The Use of GPUs for Reservoir Simulation

Mark Wakefield

ECLIPSE Development Manager,

Schlumberger

Contributors:

Garf Bowen & Arnaud Desitter, Schlumberger

John Appleyard & Jeremy Appleyard, Polyhedron Software

Schlumberger

Schlumberger Abingdon Tech. Center

Based 8 miles from Oxford, UK

- ~200 employees involved in developing oilfield software
- ~50 people involved in commercial simulator development
  - ECLIPSE*: Established FORTRAN/MPI code focusing on high end physics
  - INTERSECT*: New to market C++/MPI code focusing on scalability and large model workflows

GPU Technology Evaluation

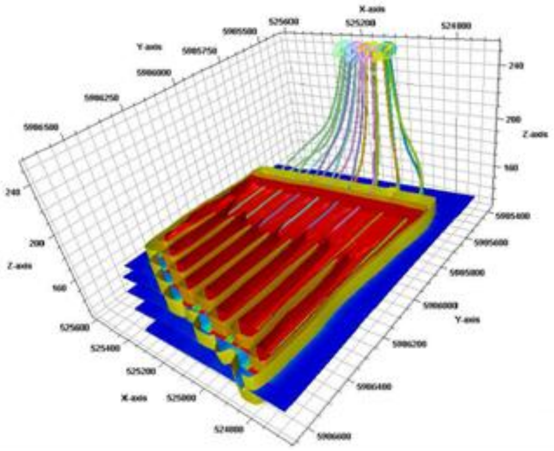

Have been evaluating GPU technology for the past 2+ years, particularly the NVIDIA CUDA architecture

1. How can GPUs be used in existing commercial products

2. Considerations for algorithm design

3. Considerations for software engineering

Simulator Overview



- Evolutionary, timesteps from an initial state
- Solves a set of non-linear conservation equations
- Solved implicitly due to fast propagating pressure transients
- Newton iteration requires solution of linear system
- Solution of linear system is critical to performance but not naturally parallel
- Models routinely have 10^4 – 10^7 grid blocks

Timestep Loop to Find Solution x at t + Δt

1) Property calculations from x

2) Residual and Jacobian assembly

$$r_{comp}(x) = \frac{\partial c_{comp}}{\partial t}(x) - f_{comp}(x) - w_{comp}(x)$$

Accumulation Flux Source

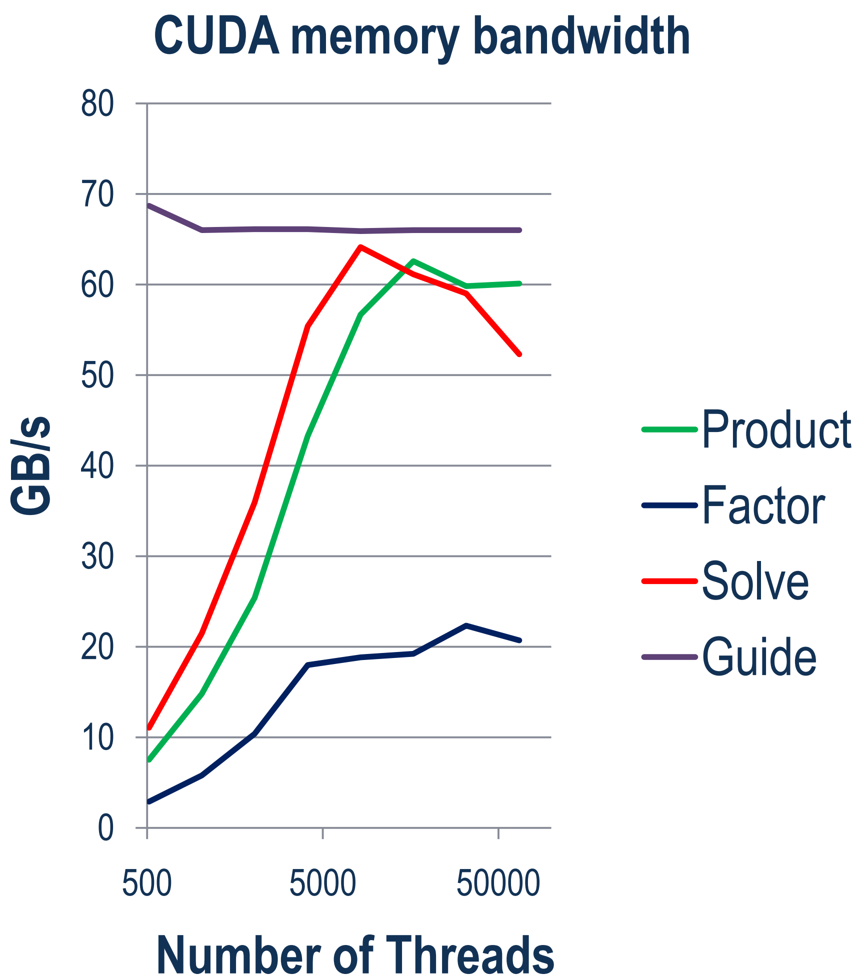
3) Linear solve to find solution increment

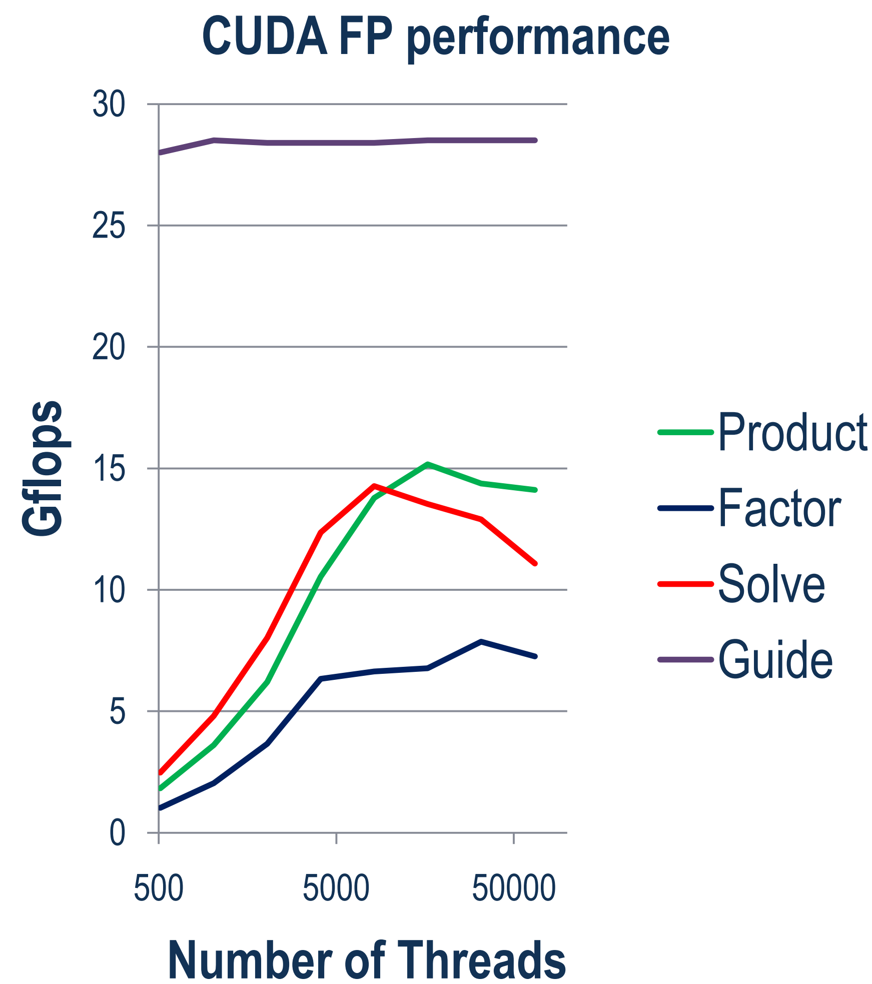
$$\frac{\partial r}{\partial x}\delta x = r, \qquad x = x - \delta x$$

Suitable Components to Migrate to GPU

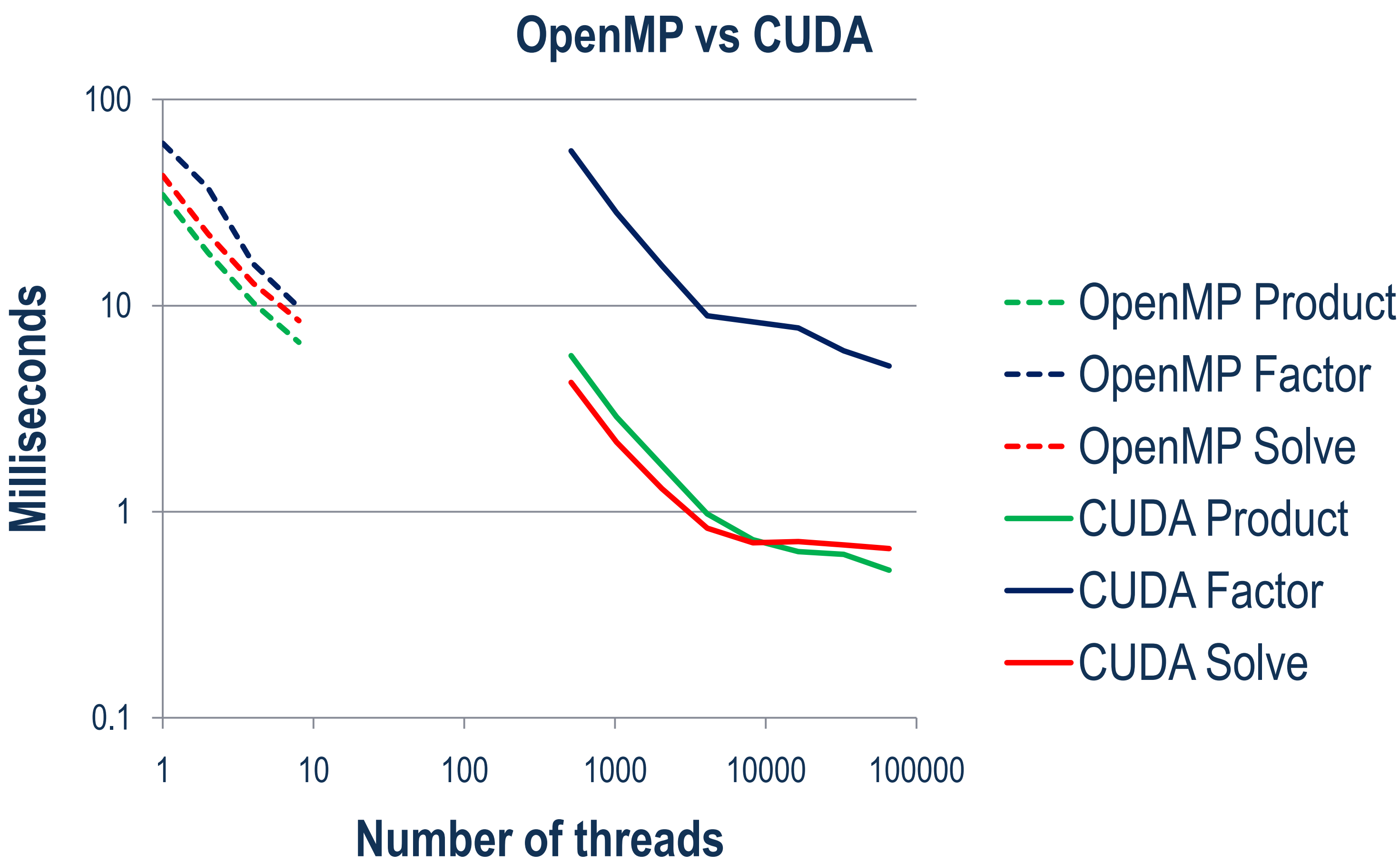
Component	% run time & % code base	Type of algorithm
Property calculations	15-30% run time 15+15% code base Ratio ~ 1	Largely independent cell calculations for fluid properties. Also includes well model. Many branches depending on model
Matrix Assembly	15 – 20% run time 1% code base Ratio ~ 15	Independent assembly of governing equations for each cell
Linear Solver	50 – 70% run time 2% code base Ratio ~ 30	Most potential but also the most sequential part of the code

GPU Linear Solver Components





Should We Write a GPU Linear Solver?



Intel Xeon X5482 & NVIDIA Tesla C1060

Potential Demonstrated

- Ideas evolved into the Massively Parallel Nested Factorization (MPNF) algorithm
- Presented at the 2011 SPE RSS
  - Accelerating Reservoir Simulators using GPU Technology, John R. Appleyard and Jeremy D. Appleyard, Polyhedron Software, and Mark A. Wakefield and Arnaud L. Desitter, Schlumberger (SPE-141402-PP)
- Results generated on an Intel Xeon X5550 with a NVIDIA Tesla C2050

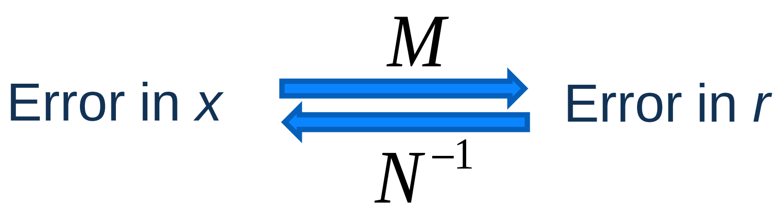
Iterative Linear Solver Essentials

Solve Mx = r, constructing the solution in terms of a small number of basis vectors (v)

$$\tilde{x} = \sum \alpha_i v_i \qquad \left\| M\tilde{x} - r \right\| < \varepsilon$$

Performance depends on the ability to generate good v's

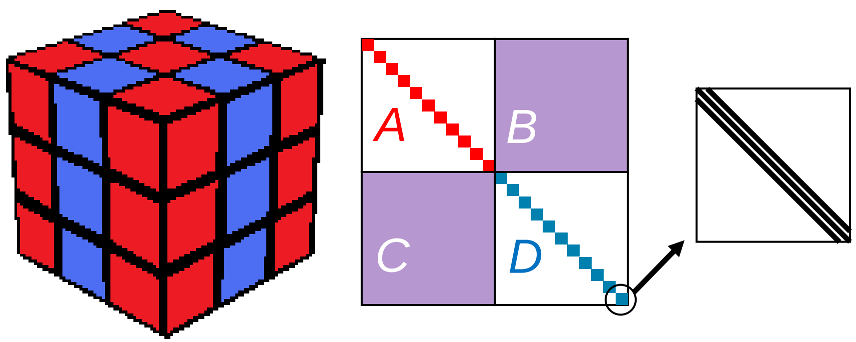
Preconditioner N is an easy to invert approximation to M



N often a factorization (LU) which is essentially a serial algorithm

A GPU Based Preconditioner

Color order grid so that columns are independent



All columns in a color can be factored in parallel. Solve Ny=r as

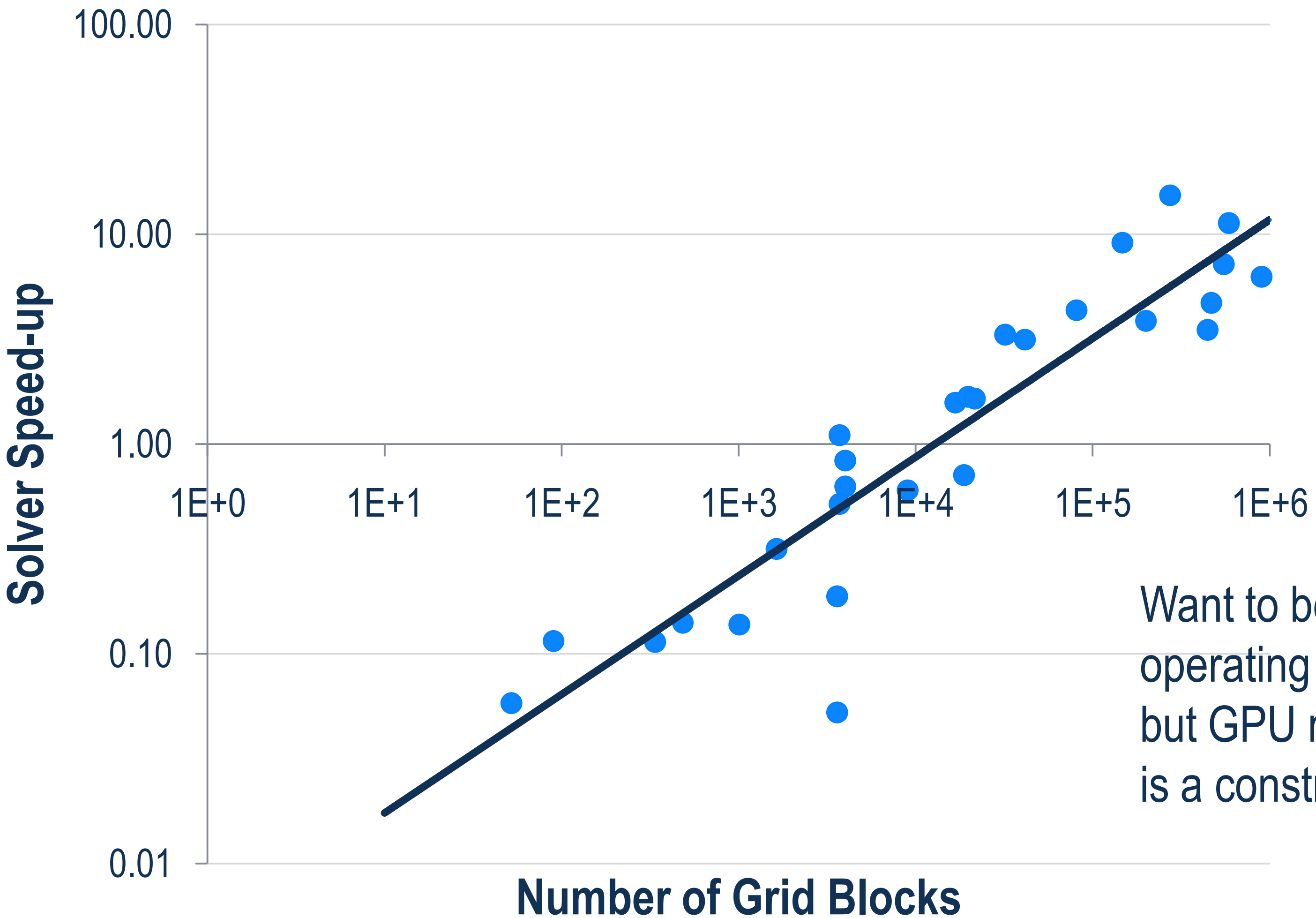
$$\begin{bmatrix} A & \\ C & D \end{bmatrix} \begin{bmatrix} s_R \\ s_B \end{bmatrix} = \begin{bmatrix} r_R \\ r_B \end{bmatrix} \qquad \begin{bmatrix} I & A^{-1}B \\ & I - E \end{bmatrix} \begin{bmatrix} y_R \\ y_B \end{bmatrix} = \begin{bmatrix} s_R \\ s_B \end{bmatrix}$$

GPU Linear Solver Performance

Compared with serial CPU solver on a suite of 30 models with up to 900k grid blocks

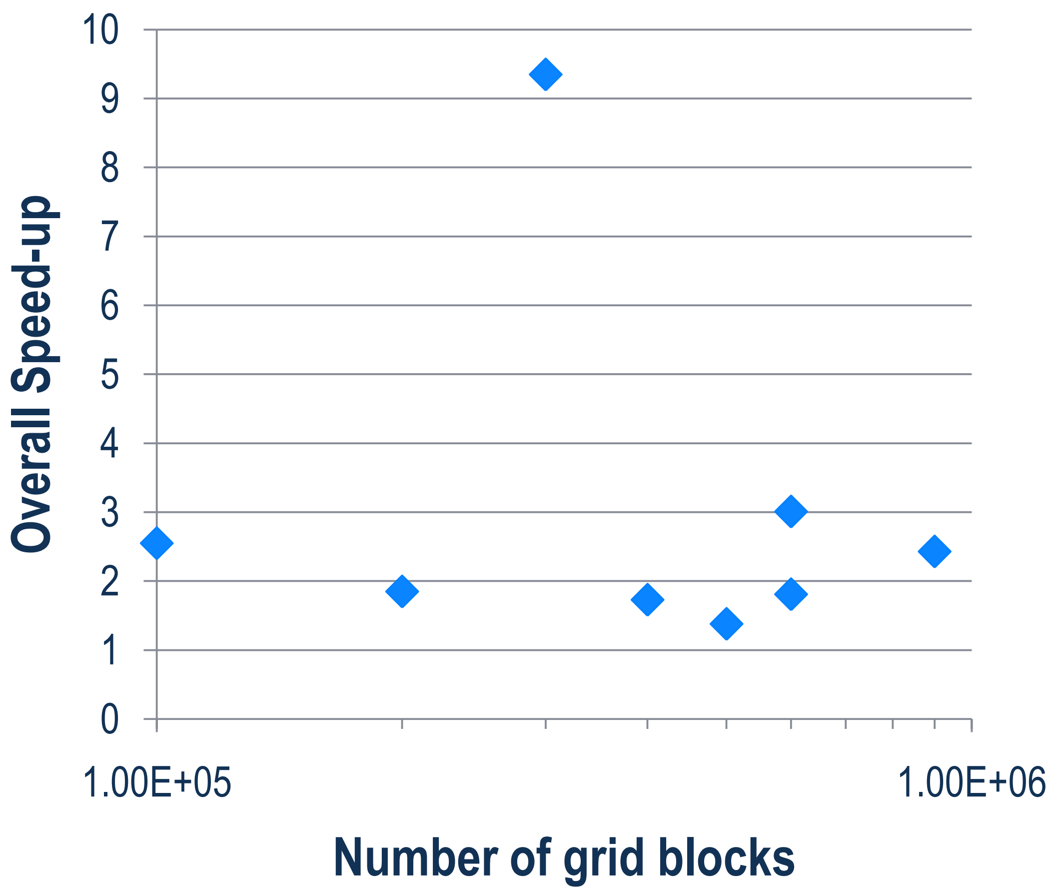
- All models ran to completion with comparable results
- 50% more linear iterations
- 10% non-linear iterations & 10% more timesteps
- Due to trade off between parallelization and accuracy of preconditioner.

Speed-up for Linear Solver Only

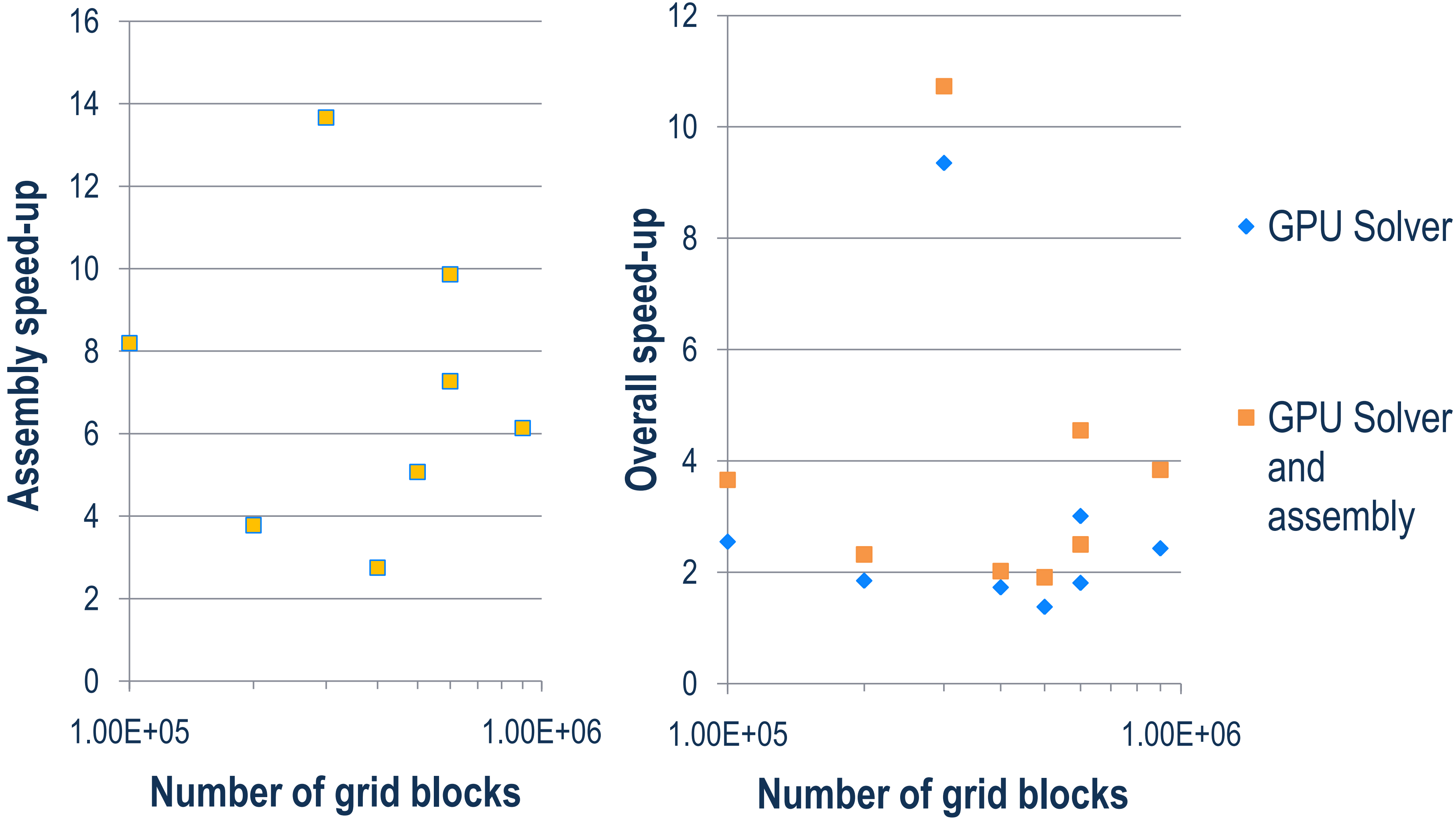


Want to be operating here but GPU memory is a constraint

Overall Speed-up with GPU Linear Solver



Next Step – Assembly on the GPU



GPU Technology Evaluation Conclusions

1. How can GPUs be used in existing commercial products
   – With modest effort we can achieve a ~3x speed-up

2. Considerations for algorithm design

3. Considerations for development/architecture frameworks

Algorithm Design - Linear Solver Scalability (I)

Problem size fixed, increase number of GPUs

MPNF is algorithmically unchanged by a domain decomposition layer:

- Process same color simultaneously across all domains
- corrections between colors are passed both within and between domains
- Same iteration count whether on one machine, or a cluster
- Under exploration

Algorithm Design - Linear Solver Scalability (II)

Problem size increasing, fixed number of GPUs

- MPNF has shown good scalability but GPU memory is a constraint on testing
- MPNF is not as recursive as a multi-grid algorithm.
- SLB currently working on massively parallel recursive algorithms that might exhibit better scalability on very large problems

Software Engineering – Past Experiences

- Scientific code can have a long life time
  - Clients still run old versions & models
- Working with non-proprietary standards has enabled ECLIPSE* to adapt to:
  - OS changes
  - Hardware changes
  - Interconnect changes



Software Engineering - Past Experiences

- Simulator developers are not generally Computer Scientists so need an effective environment where performance comes naturally
- Developers swap projects – need a productive environment

Continue to explore pros/cons of other solutions:

Alternative hardware targets - Intel ArBB

Platform neutral approaches - Oxford Parallel Library (OP2)

Higher level implementations - CUDA Thrust

GPU Technology Evaluation Conclusions

1. How can GPUs be used in existing commercial products
   – With modest effort we can achieve a ~3x speed-up

2. Considerations for algorithm design
   – Have demonstrated that existing algorithms can be adapted. Massively parallel algorithms still an area of internal research.

3. Considerations for software engineering
   – Not yet obvious - but we know the cores are coming!