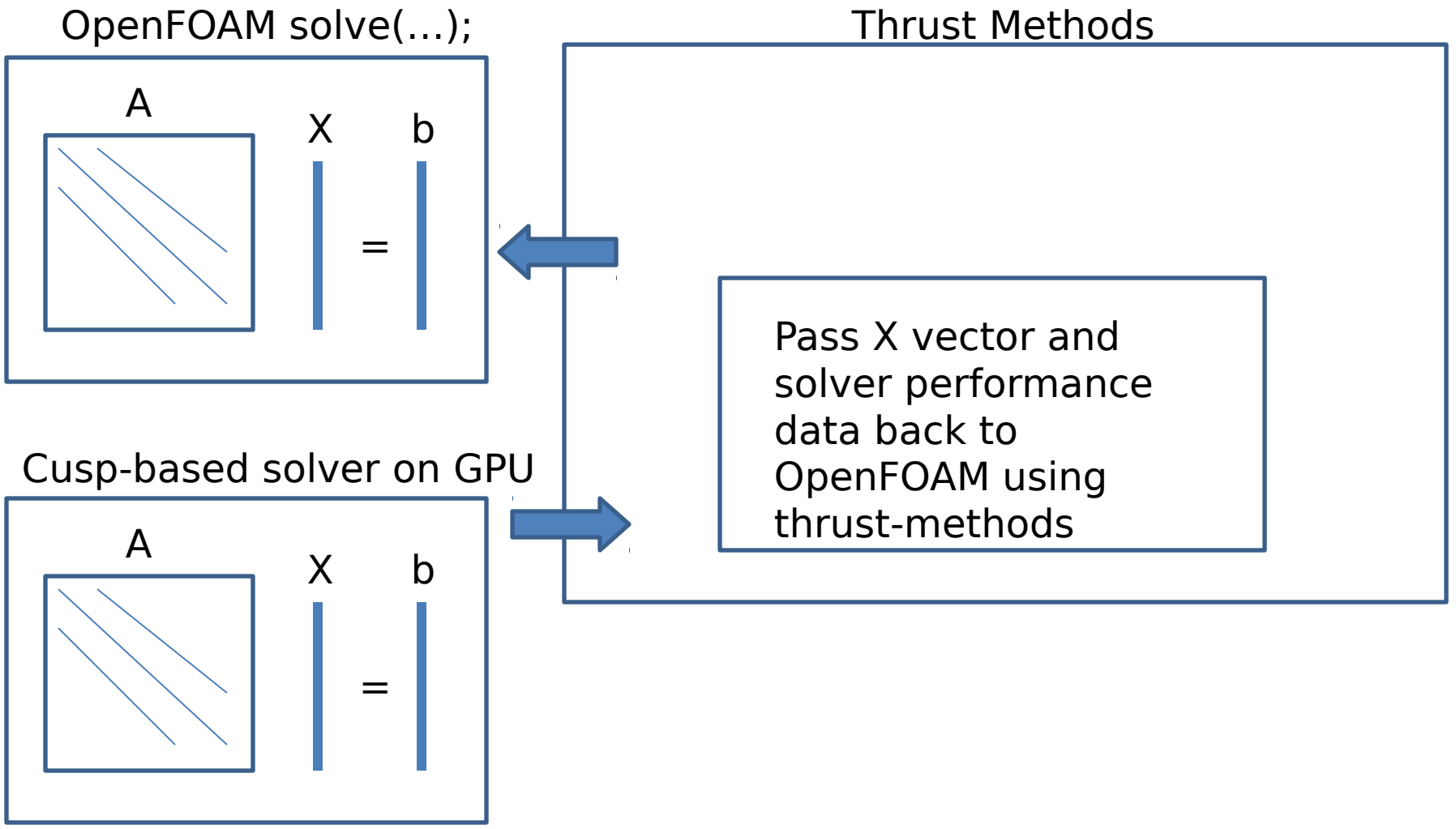
Integrating CUSP into OpenFOAM



Integrating CUSP into OpenFOAM



Integrating CUSP into OpenFOAM



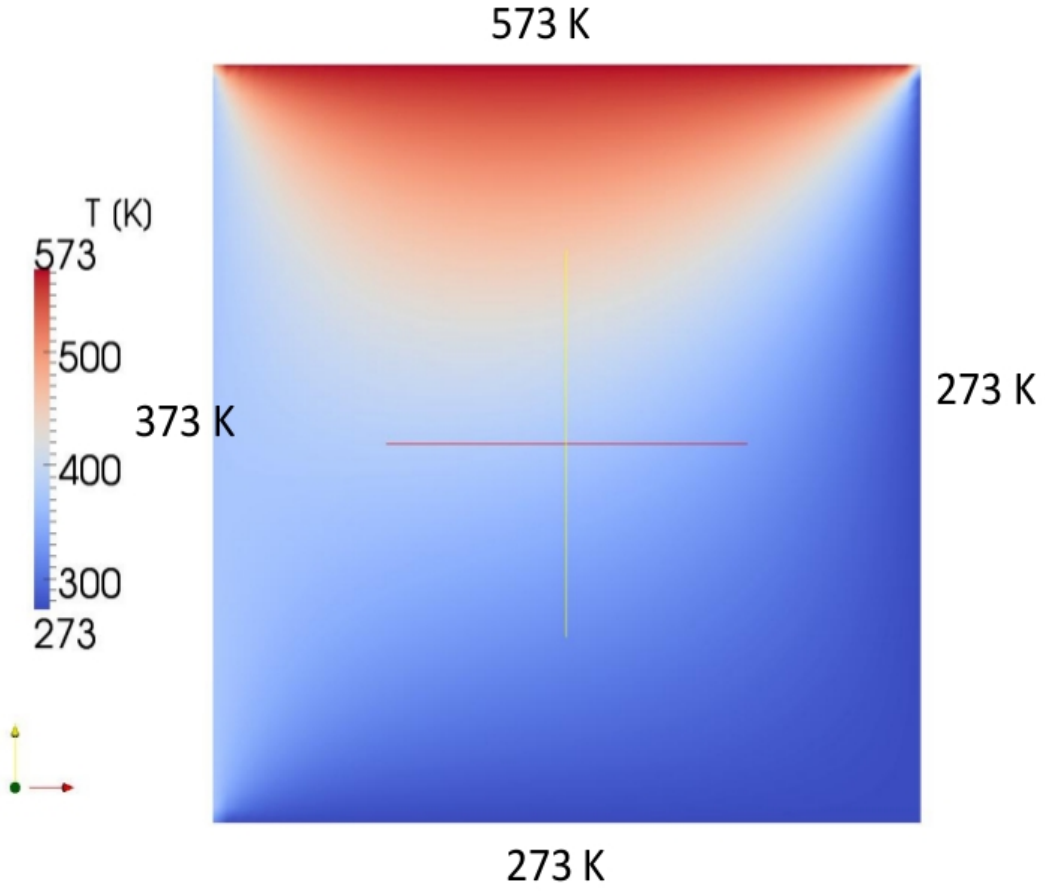
Preliminary Results

A test Problem.

2D Heat Equation

$$\nabla^2 T = 0$$

Vary N from 10-2000
where N^2 = nCells



Preliminary Results

Solver Settings

All CG solvers

Tolerance = 1e-10;
MaxIter 1000;

```
solver          GAMG;
tolerance       1e-10;
smoother        GaussSeidel;
nPreSweeps      0;
nPostSweeps     2;
cacheAgglomeration true;
nCellsInCoarsestLevel sqrt(nCells);
agglomerator    faceAreaPair;
mergeLevels     1;
```

Preliminary Results

Setup

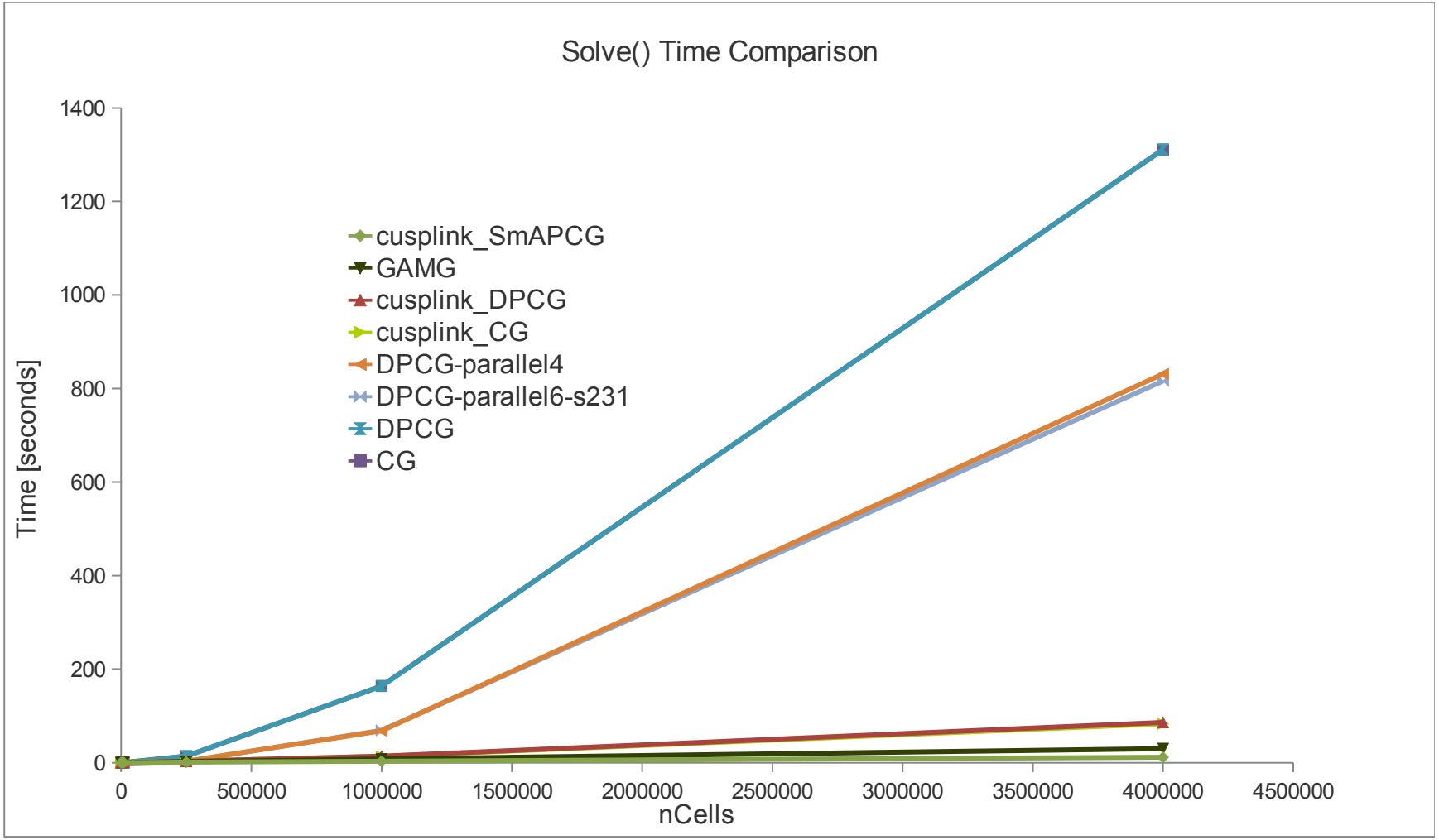
CUDA version 4.0
CUSP version 0.2
Thrust version 1.4
Ubuntu 10.04

CPU: Dual Intel Xeon Quad Core E5430 2.66GHz
Motherboard: Tyan S5396
RAM: 24 gig

GPU: Tesla C2050 3GB DDR5
515 Gflops peak double precision
1.03 Tflops Peak single precision
14 MP * 32 cores/MP = 448 cores
Host-device memory bw = 1566 MB/sec (Motherboard specific)

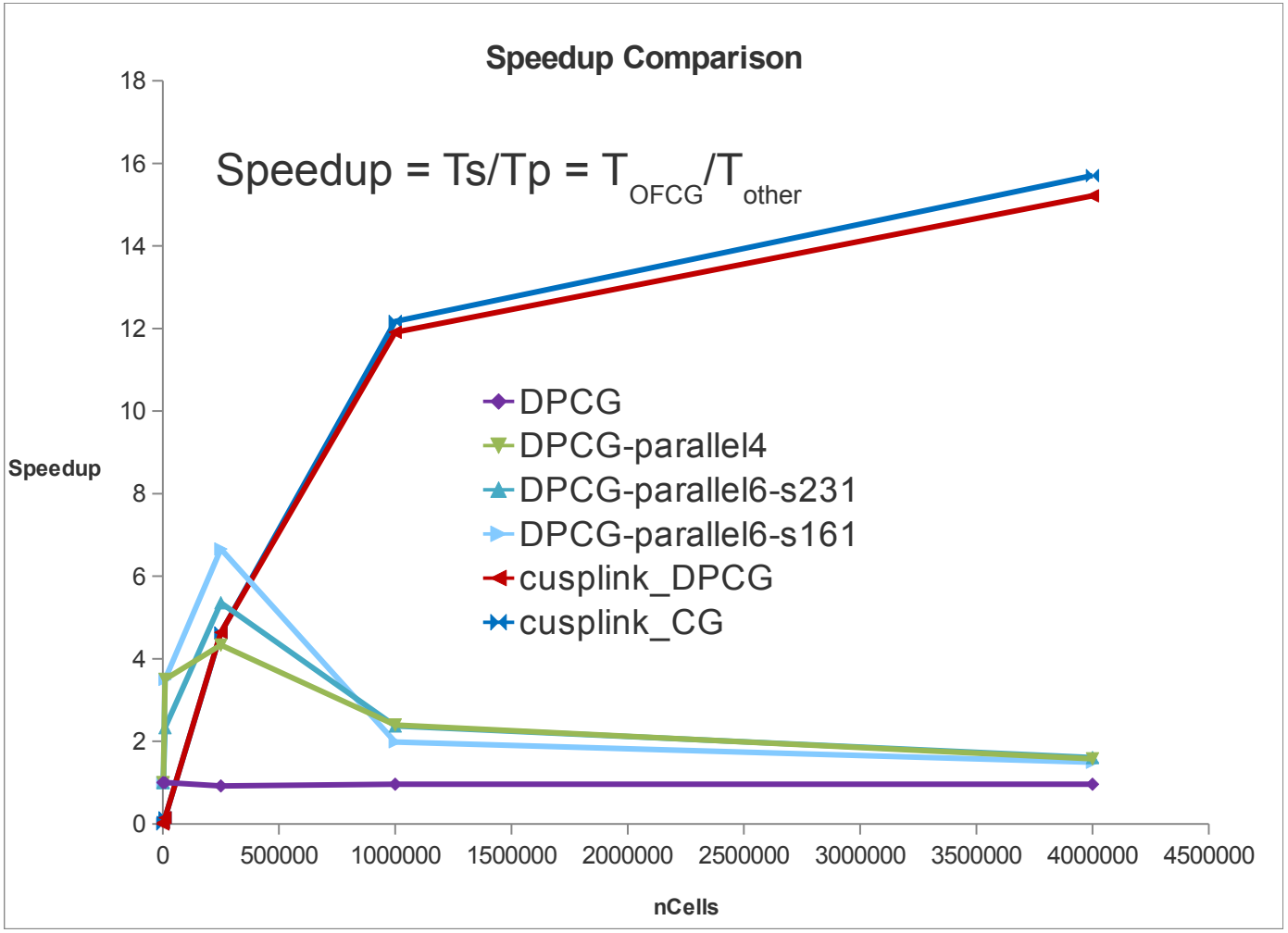
Preliminary Results

Solve Time



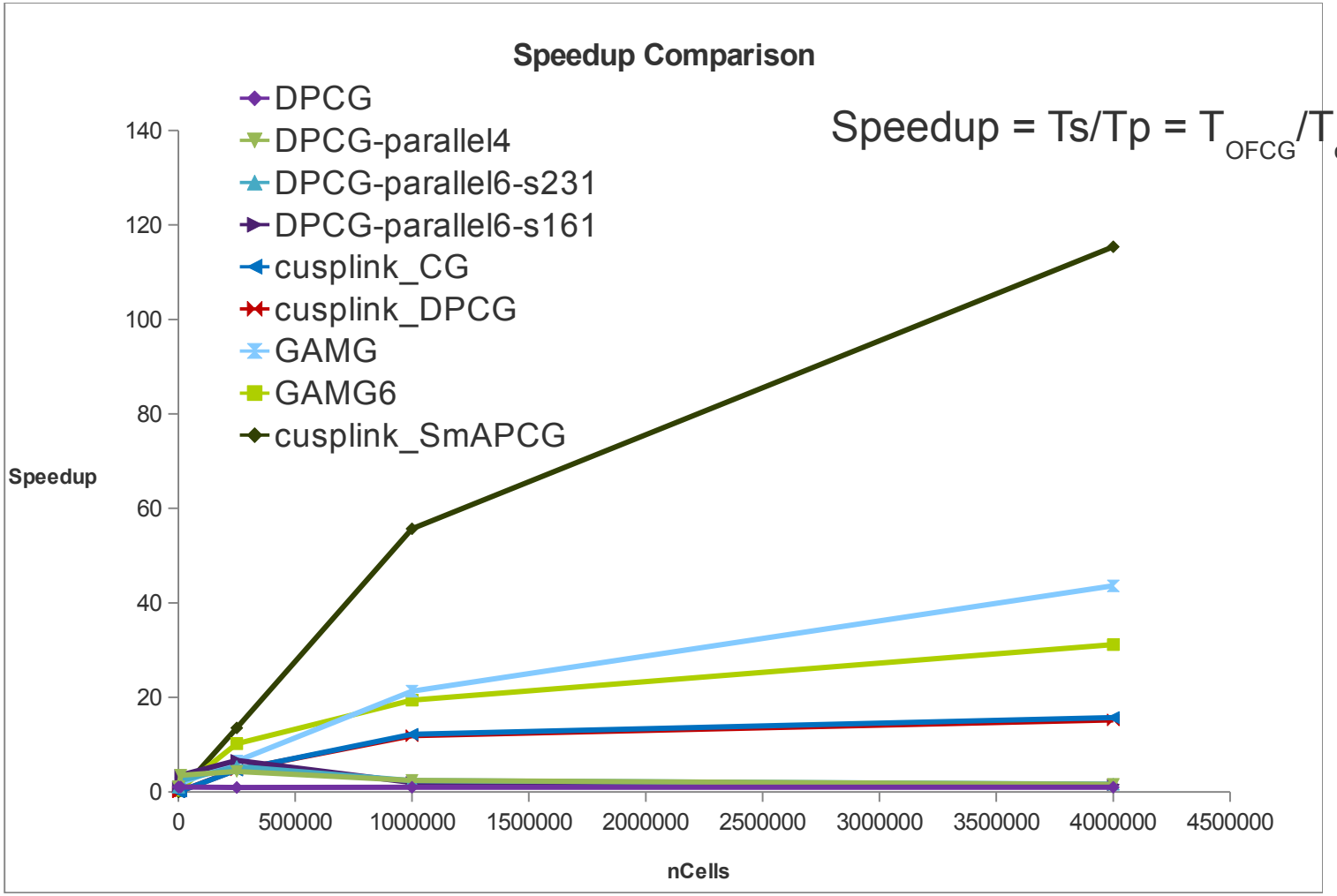
Preliminary Results

Solution Speedup



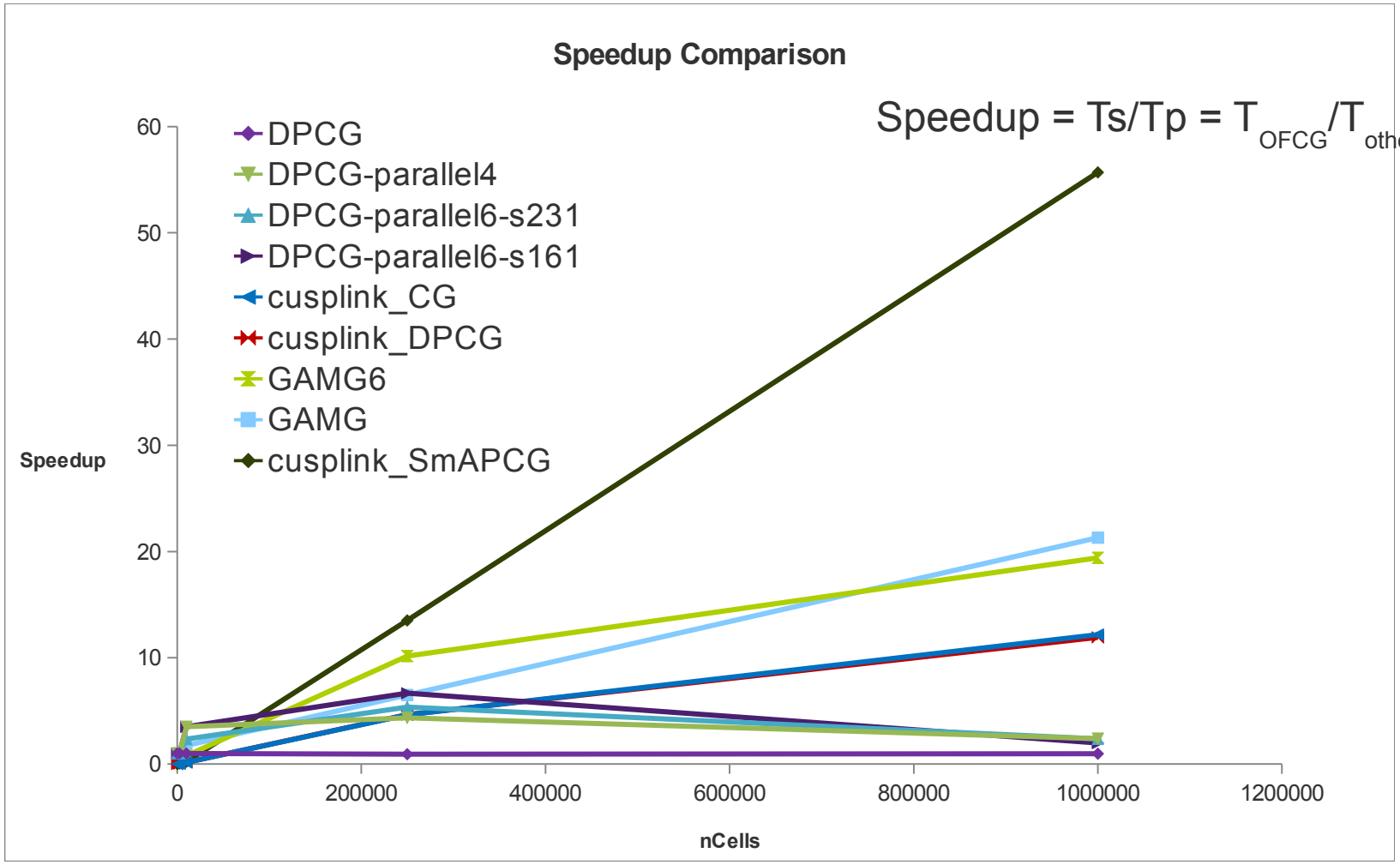
Preliminary Results

Solution Speedup



Preliminary Results

Solution Speedup



Preliminary Results

Notes on Comparison

- GPU Tesla C2050
- All are double precision results in CSR
- Low precision timers used
- Parallel CPU results a little skewed, i.e. other processes occurring on cpu
- Residuals are scaled in the same manner as OF
- GPU Results include data transfer time
- The GPU smoothed aggregate preconditioner was 115x faster than CG

Important Considerations

Before you jump….

- Tool is not multi GPU yet.
- Proper motherboard and power source
- CUDA Enabled hardware
- Amount of System Ram >= GPU RAM
- Working install of CUDA, CUSP, and OF-ext

Next Steps

Where to Start?

- Finish asymmetric solvers
- Clean up current version and release code

- Multi GPU support
- Improve the solvers as CUSP improves

Take Home Messages

- The GPU only solves the Ax=b system
- We have double precision
- GPUs have been integrated into OpenFOAM using Thrust and CUSP
- As cusp and thrust improve, nothing needs to be changed in this code, only to update cusp and thrust.
- They have been shown to be faster in the cases provided, because it is mostly solving Ax = b.
- Residuals are calculated the same as in OpenFOAM
- Multi-GPU still needs attention.
- The results show that memory bandwidth still is an issue with this particular setup and results could be faster with other setup.

Acknowledgements

Funding and Support

Nvidia Professor Partnership Program

Chemical Reaction Engineering Laboratory (CREL) MRE Fund (http://crelonweb.eec.wustl.edu/)

OpenFOAM Developers Community

Advisors

Dr. Ramachandran

Dr. Dudukovic

Sources

1. Nvidia CUDA Programming Guide, Version 4.0, 2011. Nvidia Corporation.
2. Nathan Bell and Michael Garland, Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations, 2010, http://cusp-library.googlecode.com,Version 0.1.0
3. Jared Hoberock and Nathan Bell, Thrust: A Parallel Template Library, 2010, http://www.meganewtons.com/,Version 1.3.0

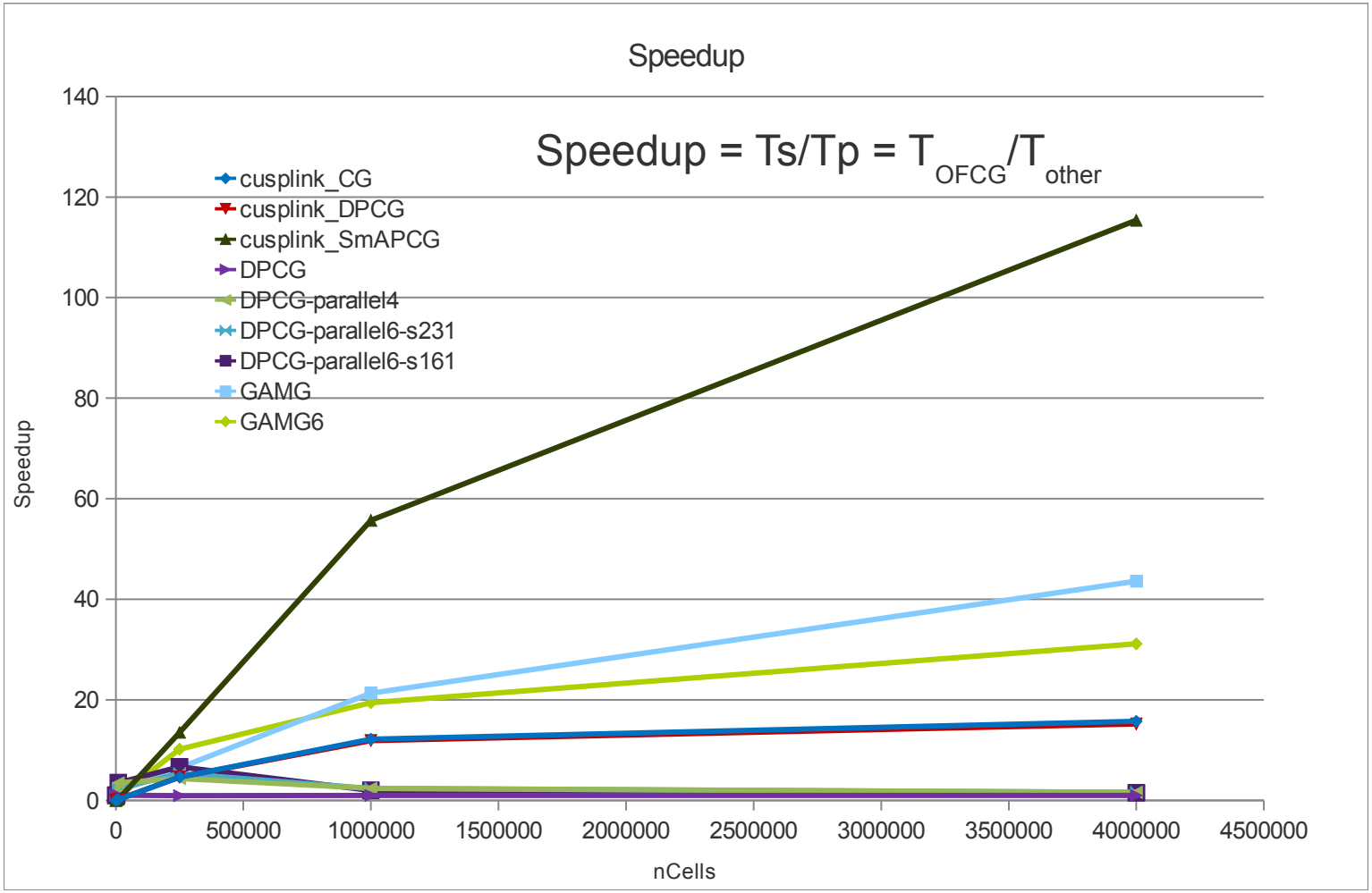
Thanks for your attention!

Questions?

Contact Info:
Dan Combest
dcombest@seas.wustl.edu

Preliminary Results

Solution Speedup



Residual Scaling

For matrix A x = b,

residual is defined as

res = b - Ax

We then apply residual scaling with the following normalisation factor procedure:

Type xRef = gAverage(x);

wA = A x;
pA = A xRef;

NormFactor = gSum(cmptMag(wA - pA) + cmptMag(source - pA)) + matrix.small_;

and the scaled residual is:

residual = gSum(cmptMag(source - wA))/normFactor;

I will save you from complications with vectors and tensors in my block solver. :-)

Enjoy,

Hrv

Source: http://www.cfd-online.com/Forums/openfoam-solving/57903-residuals-convergence-segregated-solvers.html