GPU TECHNOLOGY CONFERENCE

Efficient Tridiagonal Solvers for ADI methods and Fluid Simulation

Nikolai Sakharnykh - NVIDIA

San Jose Convention Center, San Jose, CA | September 21, 2010

PRESENTED BY NVIDIA.

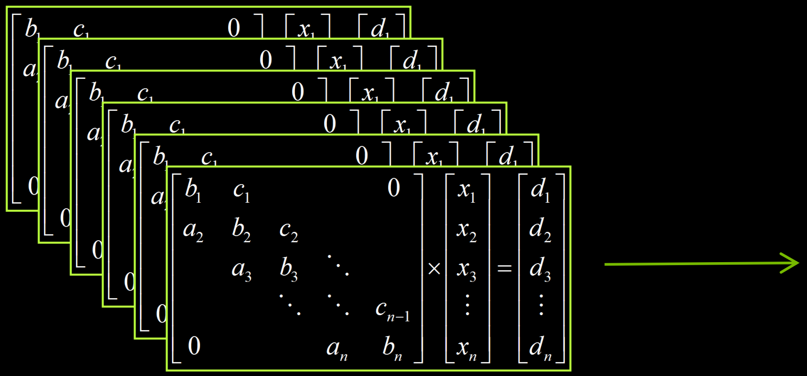
Introduction

- Tridiagonal solvers – very popular technique in both compute and graphics applications
- Application in Alternating Direction Implicit (ADI) methods
- 2 different examples will be covered in this talk:
  - 3D Fluid Simulation for research and science
  - 2D Depth-of-Field Effect for graphics and games

Outline

- Tridiagonal Solvers Overview
  - Introduction
  - Gauss Elimination
  - Cyclic Reduction
- Fluid Simulation in 3D domain
- Depth-of-Field Effect in 2D domain

Tridiagonal Solvers



$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

- Need to solve many independent tridiagonal systems
  - Matrix sizes configurations are problem specific

Gauss Elimination (Sweep)

- Forward sweep: $c_i = \dfrac{c_i}{b_i - c_{i-1}a_i}$ $\quad d_i = \dfrac{d_i - d_{i-1}a_i}{b_i - c_{i-1}a_i}$ $\quad i = 1,2,\ldots,n$
- Backward substitution: $x_n = d_n$ $\quad x_i = d_i - c_i x_{i+1}$ $\quad i = n-1,\ldots,1$
- The fastest serial approach
  - O(N) complexity
  - Optimal number of operations

GPU TECHNOLOGY CONFERENCE

Cyclic Reduction

- Eliminate unknowns using linear combination of equations:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = y_i \longrightarrow \bar{a}_i x_{i-2} + \bar{b}_i x_i + \bar{c}_i x_{i+2} = \bar{y}_i$$

- After one reduction step a system is decomposed into 2:

$$\begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix} \longrightarrow \begin{bmatrix} \bar{b}_1 & 0 & \bar{c}_1 & & & & \\ 0 & \bar{b}_2 & 0 & \bar{c}_2 & & & \\ \bar{a}_3 & 0 & \bar{b}_3 & 0 & \bar{c}_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & \bar{a}_{n-2} & 0 & \bar{b}_{n-2} & 0 & \bar{c}_{n-2} \\ & & & \bar{a}_{n-1} & 0 & \bar{b}_{n-1} & 0 \\ & & & & \bar{a}_n & 0 & \bar{b}_n \end{bmatrix} \longrightarrow \begin{bmatrix} \bar{b}_1 & \bar{c}_1 & & & & & & \\ \bar{a}_3 & \bar{b}_3 & \ddots & & & & & \\ & \ddots & \ddots & \bar{c}_{n-3} & & & & \\ & & \bar{a}_{n-1} & \bar{b}_{n-1} & 0 & & & \\ & & & 0 & \bar{b}_2 & \bar{c}_2 & & \\ & & & & \bar{a}_4 & \bar{b}_4 & \ddots & \\ & & & & & \ddots & \ddots & \bar{c}_{n-2} \\ & & & & & & \bar{a}_n & \bar{b}_n \end{bmatrix}$$

- Reduction step can be done by N threads in parallel

Cyclic Reduction

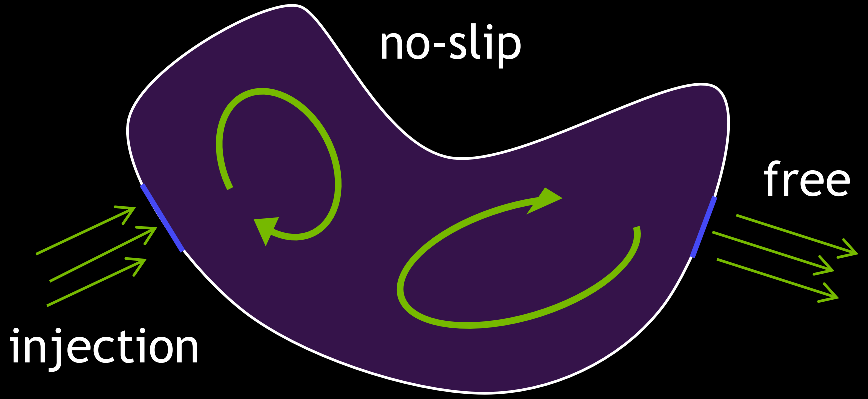
- Parallel Cyclic Reduction (PCR)
  - Apply reduction to new systems and repeat O(log N)
  - For some special cases can use fewer steps
- Cyclic Reduction (CR)
  - Forward reduction, backward substitution
  - Complexity O(N), but requires more operations than Sweep

Outline

- Tridiagonal Solvers Overview
- Fluid Simulation in 3D domain
  - Problem statement, applications
  - ADI numerical method
  - GPU implementation details, optimizations
  - Performance analysis and results comparisons
- Depth-of-Field Effect in 2D domain

Problem Statement

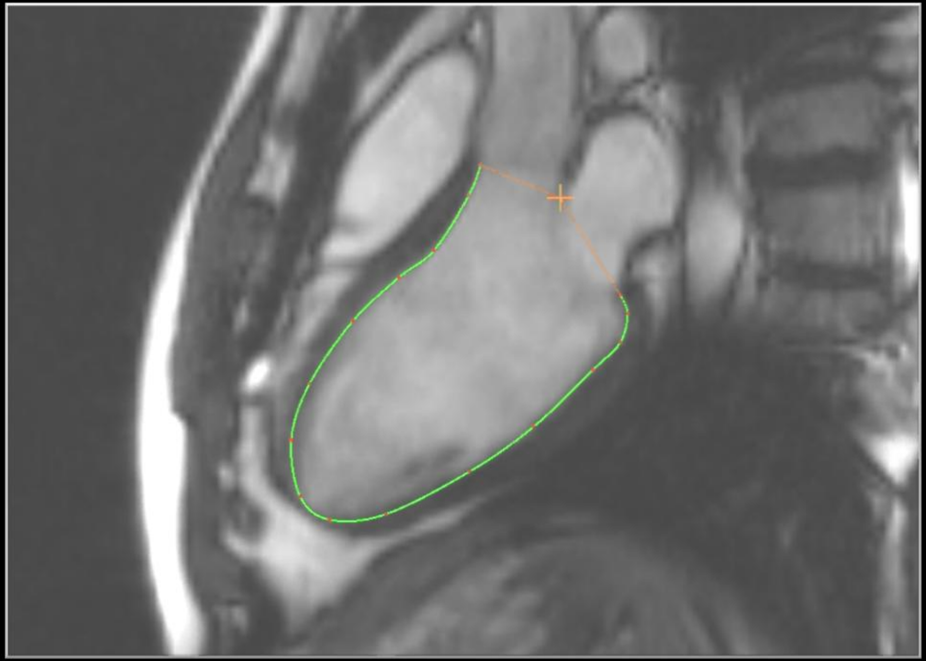
- Viscid incompressible fluid in 3D domain
- New challenge: complex dynamic boundaries



- Euler coordinates: velocity and temperature

Applications

- Blood flow simulation in heart



Capture boundaries movement from MR or US

↓

Simulate blood flow inside heart volume

Applications

- Sea and ocean simulations

Static boundaries

Additional simulation parameters: salinity, etc.

Definitions

Density	$\rho = const = 1$
Velocity	$\mathbf{u} = (u, v, w)$
Temperature	T
Pressure	p

- Equation of state
  - Describe relation between p and T
  - Example: $p = \rho RT = RT$

    R - gas constant for air

Governing equations

- Continuity equation
  - For incompressible fluids: $\operatorname{div}\mathbf{u} = 0$
- Navier-Stokes equations:
  - Dimensionless form, use equation of state

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = -\nabla T + \frac{1}{\mathrm{Re}}\nabla^2\mathbf{u}$$

Re - Reynolds number (= inertia/viscosity ratio)

Governing equations

- Energy equation:
  - Dimensionless form, use equation of state

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T = -\nabla T + \frac{1}{\Pr \cdot \mathrm{Re}} \Delta T + \frac{\gamma - 1}{\gamma \cdot \mathrm{Re}} \Phi$$

γ - heat capacity ratio

$\Pr$ - Prandtl number

Φ - dissipative function

ADI numerical method

- Alternating Direction Implicit

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}=-\frac{\partial T}{\partial x}+\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)$$

X

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=-\frac{\partial T}{\partial x}+\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}\right)$$

Fixed Y, Z

Y

$$\frac{\partial u}{\partial t}+v\frac{\partial u}{\partial y}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial y^2}\right)$$

Fixed X, Z

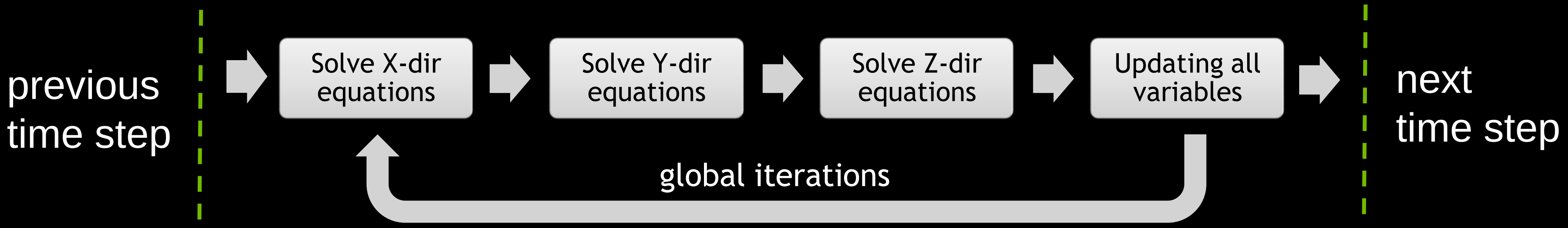
Z

$$\frac{\partial u}{\partial t}+w\frac{\partial u}{\partial z}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial z^2}\right)$$

Fixed X, Y

ADI method - iterations

- Use global iterations for the whole system of equations



- Some equations are not linear:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -\frac{\partial T}{\partial x} + \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}\right)$$

  – Use local iterations to approximate the non-linear term

Discretization

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -\frac{\partial T}{\partial x} + \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}\right)$$

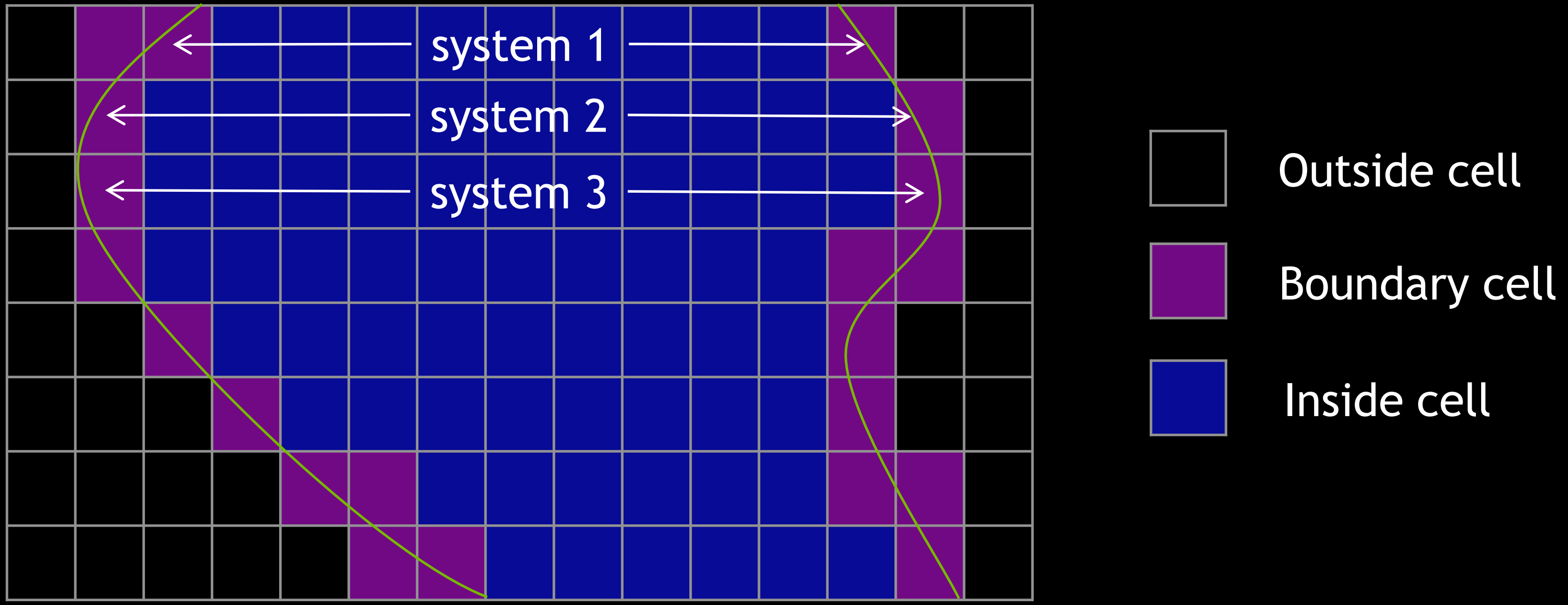
- Use regular grid, implicit finite difference scheme:

$$\frac{\overset{i}{u^{n+1}_{i,j,k}} - u^{n}_{i,j,k}}{\Delta t} + u^{n}_{i,j,k}\frac{\overset{i+1}{u^{n+1}_{i+1,j,k}} - \overset{i-1}{u^{n+1}_{i-1,j,k}}}{\Delta x} = -\frac{T^{n}_{i+1,j,k} - T^{n}_{i-1,j,k}}{\Delta x} + \frac{1}{\mathrm{Re}}\frac{\overset{i+1}{u^{n+1}_{i+1,j,k}} - \overset{i}{2u^{n+1}_{i,j,k}} + \overset{i-1}{u^{n+1}_{i-1,j,k}}}{\Delta x^2}$$

- Got a tridiagonal system for $u^{n+1}_{i,j,k}$ $\quad i = 1,..,N_x$
  - Independent system for each fixed pair (j, k)

Tridiagonal systems

- Need to solve lots of tridiagonal systems
- Sizes of systems may vary across the grid



Implementation details

```
<for each direction X, Y, Z>
{
    <for each local iteration>
    {
        <for each equation u, v, w, T>
        {
            build tridiagonal matrices and rhs
            solve tridiagonal systems
        }
        update non-linear terms
    }
}
```

GPU implementation

- Store all data arrays entirely on GPU in linear memory
  - Reduce amount of memory transfers to minimum
  - Map 3D arrays to linear memory

(X, Y, Z)

↓

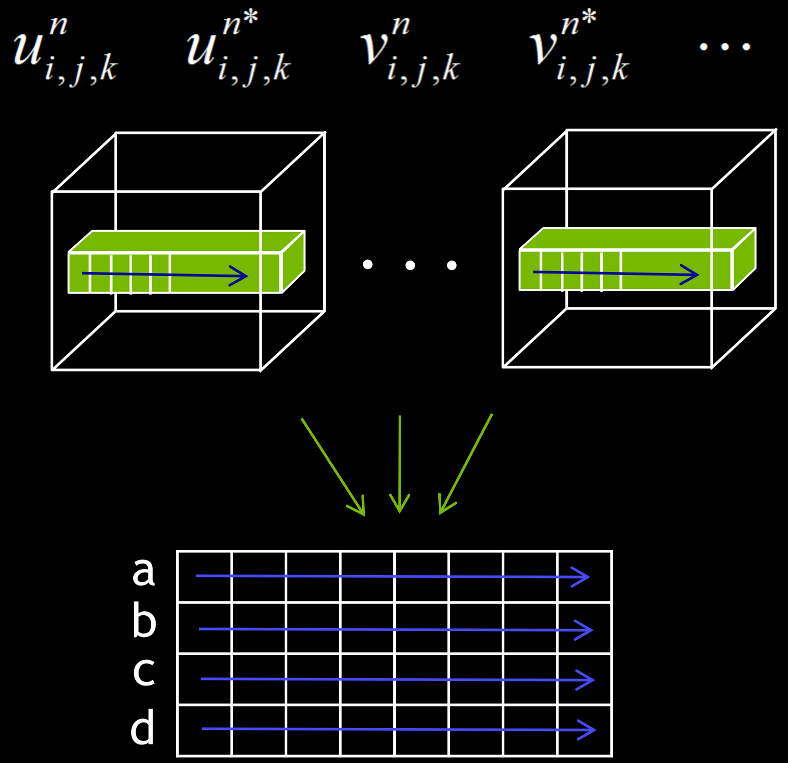
Z + Y * dimZ + X * dimY * dimZ

Z – fastest-changing dimension

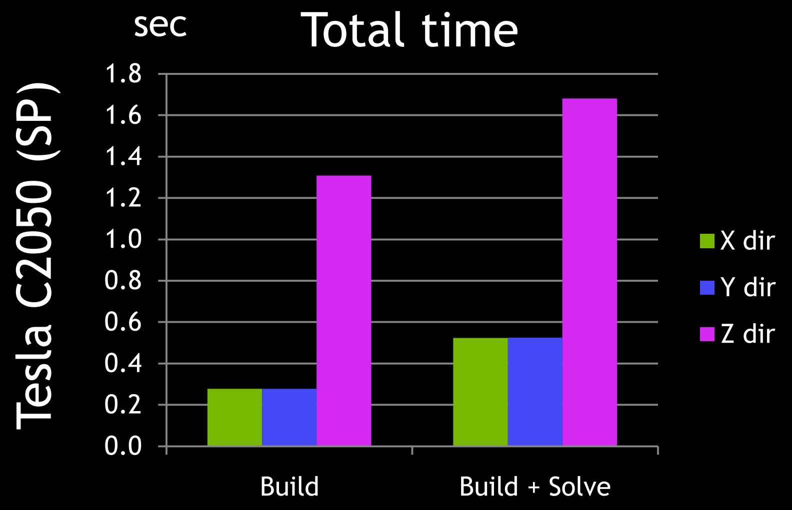
- Main tasks
  - Build matrices
  - Solve systems
- Additional routines for non-linear updates
  - Merge, copy 3D arrays with masks

Building matrices

- Input data:
  - Previous/non-linear 3D layers
- Each thread computes:
  - Coefficients of a tridiagonal matrix
  - Right-hand side vector
- Use C++ templates for direction and equation



Building matrices - performance



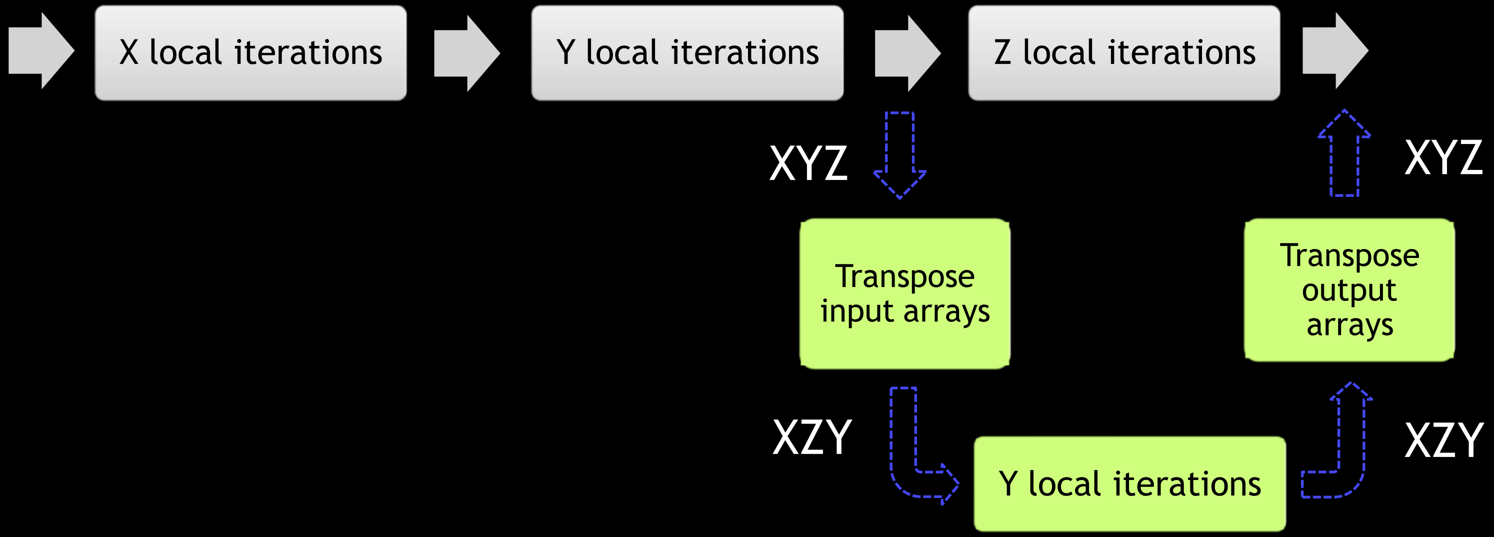
Build kernels

Dir	Requests per load	L1 global load hit %	IPC
X	2 - 3	25 - 45	1.4
Y	2 - 3	33 - 44	1.4
Z	32	0 - 15	0.2

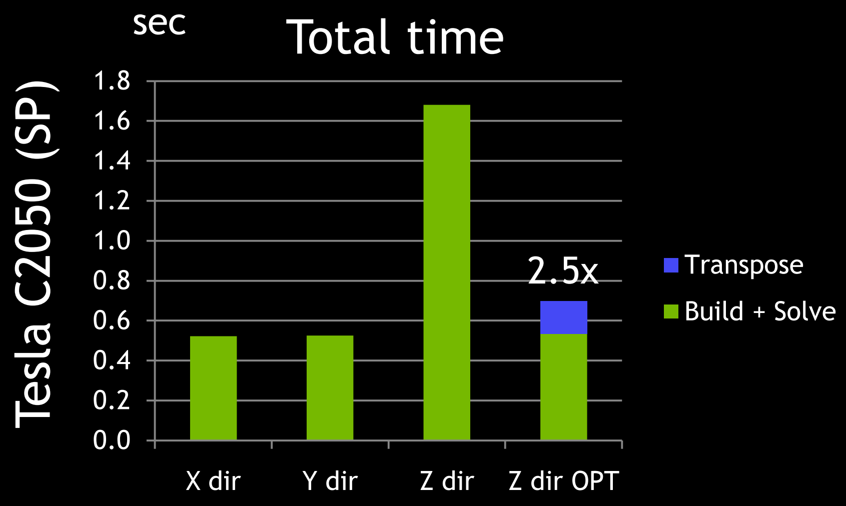
- Poor Z direction performance compared to X/Y
  - Threads access contiguous memory region
  - Memory access is uncoalesced, lots of cache misses

Building matrices - optimization

- Run Z phase in transposed XZY space
  - Better locality for memory accesses
  - Additional overhead on transpose



Building matrices - optimization



Build kernels

Z dir	Requests per load	L1 global load hit %	IPC
Original	32	0 – 15	0.2
Transposed	2 – 3	30 – 38	1.3

- Tridiagonal solver time dominates over transpose
  – Transpose will takes less % with more local iterations

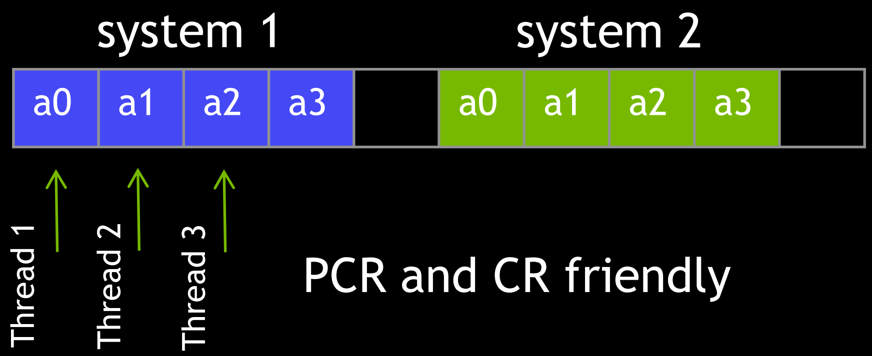
Problem analysis

- Big number of tridiagonal systems
  - Grid size squared on each step: 16K for 128^3
- Relatively small system sizes
  - Max of grid size: 128 for 128^3
- 1 thread per system is a suitable approach for this case
  - Enough systems to utilize GPU

Solving tridiagonal systems

- Matrix layout is crucial for performance

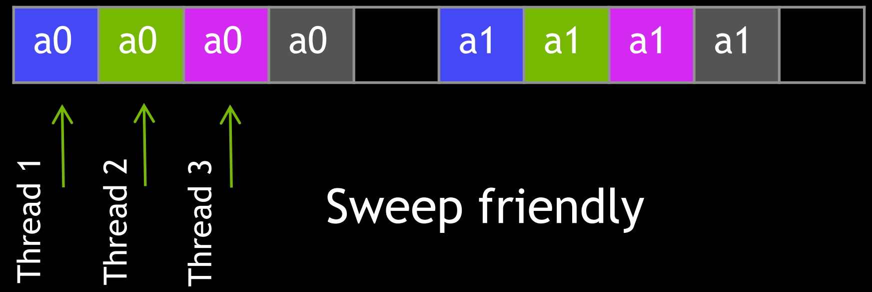
Sequential layout



PCR and CR friendly

ADI – Z direction

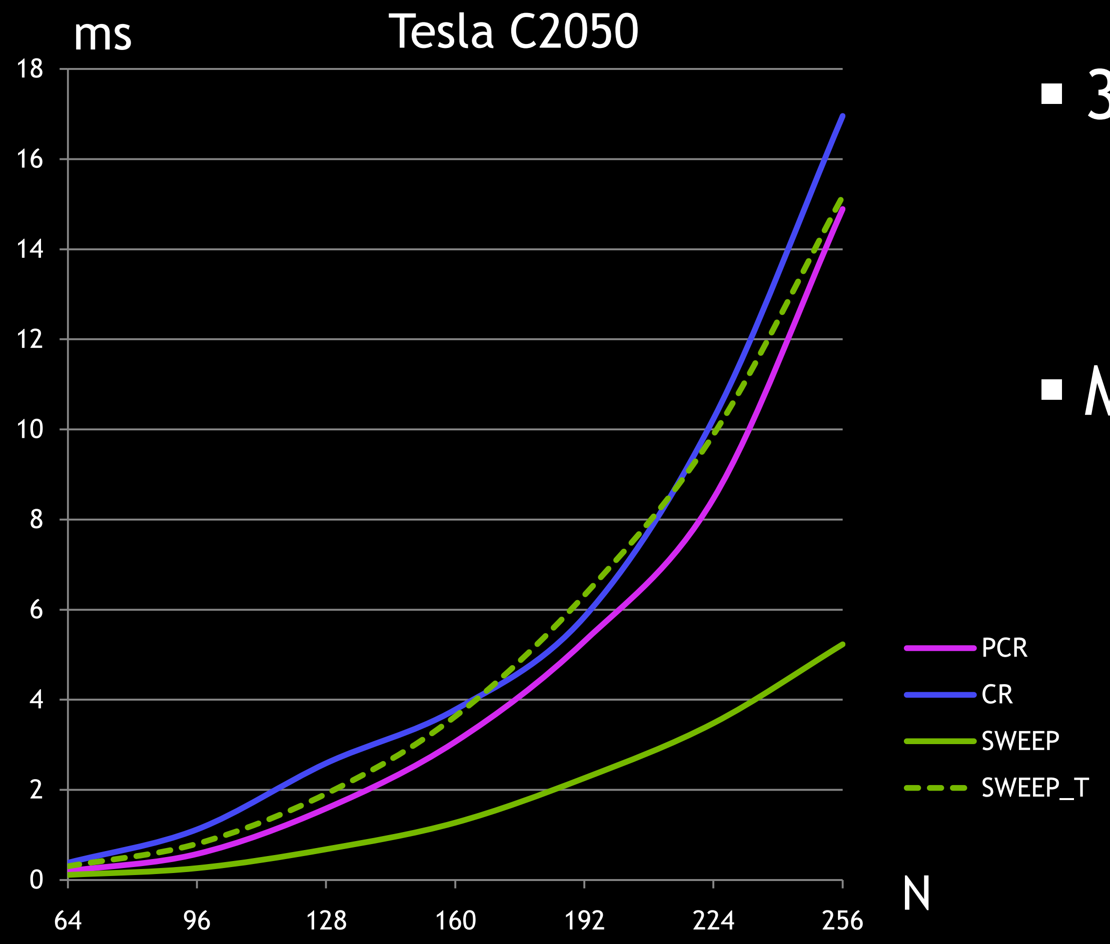
Interleaved layout



Sweep friendly

ADI – X, Y directions

Tridiagonal solvers performance



- 3D ADI test setup
  - Systems size = N
  - Number of systems = N^2
- Matrix layout
  - *CR, SWEEP_T: sequential
  - SWEEP: interleaved

PCR/CR implementations from:
"Fast tridiagonal solvers on the GPU"
by Y. Zhang, J. Cohen, J. Owens, 2010

Choosing tridiagonal solver

- Application for 3D ADI method
  – For X, Y directions matrices are interleaved by default
  – Z is interleaved as well if doing in transposed space

↓

- Sweep solver seems like the best choice
  – Although can experiment with PCR for Z direction

Performance analysis - Fermi

- L1/L2 effect on performance
  - Using 48K L1 instead of 16K gives 10-15% speed-up
  - Turning L1 off reduces performance by 10%
  - Really help on misaligned accesses and spatial reuse

- Sweep effective bandwidth on Tesla C2050
  - Single Precision = 119 GB/s, 82% of peak
  - Double Precision = 135 GB/s, 93% of peak

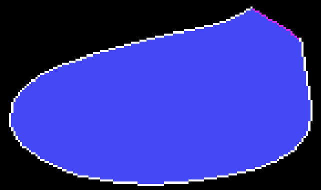
Performance benchmark

- CPU configuration:
  - Intel Core i7 4 cores @ 2.8 GHz
  - Use all 4 cores through OpenMP (3-4x speed-up vs 1 core)
- GPU configurations:
  - NVIDIA Tesla C1060
  - NVIDIA Tesla C2050
- Measure Build + Solve time for all iterations

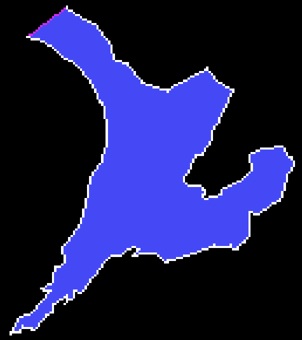
Test cases

Cube

Heart Volume



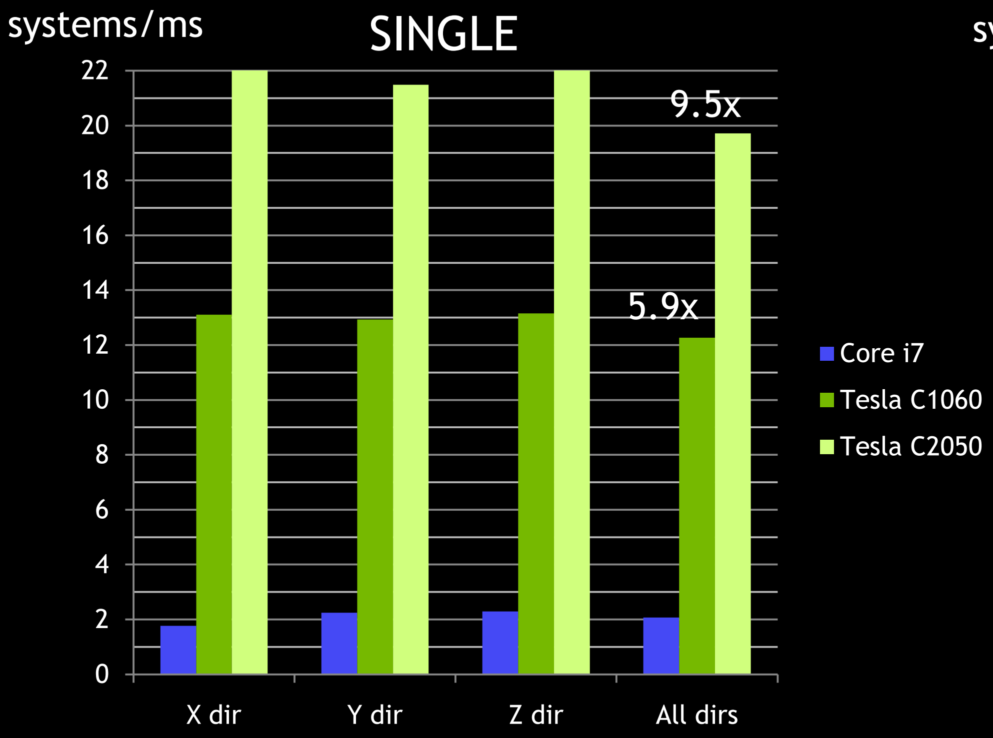
Sea Area

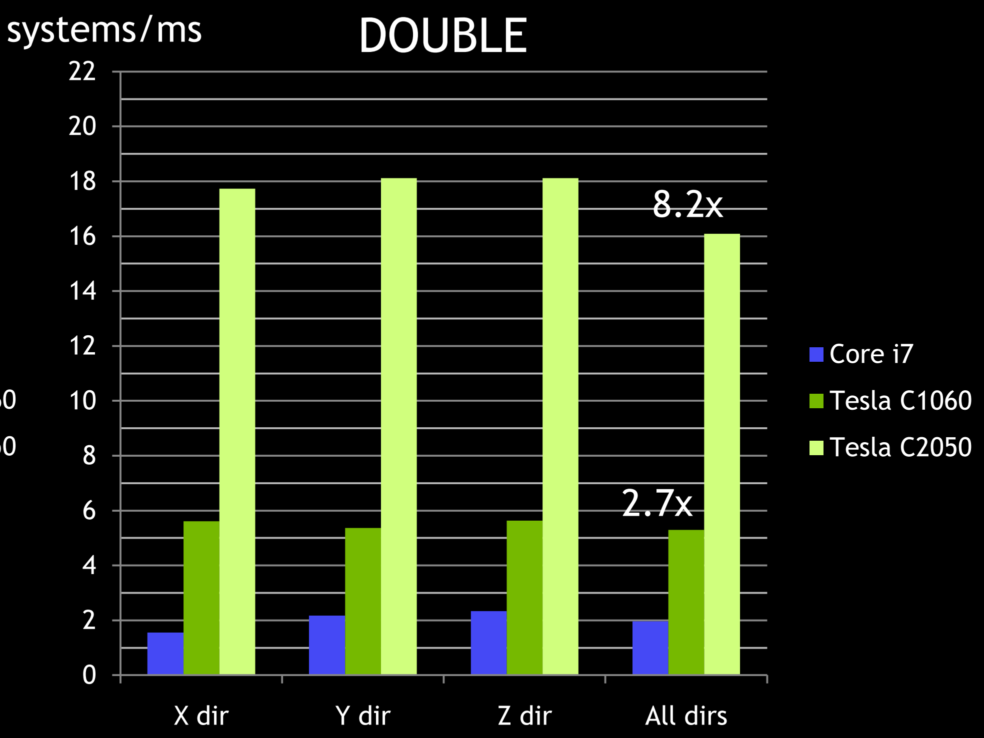


Test	Grid size	X dir	Y dir	Z dir
Cube	128 x 128 x 128	10K	10K	10K
Heart Volume	96 x 160 x 128	13K	8K	8K
Sea Area	160 x 128 x 128	18K	20K	6K

Performance results - Cube

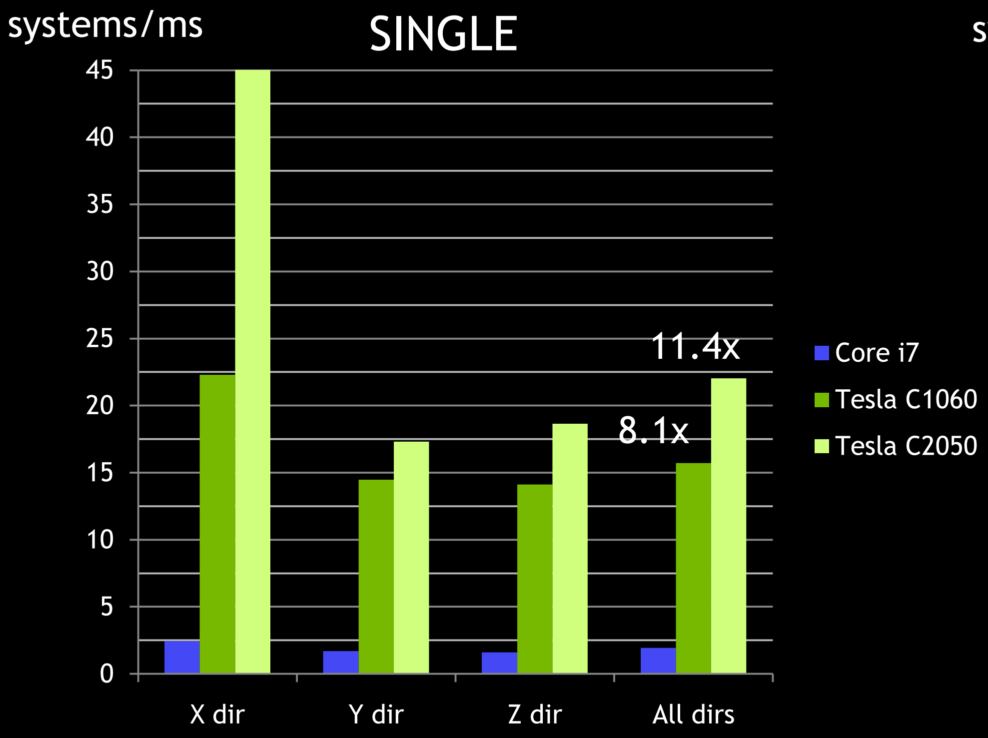
- Same number of systems for X/Y/Z, all sizes are equal

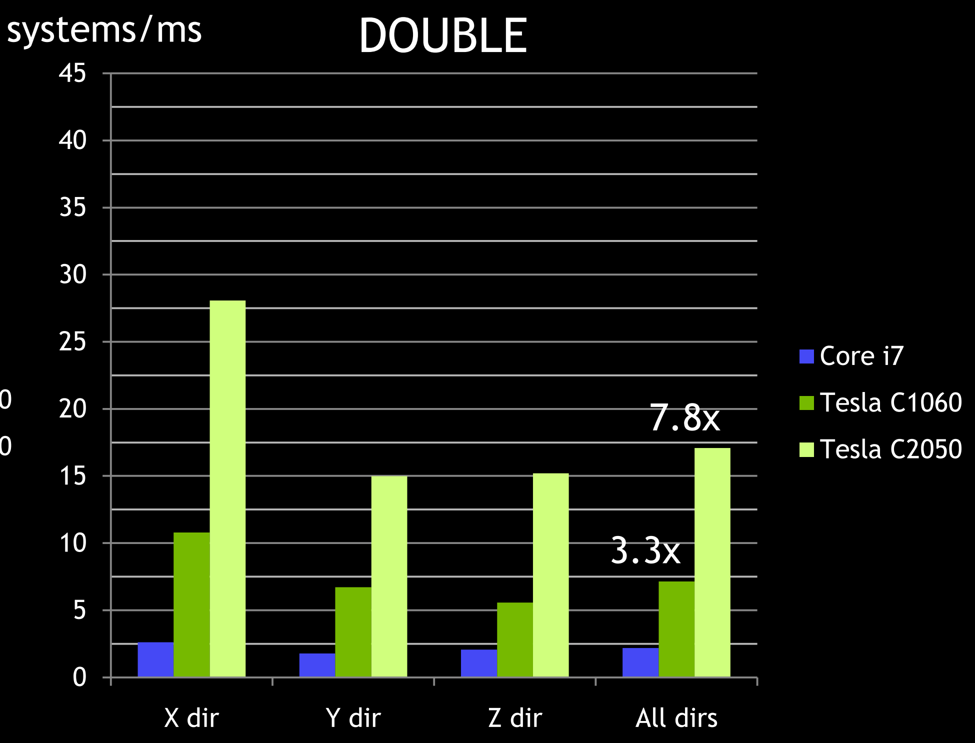




Performance results - Heart Volume

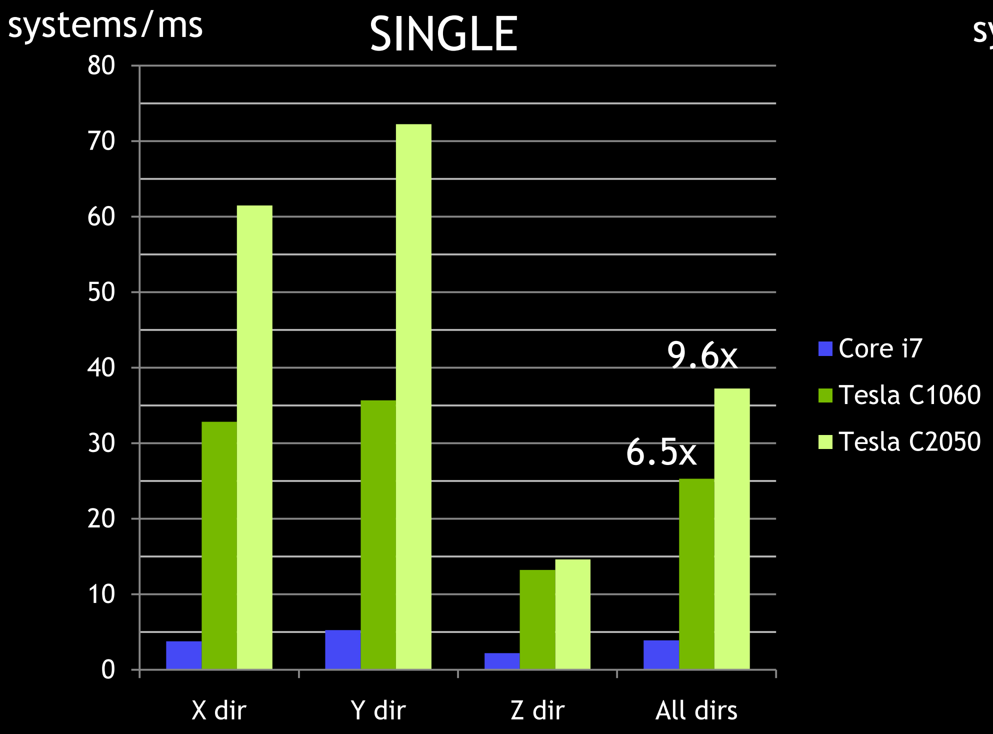
- More systems for X, sizes vary smoothly

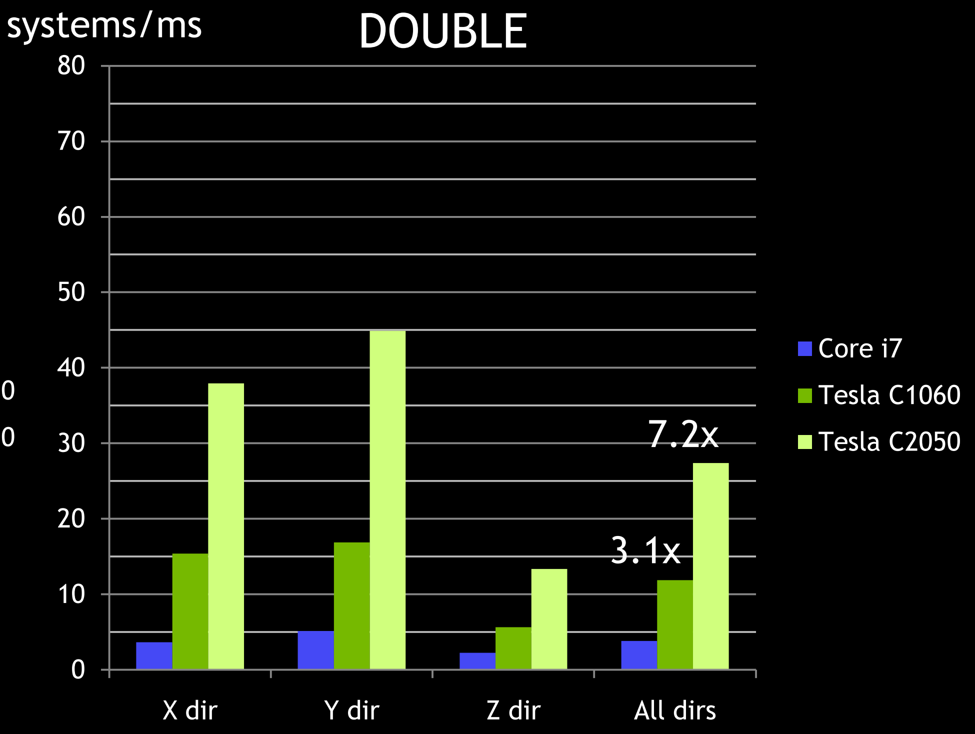




Performance results - Sea Area

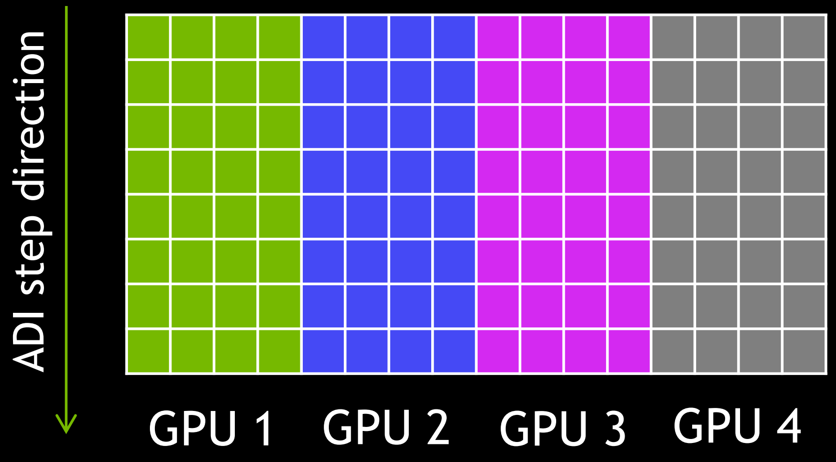
- Lots of small systems of different size for X/Y





Future work

- Current main limitation is memory capacity
  - Lots of 3D arrays: grid data, time layers, tridiagonal matrices
- Multi-GPU and clusters enable high-resolution grids



Split grid along current direction (X, Y or Z)

Assign each partition to one GPU

Solve systems and update common 3D layers

References

- http://code.google.com/p/cmc-fluid-solver/
- "Tridiagonal Solvers on the GPU and Applications to Fluid Simulation", GPU Technology Conference, 2009, Nikolai Sakharnykh
- "A dynamic visualization system for multiprocessor computers with common memory and its application for numerical modeling of the turbulent flows of viscous fluids", Moscow University Computational Mathematics and Cybernetics, 2007, Paskonov V.M., Berezin S.B., Korukhova E.S.

Outline

- Tridiagonal Solvers Overview
- Fluid Simulation in 3D domain
- Depth-of-Field Effect in 2D domain
  - Problem statement
  - ADI numerical method
  - Analysis and hybrid approach
  - Shared memory optimization
  - Visual results

Problem statement

- ADI method application for 2D problems
- Real-time Depth-Of-Field simulation
  - Using diffusion equation to blur the image

$$\frac{\partial u}{\partial t} = \nabla \cdot (\beta \cdot \nabla u)$$

- Now need to solve tridiagonal systems in 2D domain
  - Different setup, different methods for GPU

Numerical method

- Alternating Direction Implicit method

$$\frac{\partial u}{\partial t} = \nabla \cdot (\beta \cdot \nabla u)$$

X

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\beta(x, y_j)\frac{\partial u}{\partial x}\right)$$

for all fixed $y_j = 0, \ldots, H-1$

Y

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial y}\left(\beta(x_i, y)\frac{\partial u}{\partial y}\right)$$

for all fixed $x_i = 0, \ldots, W-1$

Implementing ADI method

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\beta(x, y_j)\frac{\partial u}{\partial x}\right)$$

- Implicit scheme is always stable
- After discretization on a regular grid with dx = dt = 1:

$$\overset{i}{u_{i,j}^{n+1}} - u_{i,j}^{n} = \overset{i-1}{u_{i-1,j}^{n+1}}\beta_{i-1,j} - \overset{i}{u_{i,j}^{n+1}}(\beta_{i-1,j} + \beta_{i,j}) + \overset{i+1}{u_{i+1,j}^{n+1}}\beta_{i,j}$$

- Got a tridiagonal system for each fixed j

Problem analysis

- Small number of systems
  - Max 2K for high resolutions
- Large matrix dimensions
  - Up to 2K for high resolutions
- Using 1 thread per system wouldn't be so efficient
  - Low GPU utilization

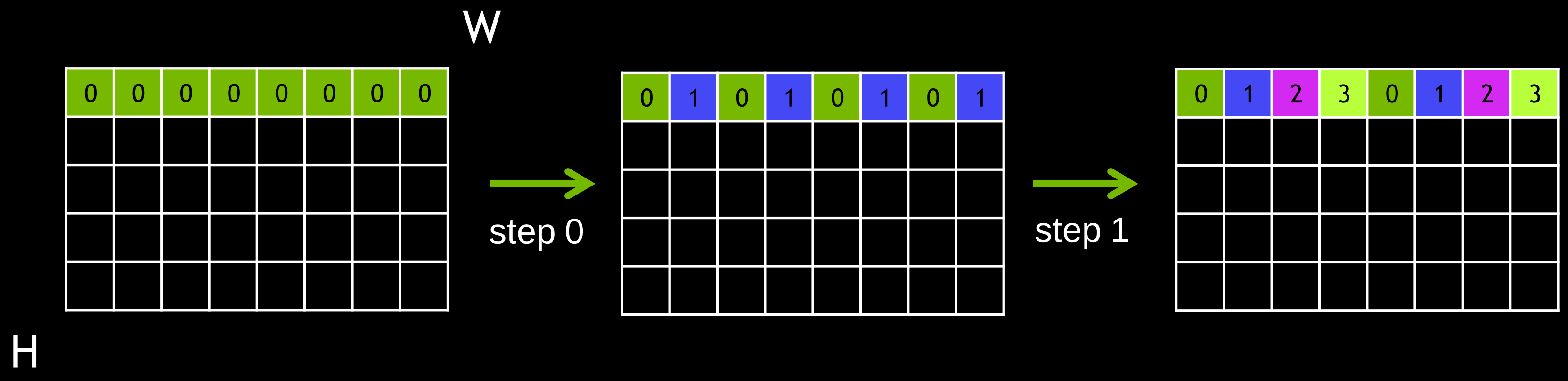
GPU implementation

- Matrix layout
  - Sequential for X direction
  - Interleaved for Y direction

- Use hybrid algorithm
  - Start with Parallel Cyclic Reduction (PCR)
    - Subdivide our systems into smaller ones
  - Finish with Gauss Elimination (Sweep)
    - Solve each new system by 1 thread

PCR step

- Doubles number of systems, halves systems size



Systems	H
Size	W

Systems	H*2
Size	W/2

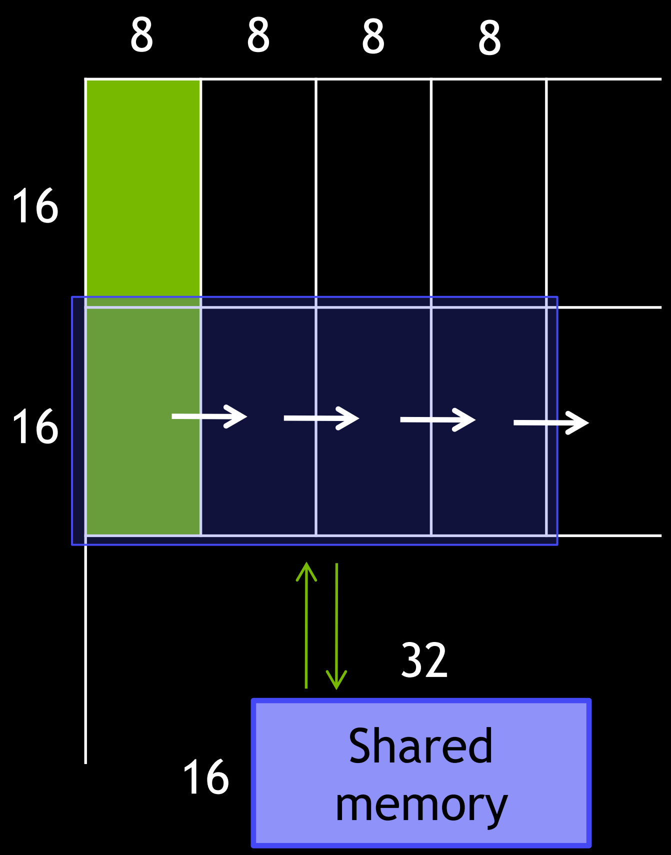
Systems	H*4
Size	W/4

Hybrid approach

- Benefits of each additional PCR step
  – Improves memory layout for Sweep in X direction
  – Subdivides systems for more efficient Sweep
- But additional overhead for each step
  – Increasing total number of operations for solving a system
- Need to choose an optimal number of PCR steps
  – For DOF effect in high-res: 3 steps for X direction

Shared memory optimization

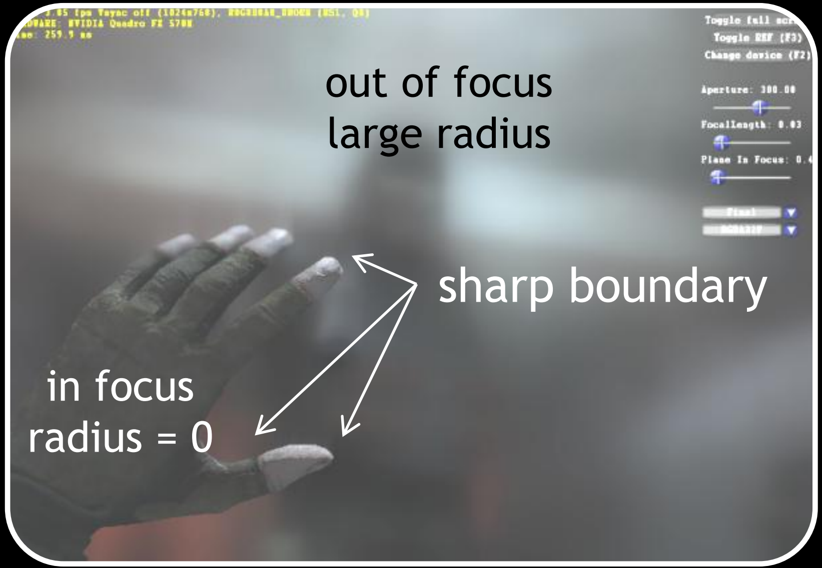
- Shared memory is used as a temporary transpose buffer
  - Load 32x16 area into shared memory
  - Compute 4 iterations inside shared memory
  - Store 32x16 back to global memory
- Gives 20% speed-up



Diffusion simulation idea

- Diffuse temperature series (= color) on 2D plate (= image) with non-uniform heat conductivity (= circles of confusion)

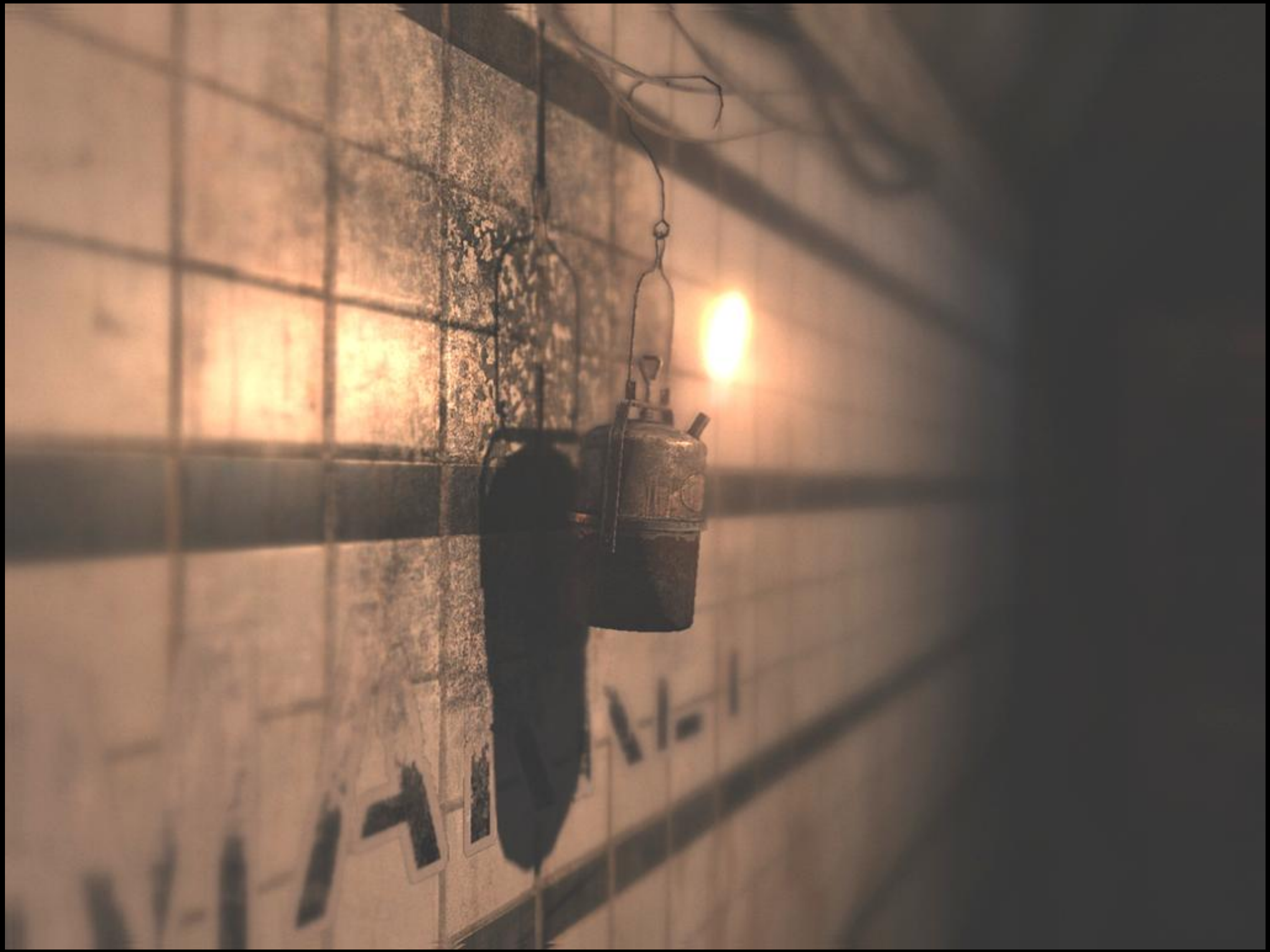
- Key features:
  - No color bleeding
  - Support for spatially varying arbitrary circles of confusion



Depth-of-Field in games

From Metro2033,
© THQ and 4A Games

Depth-of-Field in games



From Metro2033,
© THQ and 4A Games

References

- Pixar article about Diffusion DOF:

http://graphics.pixar.com/library/DepthOfField/paper.pdf

- Cyclic reduction methods on CUDA:

http://www.idav.ucdavis.edu/func/return_pdf?pub_id=978

- Upcoming sample in NVIDIA SDK

Summary

- 10x speed-ups for 3D ADI methods in complex areas
  - Great Fermi performance in double precision
- Achieve real-time performance in graphics and games
  - Realistic depth-of-field effect
- Tridiagonal solvers performance is problem specific

Questions?

GPU TECHNOLOGY CONFERENCE

PRESENTED BY NVIDIA.