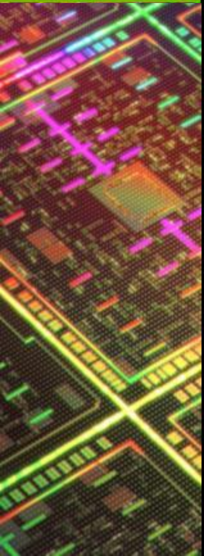
GPU TECHNOLOGY CONFERENCE

# 3D ADI Method for Fluid Simulation: Scaling to Multiple GPUs

GPU TECHNOLOGY CONFERENCE

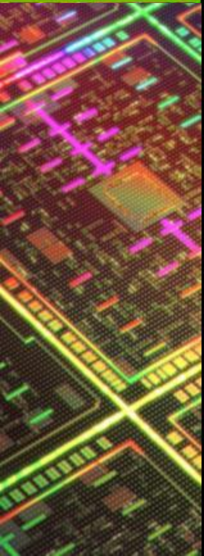


# Introduction

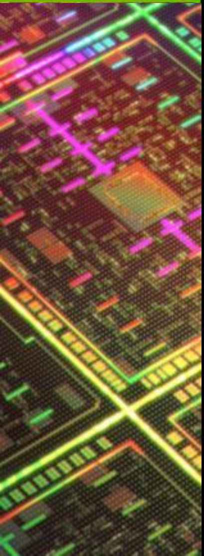
- Fluid simulation using direct numerical methods
  - Gives the most accurate result
  - Requires lots of memory and computational power
- GPUs are very suitable for direct methods
  - Have great instruction throughput and high memory bandwidth
- How will it scale on multiple GPUs?

# cmc-fluid-solver

- Open source project on Google Code
  - Started at CMC faculty of MSU, Russia
  - CPU: OpenMP, GPU: CUDA
- 3D fluid simulation using ADI solver
- Key people:
  - MSU: Vilen Paskonov, Sergey Berezin
  - NVIDIA: Nikolay Sakharnykh, Nikolay Markovskiy



GPU TECHNOLOGY CONFERENCE



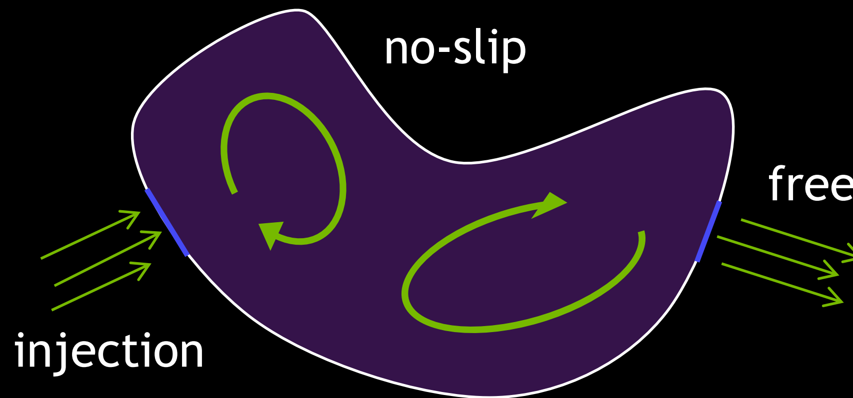
# Outline

- Fluid Simulation in 3D domain
  - Problem statement, applications
  - ADI numerical method
  - GPU implementation details, optimizations
  - Performance analysis
- Multi-GPU implementation

GPU TECHNOLOGY CONFERENCE

# Problem Statement

- Viscid weak-compressible fluid in 3D domain
- Arbitrary closed geometry for boundaries




- Euler coordinates: velocity and temperature

GPU TECHNOLOGY CONFERENCE

# Applications

- Sea and ocean simulations

Static boundaries

Additional simulation parameters: salinity, etc.

GPU TECHNOLOGY CONFERENCE

# 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

GPU TECHNOLOGY CONFERENCE

# Governing equations

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

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

$\mathrm{Re}$ - Reynolds number (= inertia/viscosity ratio)

# Governing equations

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

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

$\gamma$ - heat capacity ratio

$\mathrm{Pr}$ - Prandtl number

$\Phi$ - dissipative function

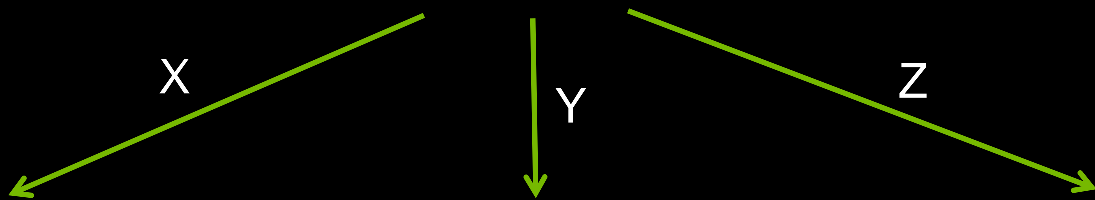
GPU TECHNOLOGY CONFERENCE

GPU TECHNOLOGY CONFERENCE

# ADI numerical method

- Splitting X-velocity equation

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




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

Fixed Y, Z

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

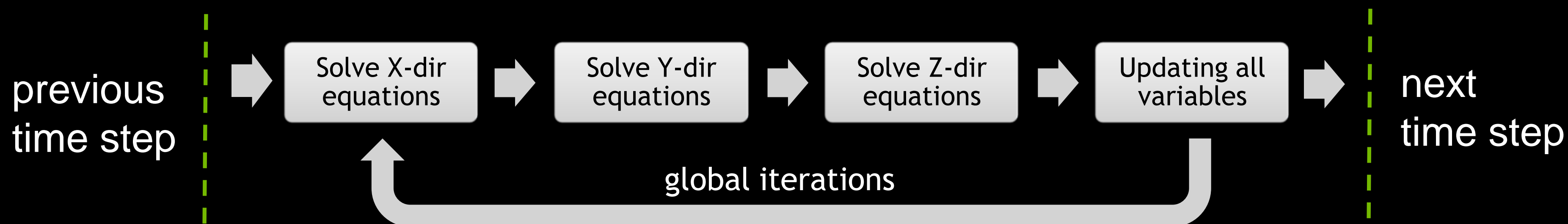
Fixed X, Z

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

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

GPU TECHNOLOGY CONFERENCE

# Discretization

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

- Use regular grid, implicit finite difference scheme:

$$\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^{n}}{\Delta t} + u_{i,j,k}^{n}\frac{u_{i+1,j,k}^{n+1} - u_{i-1,j,k}^{n+1}}{\Delta x} = -\frac{T_{i+1,j,k}^{n} - T_{i-1,j,k}^{n}}{\Delta x} + \frac{1}{\text{Re}}\frac{u_{i+1,j,k}^{n+1} - 2u_{i,j,k}^{n+1} + u_{i-1,j,k}^{n+1}}{\Delta x^2}$$

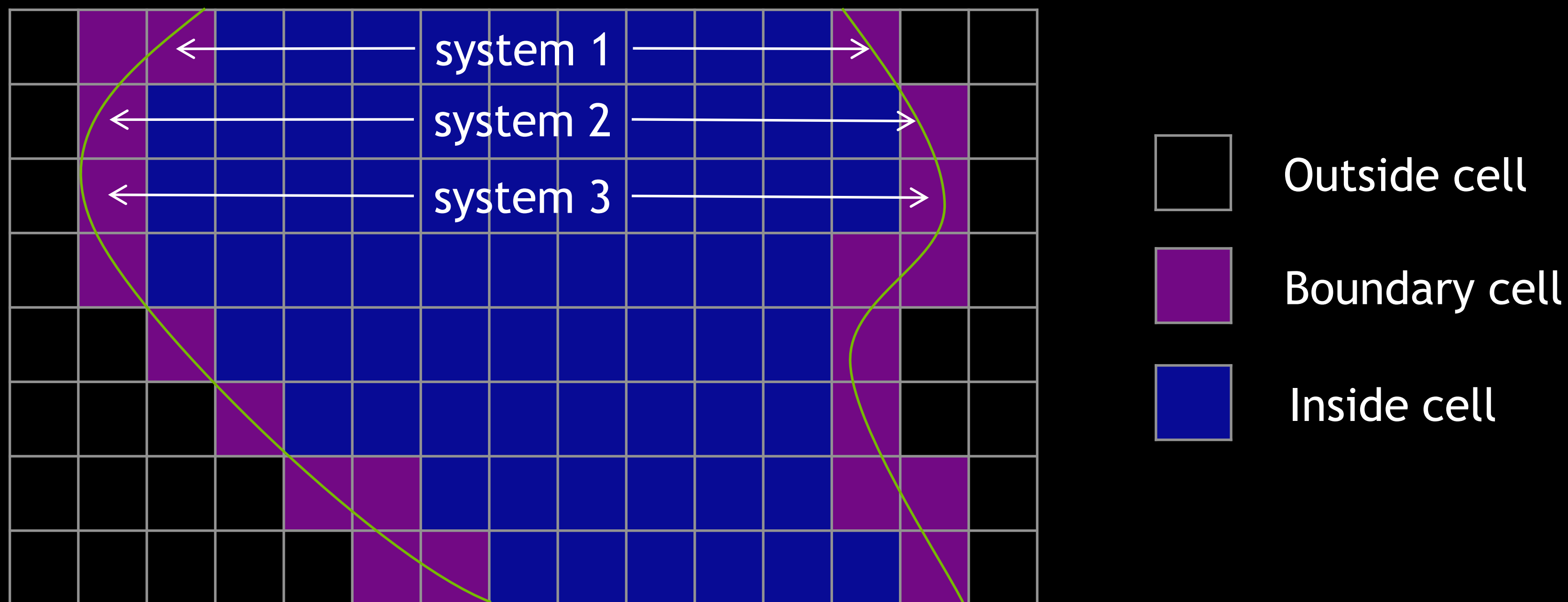
(Annotations above the circled terms: i, i+1, i-1, i+1, i, i-1)

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

GPU TECHNOLOGY CONFERENCE

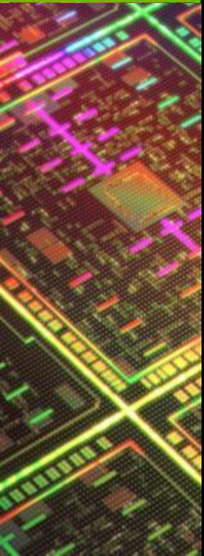
# Tridiagonal systems

- 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 TECHNOLOGY CONFERENCE

# GPU implementation

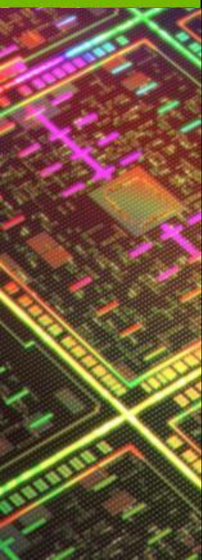
- Store all data arrays entirely in GPU memory
  - Reduce number of PCI-E transfers to minimum
  - Map 3D arrays to linear memory

(X, Y, Z)

Z + Y * dimZ + X * dimY * dimZ

Z – fastest-changing dimension

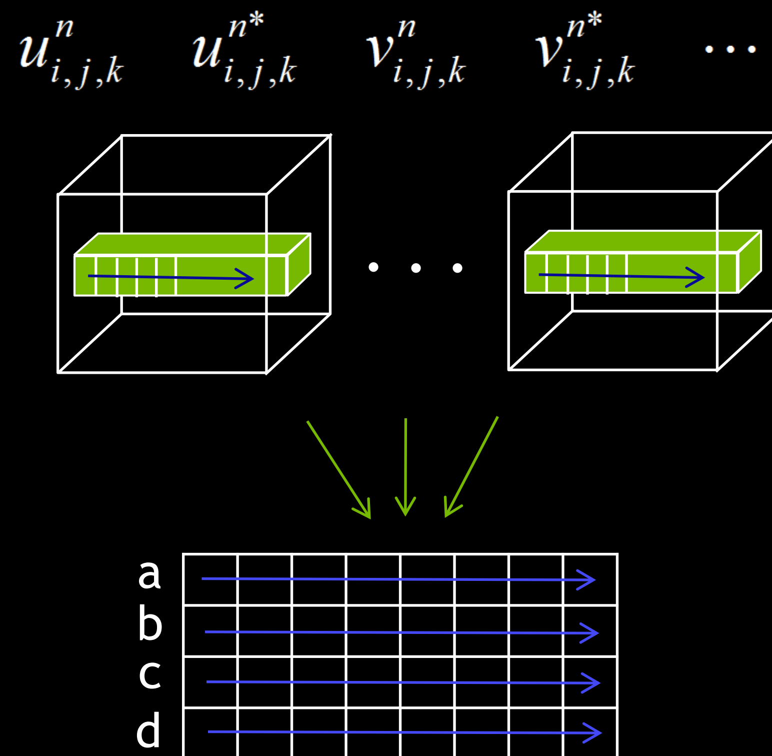
- Main kernel
  - Build matrix coefficients
  - Solve tridiagonal systems



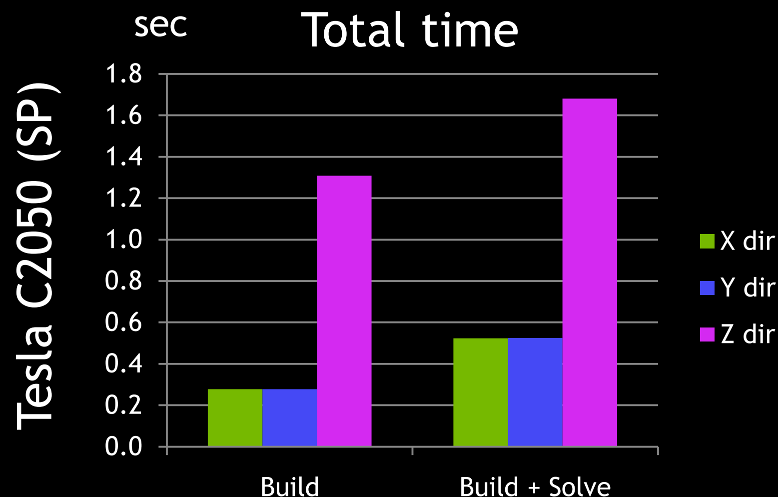
GPU TECHNOLOGY CONFERENCE

# 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 |

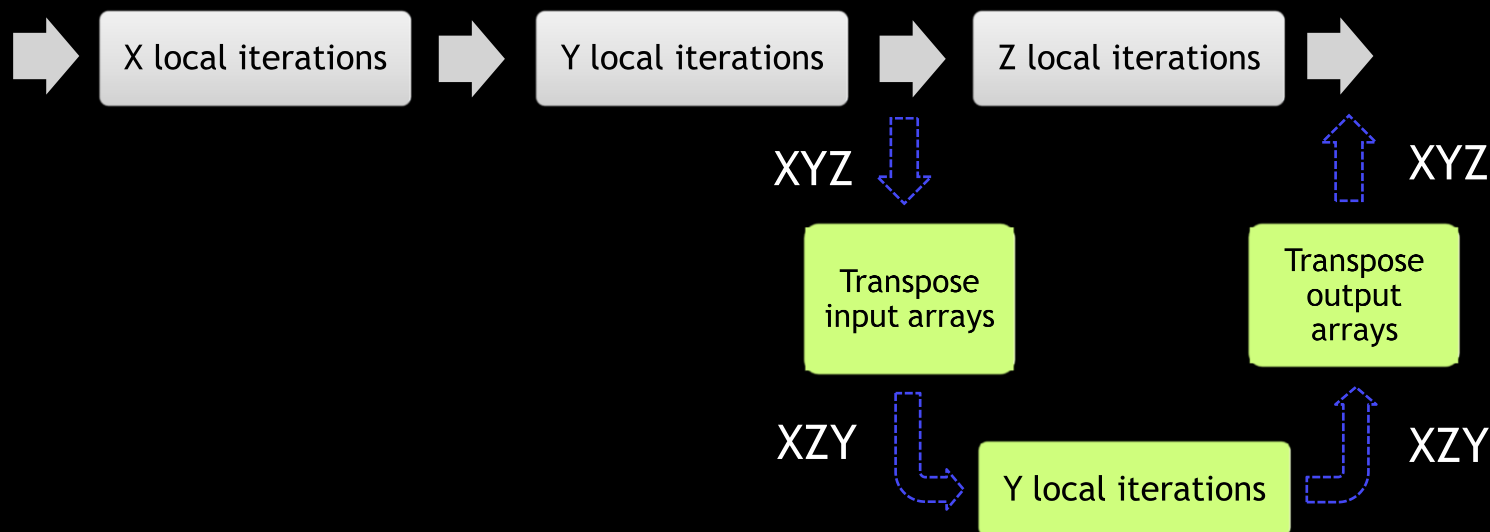
- Poor Z direction performance compared to X/Y
  - Threads access contiguous memory region
  - Memory access is uncoalesced, lots of cache misses

GPU TECHNOLOGY CONFERENCE

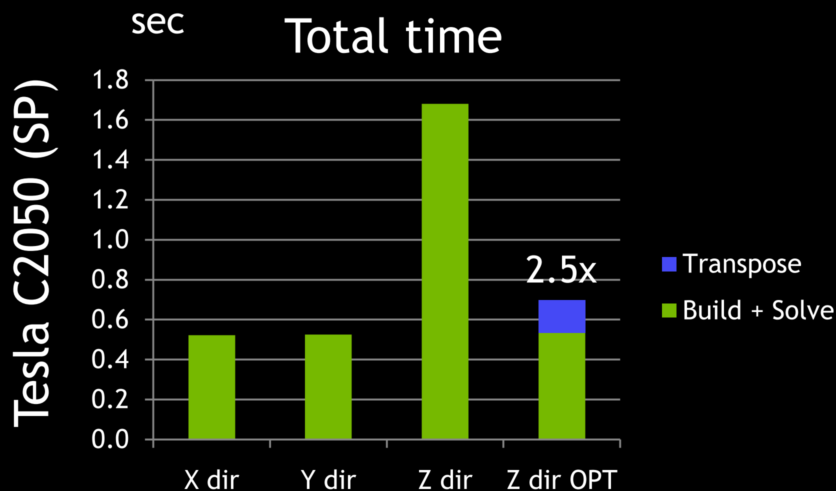
GPU TECHNOLOGY CONFERENCE

# 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

GPU TECHNOLOGY CONFERENCE

GPU TECHNOLOGY CONFERENCE

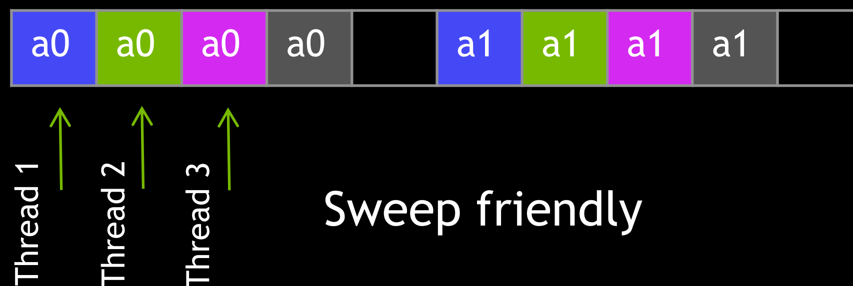
# Solving tridiagonal systems

- Number of tridiagonal systems ~ grid size squared
- Sweep algorithm is the most efficient in this case
  - 1 thread solves 1 system

```
for( int p = 1; p < end; p++ ) {
    // .. compute tridiagonal coefficients a_val, b_val, c_val, d_val ..
    get(c,p) = c_val / (b_val - a_val * get(c,p-1));
    get(d,p) = (d_val - get(d,p-1) * a_val) / (b_val - a_val * get(c,p-1));
}
for( int i = end-1; i >= 0; i-- )
    get(x,i) = get(d,i) -  get(c,i) * get(x, i+1);
```

# Solving tridiagonal systems

- Matrix layout is crucial for performance

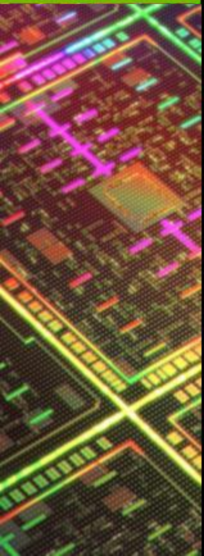


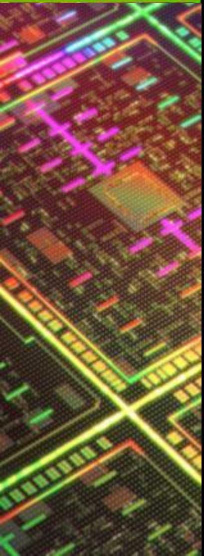

- X, Y directions matrices are interleaved by default
- Z is interleaved as well if doing in transposed space

GPU TECHNOLOGY CONFERENCE

# Solving tridiagonal systems

- 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





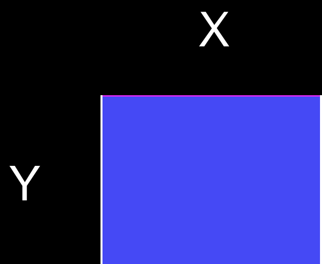
# Performance benchmark

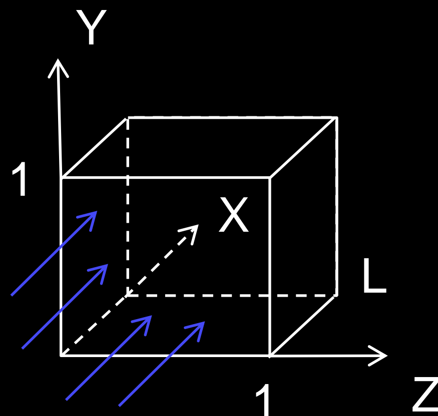
- CPU configuration:
  - Intel Core i7 4 cores @ 2.8 GHz
  - Use all 4 cores through OpenMP (3-4x speed-up vs 1 core)
- GPU configuration:
  - NVIDIA Tesla C2050
- Measure main kernel time for all iterations

GPU TECHNOLOGY CONFERENCE

# Test cases

- Cube



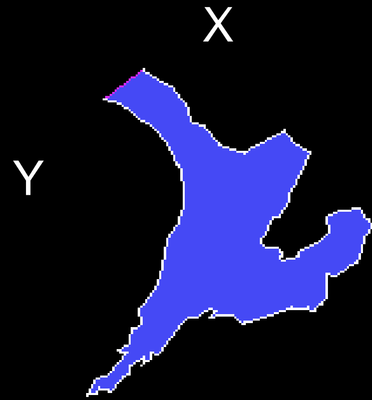



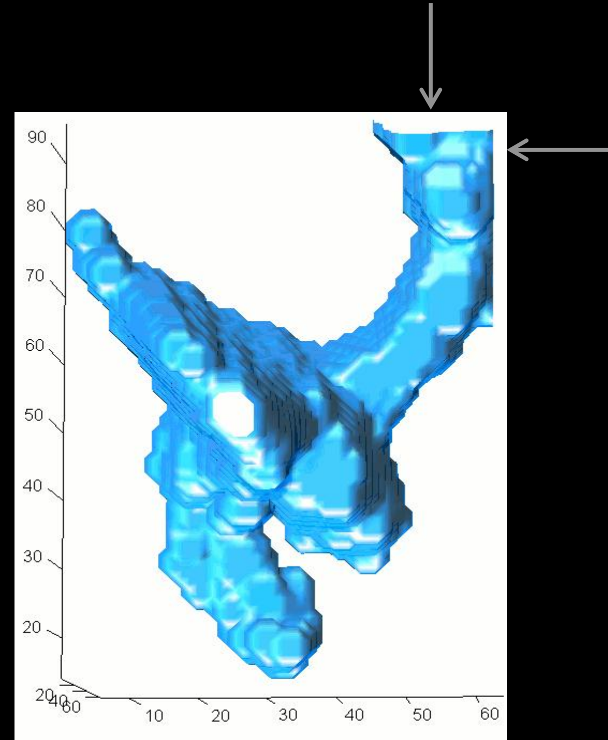

- Simple geometry
- Systems of the same size
- Need to compute in all rectangular grid points

GPU TECHNOLOGY CONFERENCE

# Test cases

- White Sea

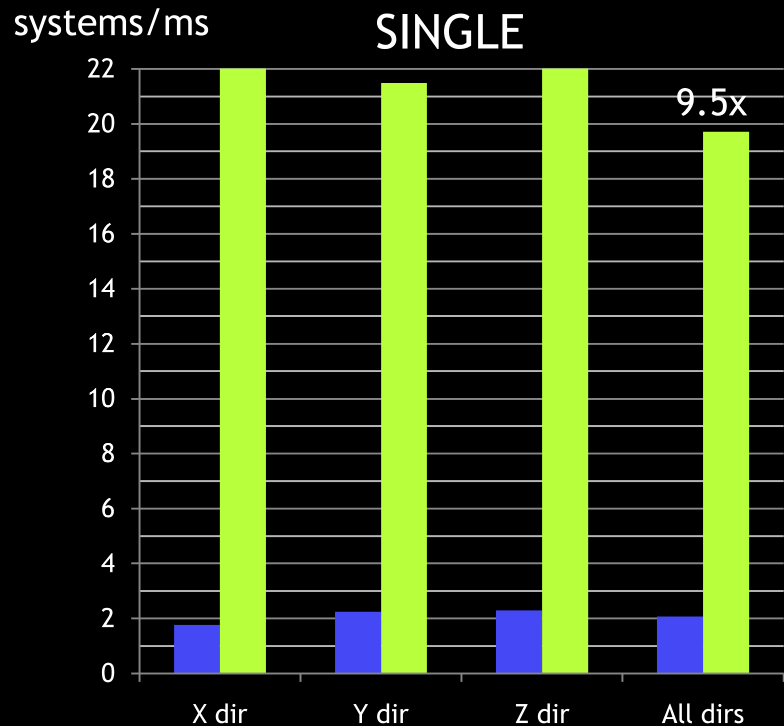



- Complex geometry
- Big divergence for system sizes
- Need to compute only inside the area

GPU TECHNOLOGY CONFERENCE

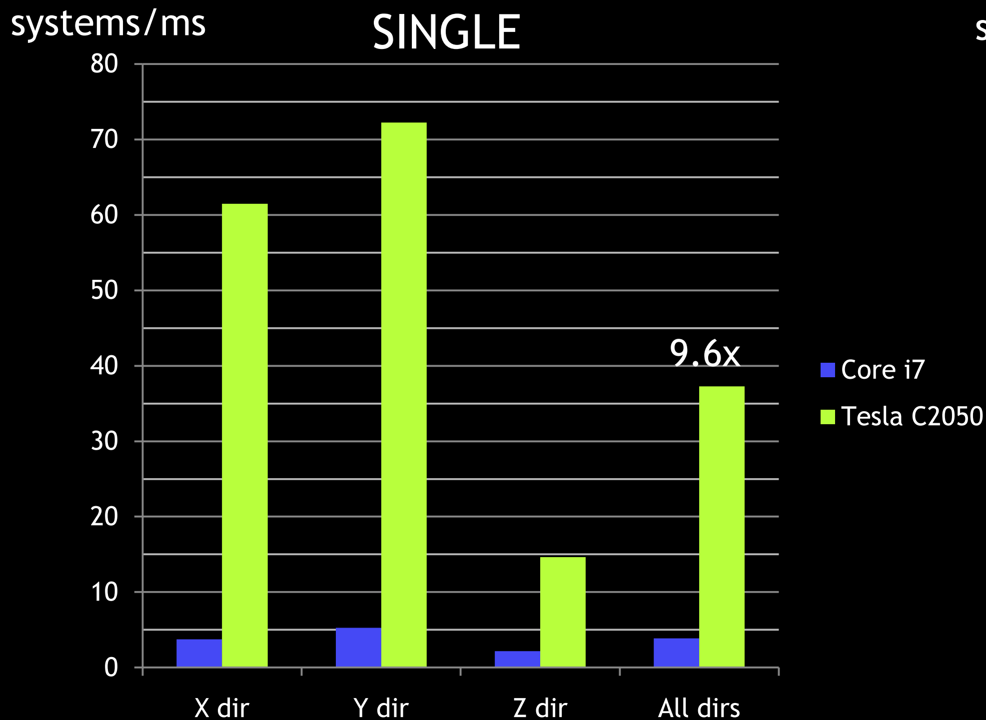
# Performance results - Cube

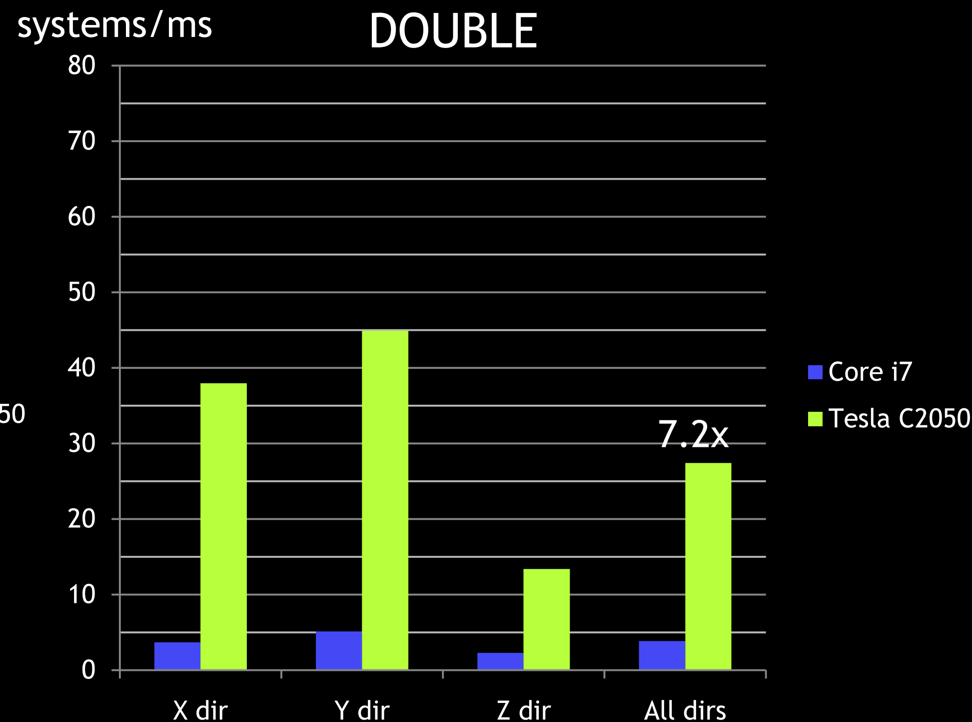


Grid 128x128x128

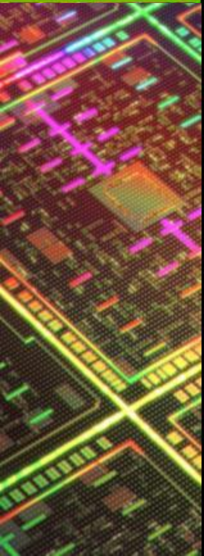
GPU TECHNOLOGY CONFERENCE

# Performance results - White Sea




Grid 160x128x128

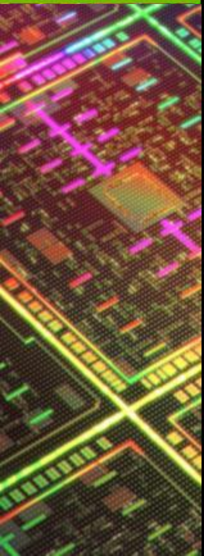


# Outline

- Fluid Simulation in 3D domain
- Multi-GPU implementation
  - General splitting algorithm
  - Running computations using CUDA 4.x
  - Benchmarking and performance analysis

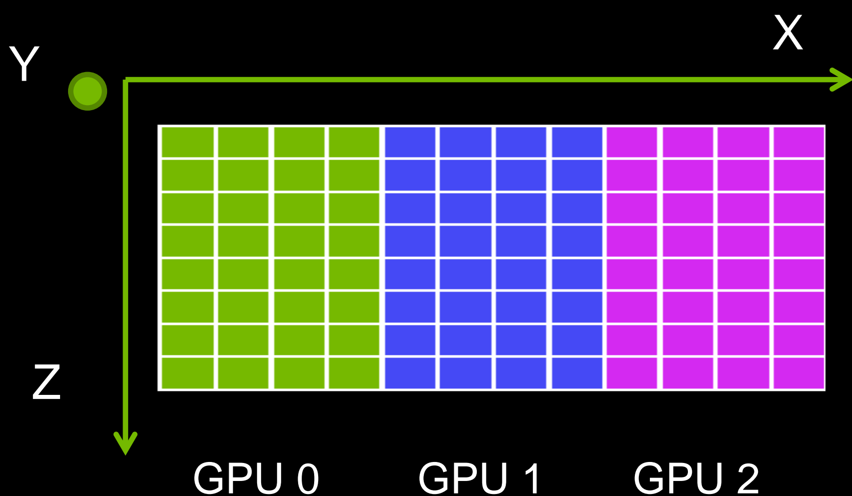
# Multi-GPU motivation

- Limited available amount of memory
  - 3D arrays: grid, temporary arrays, matrices
  - Max size of grid that can fit into Tesla M2050 ~ 224x224x224
- Distribute the computations between multiple GPUs and multiple nodes
  - Can compute large grids
  - Speed-up computations

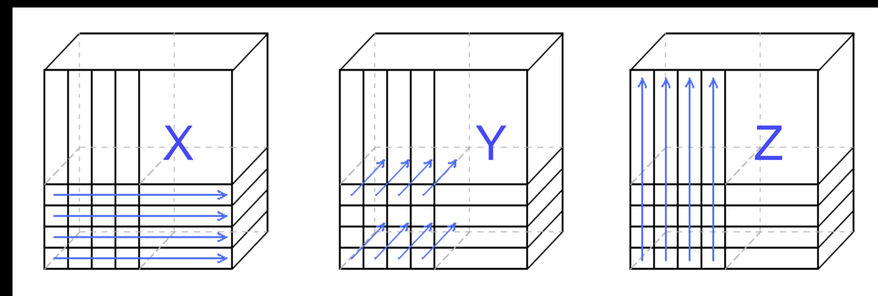


# Multi-GPU algorithm

- Solve systems along Y and Z directions independently in parallel on each GPU
  - No data exchange
- Synchronize data between multiple GPUs along X direction




Split steps for each direction:



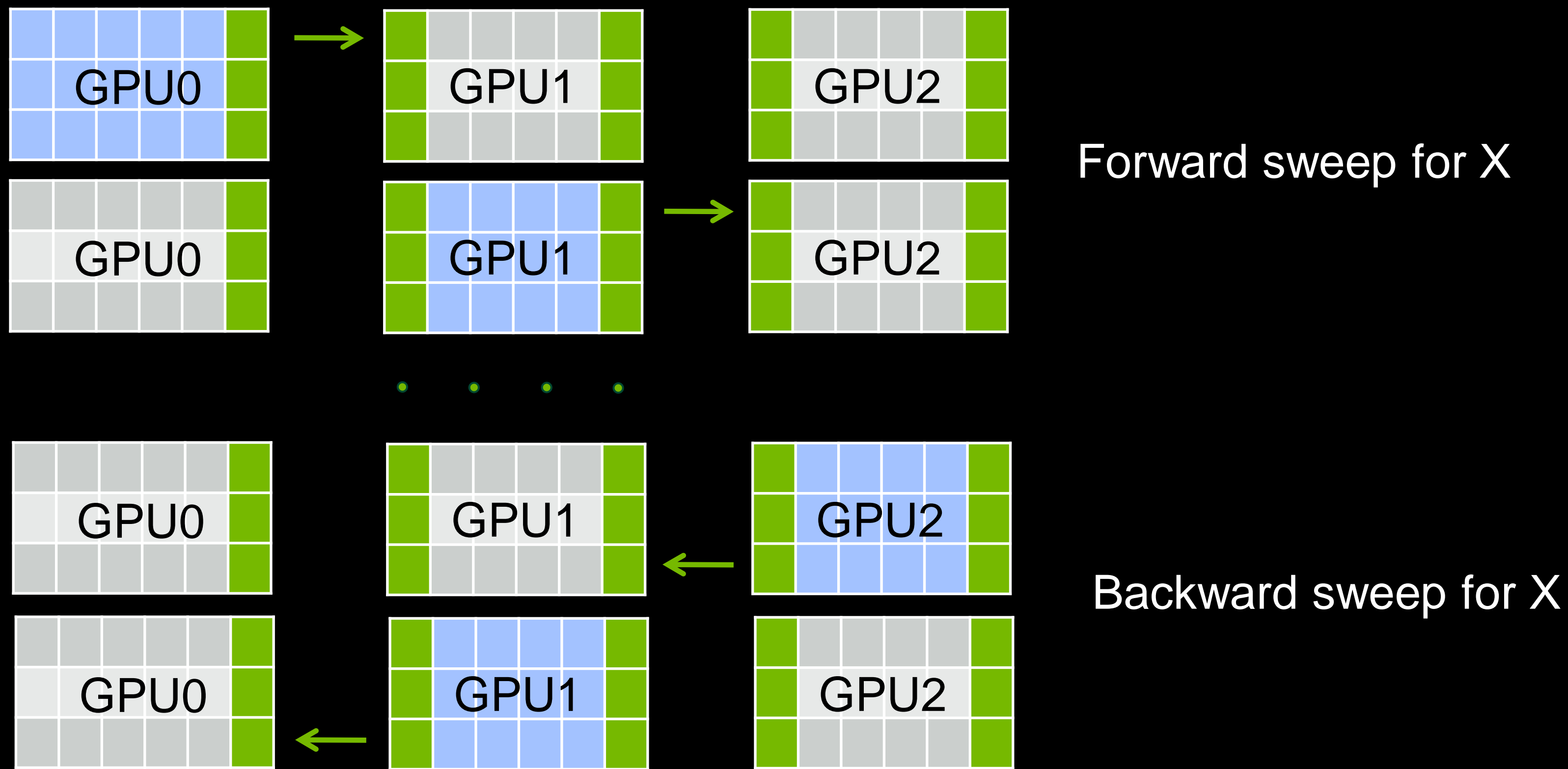

GPU TECHNOLOGY CONFERENCE

# Multi-GPU algorithm - Y, Z sweeps

- Slice the grid by X direction
  - Slowest-changing dimension
  - Z + Y * dimZ + X * dimY * dimZ
- Running the code on multiple GPUs
  - CUDA 4.x: from a single CPU thread

```
for ( int i = 0; i < gpuSize; i++ )
{
    cudaSetDevice( i );                              // switch to device
    solve_segments<dir, var, ALL><<<grid, block>>>( ... );
}
```

# Multi-GPU algorithm - X sweep




No speed-up for X direction sweeps

GPU TECHNOLOGY CONFERENCE

# Multi-GPU algorithm - X sweep

- Forward sweep

```
for ( int i = 0; i < gpuSize; i++ ) {
    cudaSetDevice( i );
    solve_segments<X, var, FORWARD><<<grid, block>>>( ... );
    if( i < gpuSize-1 ) {
        cudaMemcpyPeer( c .. );          // copy sweep coefficients to i+1
        cudaMemcpyPeer( d .. );
    }
}
```

- Backward sweep

```
for ( int i = gpuSize-1; i >= 0; i++ ) {
    cudaSetDevice( i );
    solve_segments<X, var, BACK><<<grid, block>>>( ... );
    if( i > 0 )
        cudaMemcpyPeer( x .. );          // propagate result to i-1
}
```

GPU TECHNOLOGY CONFERENCE

# Multi-GPU algorithm - halos

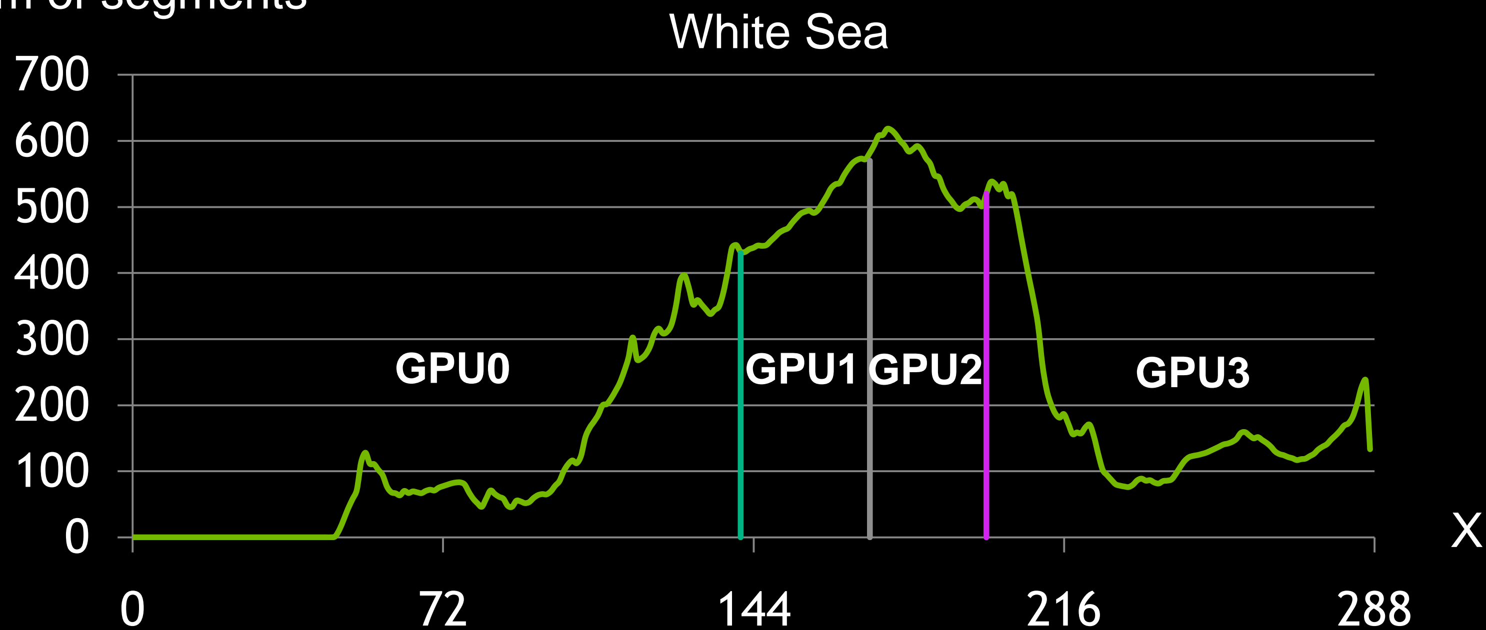
- Copy halos for non-linear layer between devices
  - Asynchronous, peer-to-peer over PCI-E

```
for( int i = 0; i < gpuSize; i++ ) {
    if( i < gpuSize - 1 )
        cudaMemcpyPeerAsync( dev_array[i+1],  i+1,
                             dev_array[i] + numTotalElems[i], i,
                             sizeof(T) * haloSize[i], fstream[i] );
    if( i > 0 )
        cudaMemcpyPeerAsync( dev_array[i-1] + haloSize + numTotalElems[i],  i-1,
                             dev_array[i] + haloSize, i,
                             sizeof(T) * haloSize[i], bstream[i] );
}
for( int i = 0; i < gpuSize; i++ ) {
    cudaSetDevice( i );
    cudaStreamSynchronize( fstream[i] );
    cudaStreamSynchronize( bstream[i] );
}
```

GPU TECHNOLOGY CONFERENCE

# Multi-GPU algorithm

- Distribute segments uniformly between multiple GPUs
  - Observed speed-up for 288x320x320 grid: +15%

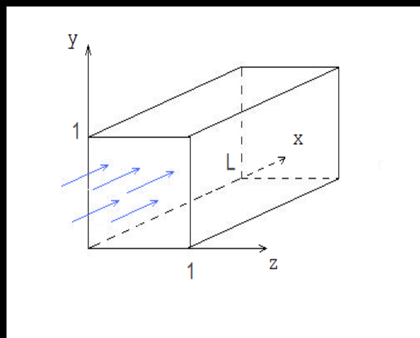


GPU TECHNOLOGY CONFERENCE

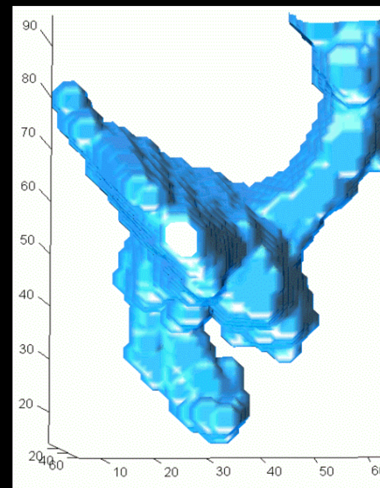
# Multi-GPU performance

- CPU: Intel X5670 @ 2.93 GHz
- P2P: 8 Tesla M2050 with P2P
- MPI: 4 InfiniBand MPI nodes, each has 1 Tesla M2090
- Test cases:

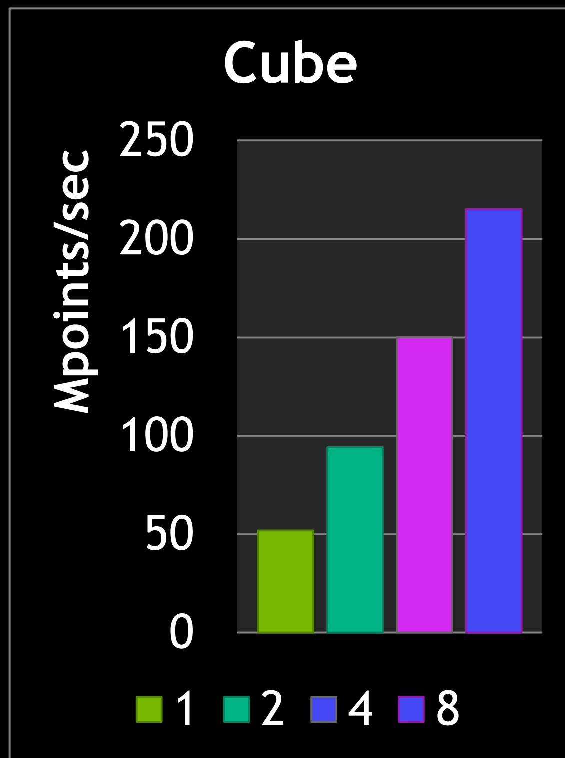
Cube

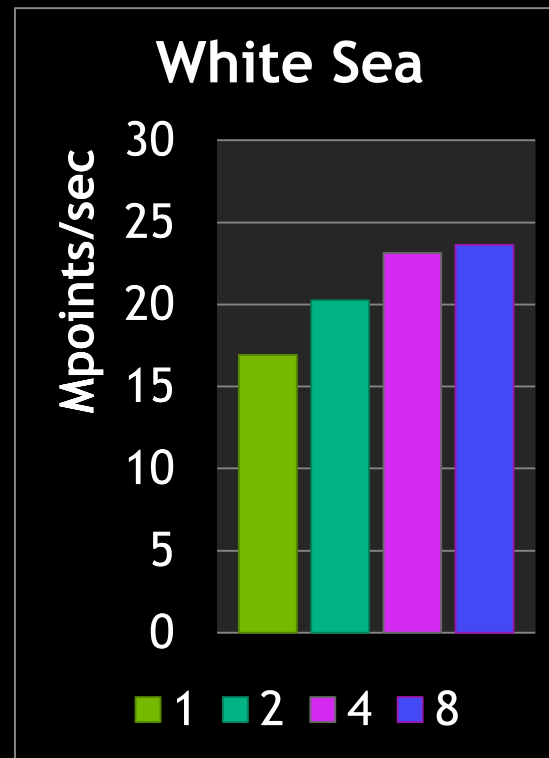



White Sea



# Results: 8 GPUs, 1 MPI node

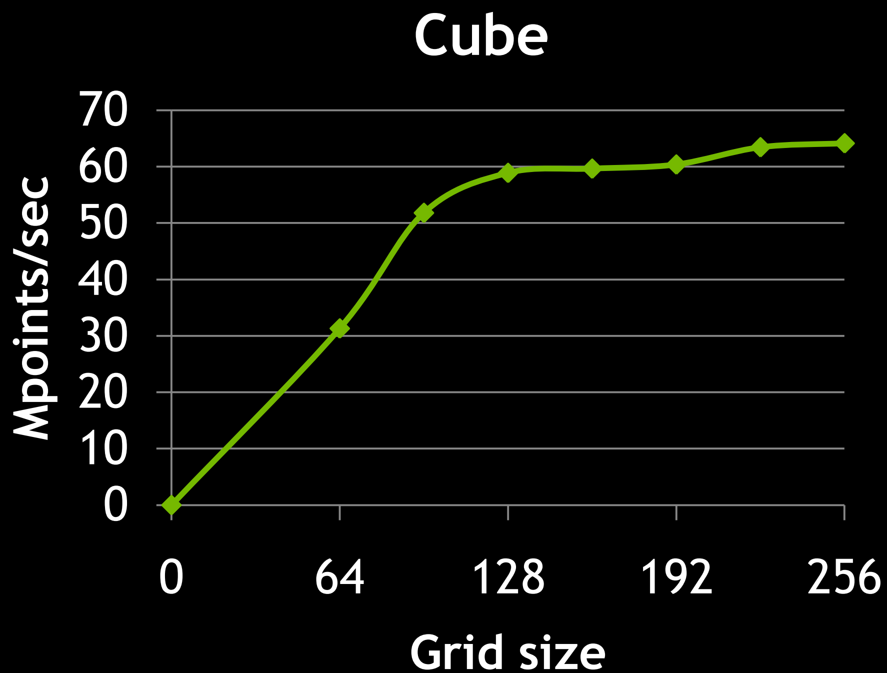


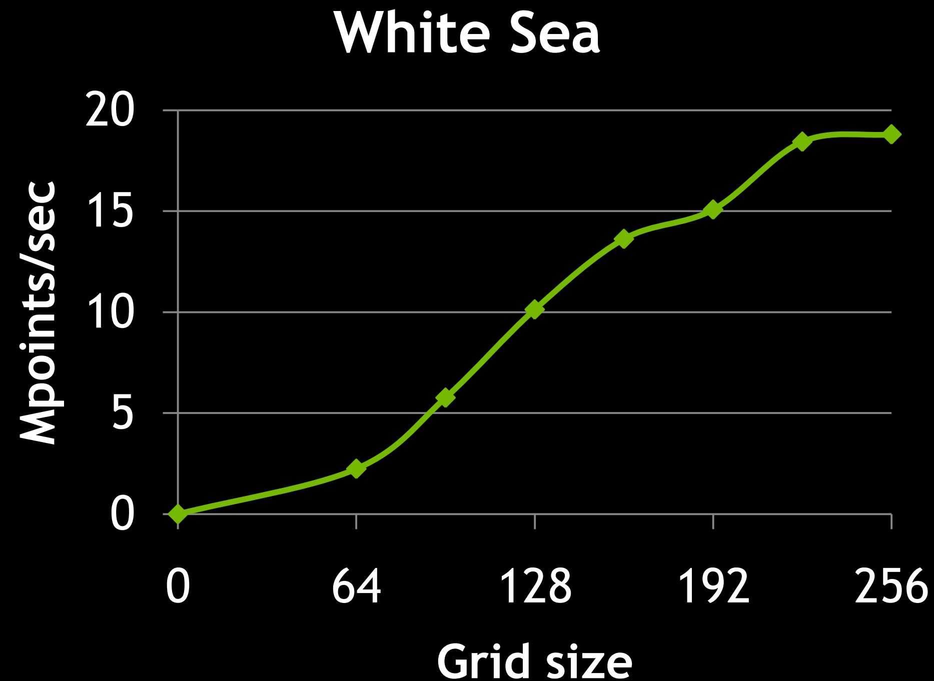




Grid 224x224x224,
Tesla M2050

GPU TECHNOLOGY CONFERENCE

# 1 GPU efficiency




Tesla M2090

GPU TECHNOLOGY CONFERENCE

GPU TECHNOLOGY CONFERENCE

# Using P2P

Cube

| | Sweep X, ms | NonLinear Sync, ms | Total, ms |
|---|---|---|---|
| **P2P** | 21.33 | 1.77 | **46.41** |
| **w/o P2P** | 25.35 | 7.95 | **56.61** |

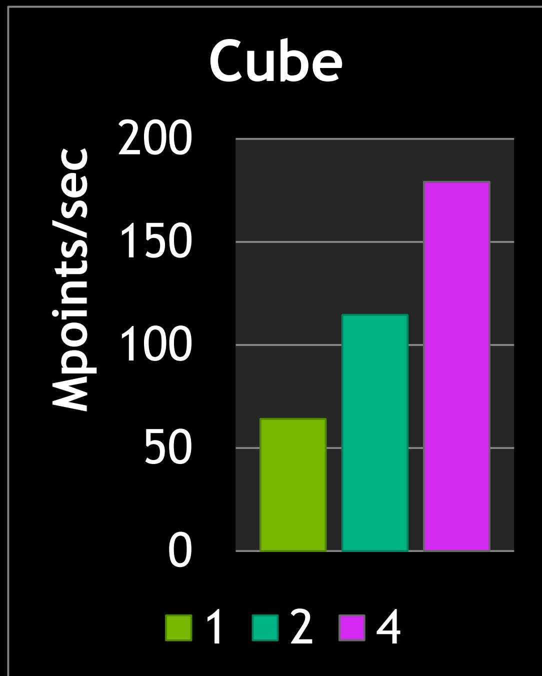
White Sea

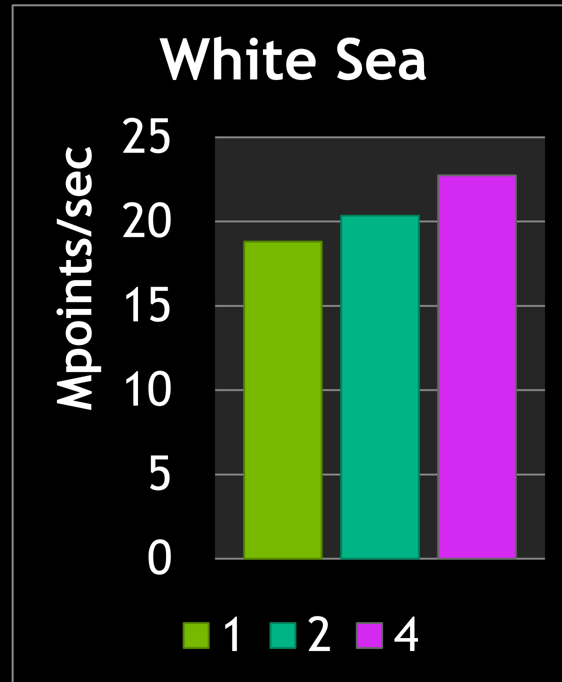
| | Sweep X, ms | NonLinear Sync, ms | Total, ms |
|---|---|---|---|
| **P2P** | 6.22 | 1.76 | **28.00** |
| **w/o P2P** | 9.78 | 7.92 | **36.91** |

Grid 224x224x224, Tesla M2050

GPU TECHNOLOGY CONFERENCE

# Results: 1 GPU, 4 MPI nodes




Grid 256x256x256,
Tesla M2090

GPU TECHNOLOGY CONFERENCE

# Using Infiniband

- mvapich2 vs mpich2

| | SweepX, ms | NonLinear Sync, ms | Total, ms |
|---|---|---|---|
| InfiniBand | 30.34 | 5.10 | **71.81** |
| MPI | 111.50 | 157.32 | **305.88** |

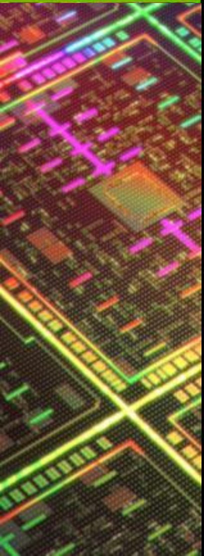
- Pinned host memory (cudaHostAlloc + CUDA_NIC_INTEROP=1)

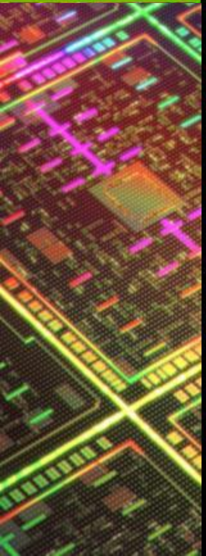
| | SweepX, ms | NonLinear Sync, ms | Total, ms |
|---|---|---|---|
| cudaHostAlloc | 31.00 | 5.10 | **71.81** |
| new/malloc | 35.39 | 9.75 | **75.12** |

Grid 256x256x256, Tesla M2090

# Results

- All parts of the algorithm except X sweep are well-parallelized
- Communication cost relatively small for 1-8 GPUs
  - Using P2P and Infiniband
- Performance and scaling factors heavily depend on input geometry and size of grid
  - Efficient work distribution methods are essential for performance





GPU TECHNOLOGY CONFERENCE

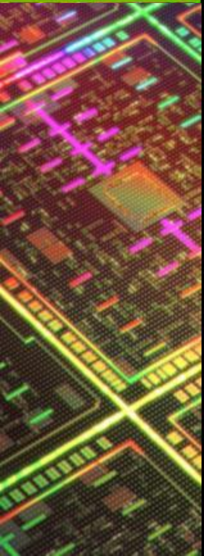
# Future work

- Test on large scale systems
  - Potentially on “Lomonosov” supercomputer at MSU
  - GPU part with peak performance of 863 TFlops
- Finish implementation of blocking algorithm
  - Eliminate waiting for sweep X by computing a few blocks for Y/Z
- Memory usage optimizations
- Explore different tridiagonal approaches

GPU TECHNOLOGY CONFERENCE

# References


- http://code.google.com/p/cmc-fluid-solver/
- "Efficient Tridiagonal Solvers for ADI methods and Fluid Simulation", GPU Technology Conference, 2010, Nikolai Sakharnykh
- "Tridiagonal Solvers on the GPU and Applications to Fluid Simulation", GPU Technology Conference, 2009, Nikolai Sakharnykh



# Summary

- 10x speed-ups 1 GPU vs 1 CPU
  - 3D ADI methods in application to fluid simulation in complex areas
- Enable large grids using multiple GPUs / nodes
  - CUDA 4.x made is easy to parallelize over multi-GPU
- Good scaling on systems with several GPUs

# Questions?

- Email: nsakharnykh@nvidia.com

Thursday, December 15, 2011, 15:00, Room 3

“Multi-GPU Cyclic Reduction for ADI Methods”, Joy Lee

GPU TECHNOLOGY CONFERENCE