Massively Parallel X-ray Scattering Simulations

Abhinav Sarje[1]

Xiaoye Sherry Li | Elaine Chan
Slim Chourou | Alexander Hexemer
[1]Computational Research | Advanced Light Source

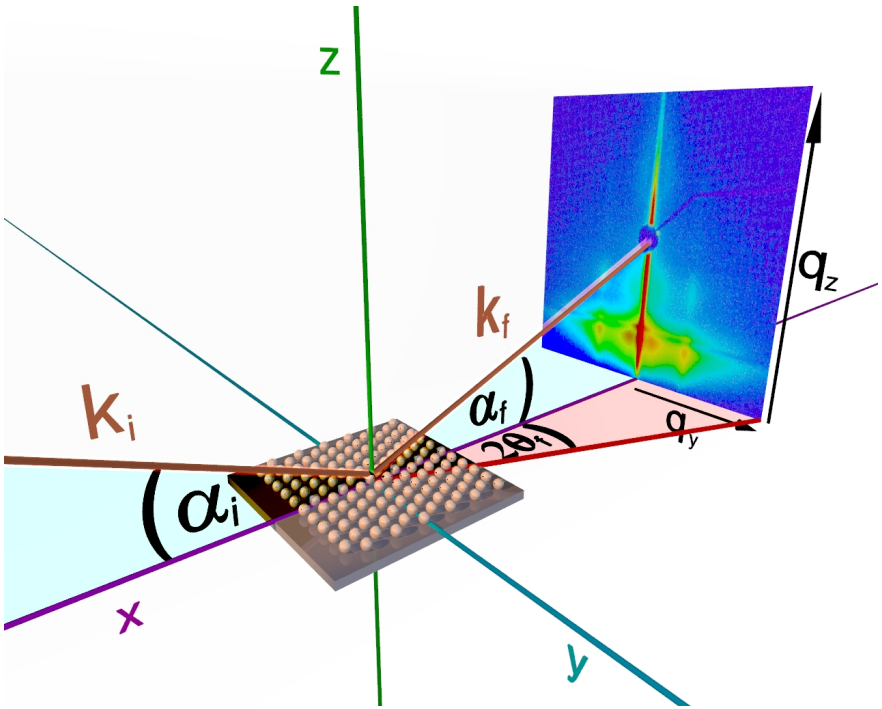
Lawrence Berkeley National Laboratory

11.14.12

Supercomputing 2012, Salt Lake City

Background: What is X-ray Scattering?

- Tool to measure structural properties of materials and characterize macromolecules and nano-particle systems at micro and nano-scales.
- Expose sample to high-energy X-rays, which get scattered.
- Light intensities are recorded as a scattering pattern.
- Contains sample structural information.
- A method: Grazing Incidence Small-angle X-ray Scattering (GISAXS).



graphic: courtesy of A. Meyer, *www.gisaxs.de*

Outline

1. Simulation of X-ray scattering patterns.
2. Motivations and challenges.
3. An HPC solution.
4. The main computational problem.
5. On GPU clusters.
6. On multi-core CPU clusters.
7. Performance & analysis.
8. Conclusions.

Simulations: Computing Scattered Light Intensities

Given

1. a sample structure model, and
2. experimental configuration,

simulate experiments and generate scattering patterns.

Based on *Distorted Wave Born Approximation* (DWBA) theory.

Q-grid: a 3D region grid in inverse space where scattered light intensities are to be computed.

Intensity: is computed at each Q-grid point $\vec{q}$.
At a point $\vec{q}$, it is proportional to square of the sum of *Form Factors* at $\vec{q}$, due to all structures in the sample:

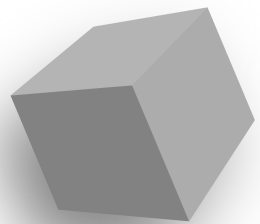
$$I(\vec{q}) \quad \propto \quad \left| \sum_{s=1}^{S} F(\vec{q}) \right|^2$$

Simulations: Computing Scattered Light Intensities

Given

1. a sample structure model, and
2. experimental configuration,

simulate experiments and generate scattering patterns.



Based on *Distorted Wave Born Approximation* (DWBA) theory.

Q-grid: a 3D region grid in inverse space where scattered light intensities are to be computed.

Intensity: is computed at each Q-grid point $\vec{q}$.
At a point $\vec{q}$, it is proportional to square of the sum of *Form Factors* at $\vec{q}$, due to all structures in the sample:

$$I(\vec{q}) \quad \propto \quad \left| \sum_{s=1}^{S} F(\vec{q}) \right|^2$$

Simulations: Computing Form Factors

- Form Factor at $\vec{q}$ is defined as an integral over shape surface.

$$F(\vec{q}) = -\frac{i}{|q|^2} \int_{S(\vec{r})} e^{i\vec{\mathbf{q}}\mathbf{r}\cdot\vec{r}} q_n(\vec{r}) d^2\vec{r}$$

- Approximated as a discretized surface (triangulated surface) integral:

$$F(\vec{q}) \approx -\frac{i}{|q|^2} \sum_{k=1}^{t} e^{i\vec{q}\cdot\vec{r}_k} q_{n,k}\sigma_k$$

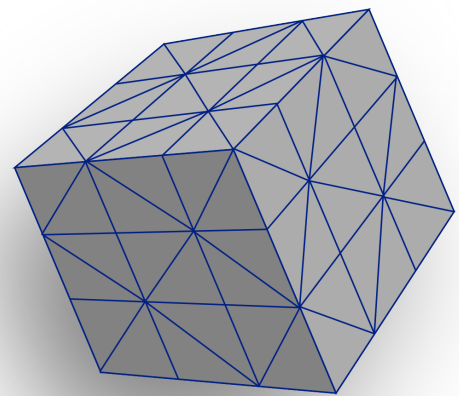
Simulations: Computing Form Factors

- Form Factor at $\vec{q}$ is defined as an integral over shape surface.

$$F(\vec{q}) = -\frac{i}{|q|^2} \int_{S(\vec{r})} e^{i\vec{\mathbf{q}}\mathbf{r}\cdot\vec{r}} q_n(\vec{r}) d^2\vec{r}$$

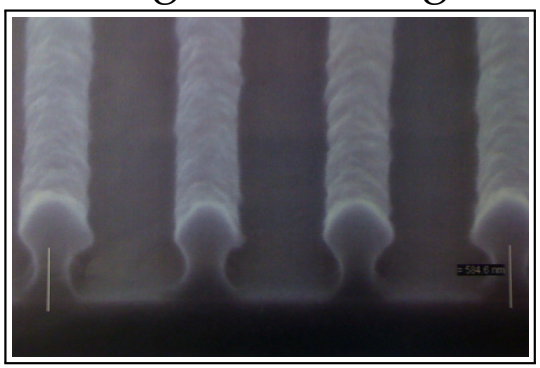
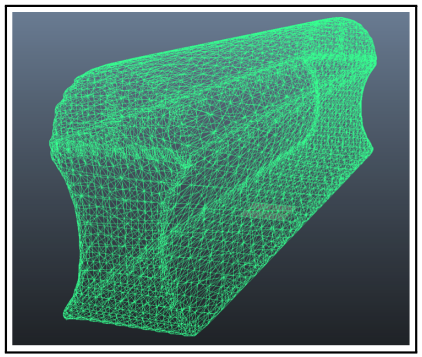
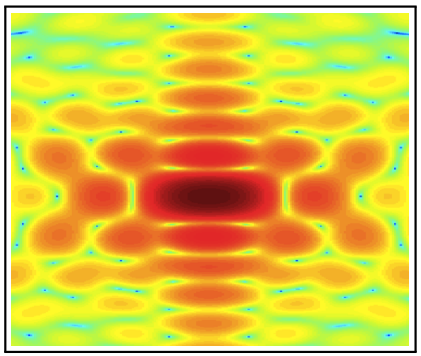
- Approximated as a discretized surface (triangulated surface) integral:

$$F(\vec{q}) \approx -\frac{i}{|q|^2} \sum_{k=1}^{t} e^{i\vec{q}\cdot\vec{r_k}} q_{n,k}\sigma_k$$

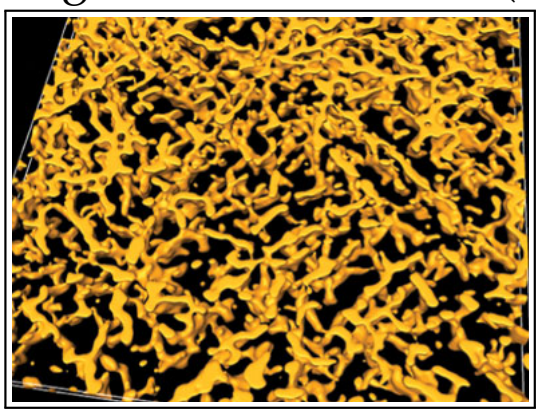
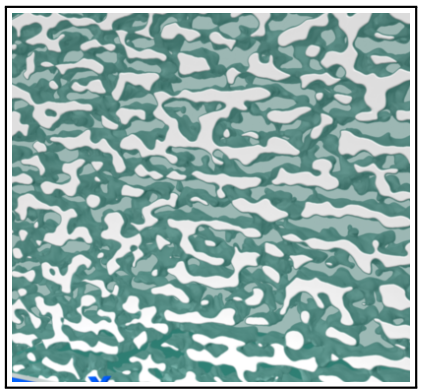
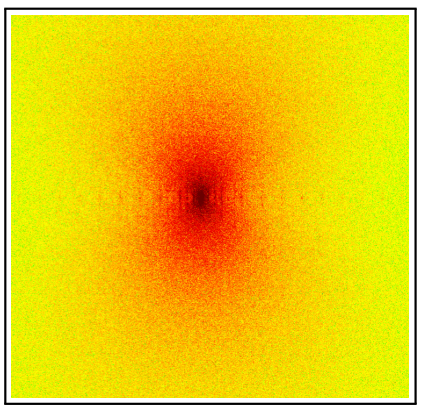


Simulations: Computed Scattering Pattern Examples

Rectangular Grating with Undercut:

Organic Photovoltaics (OPV) Tomography:

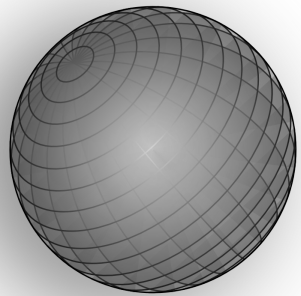




Real Sample | Model | Scattering Pattern

Why Simulate GISAXS Patterns?

- Structure prediction through fitting. Example:

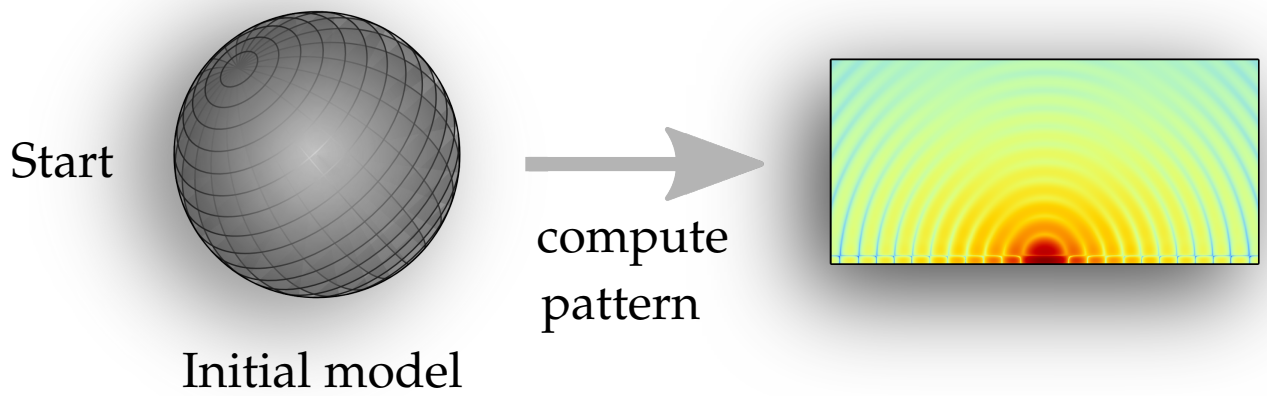
Start



Initial model

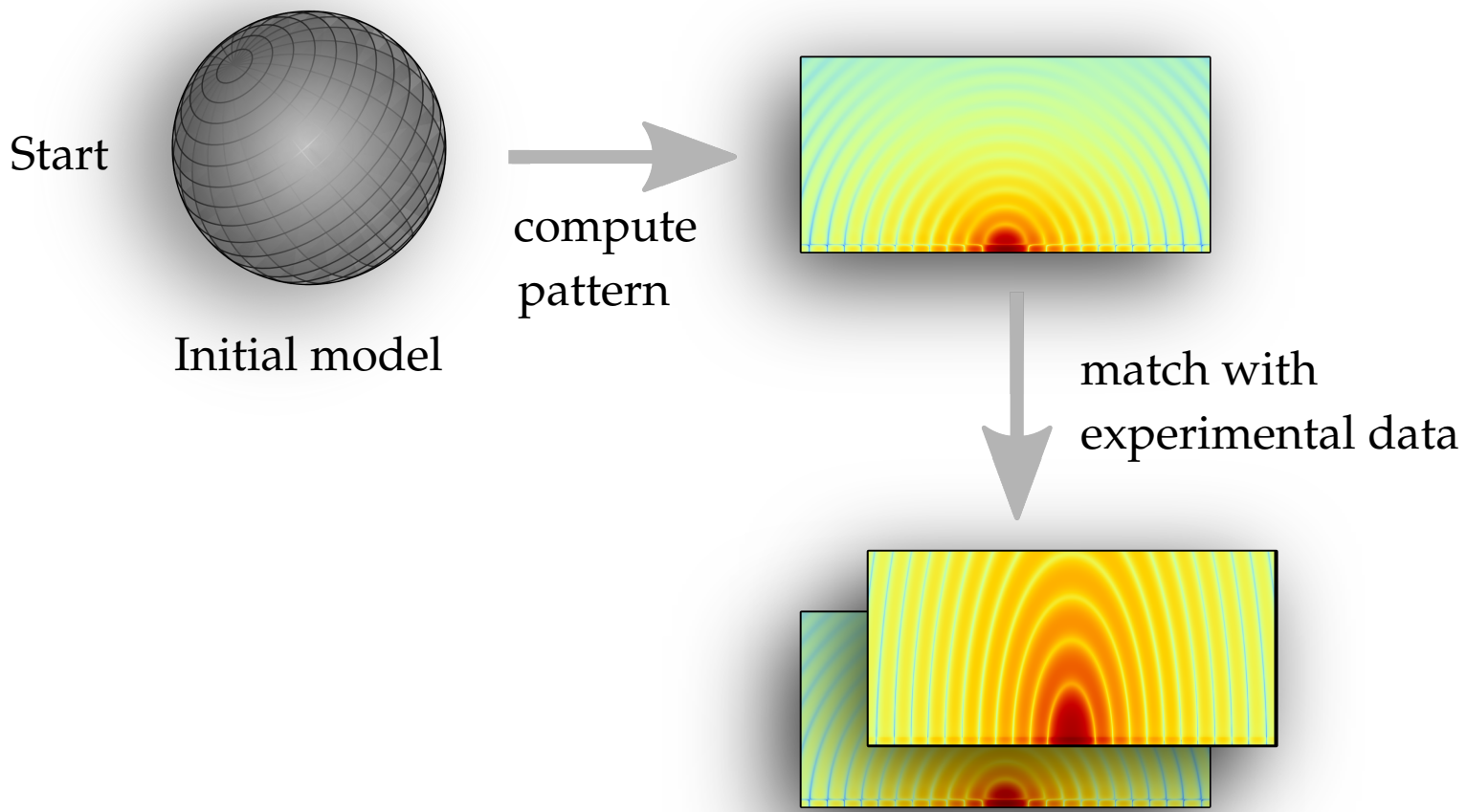
Why Simulate GISAXS Patterns?

- Structure prediction through fitting. Example:



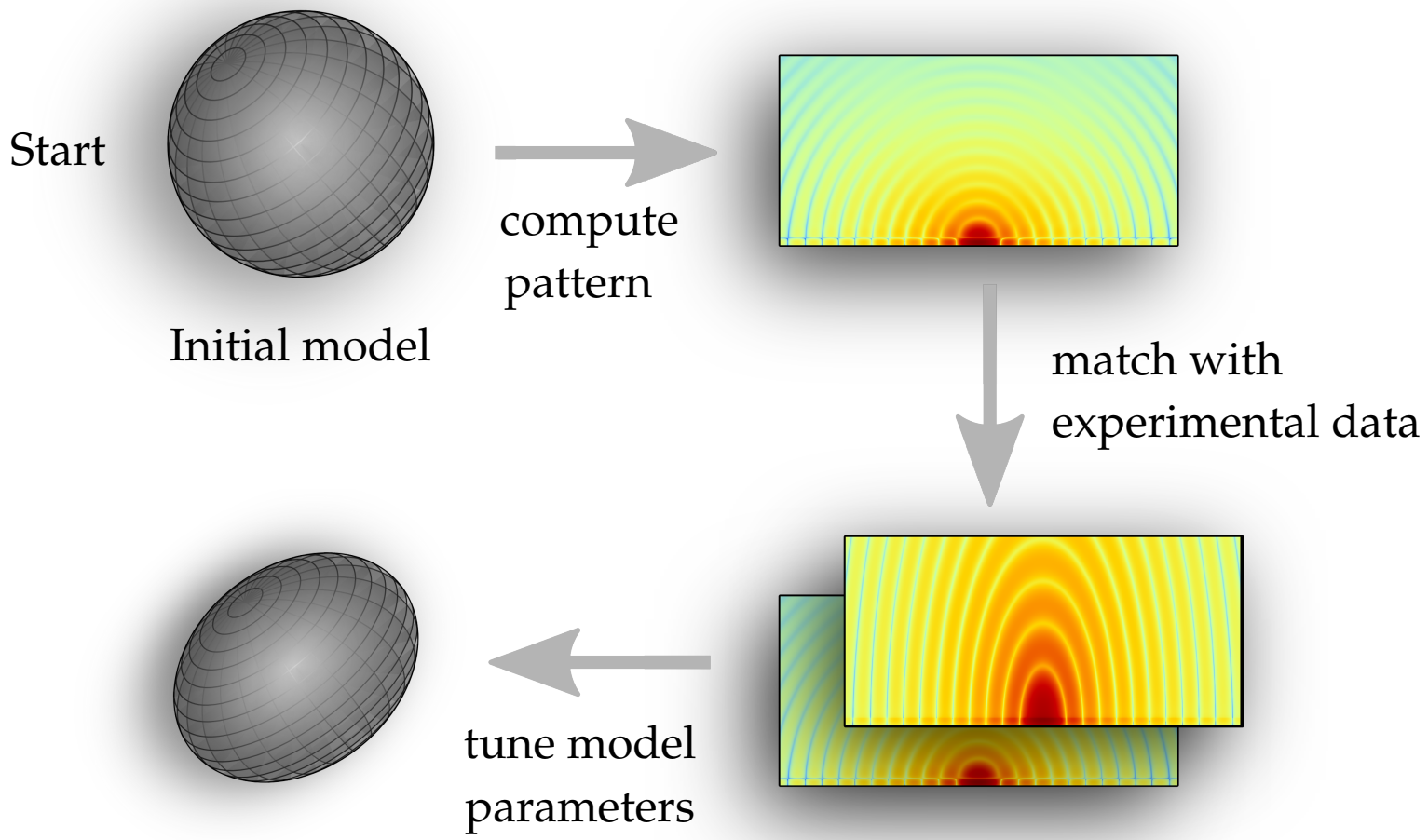
Why Simulate GISAXS Patterns?

- Structure prediction through fitting. Example:



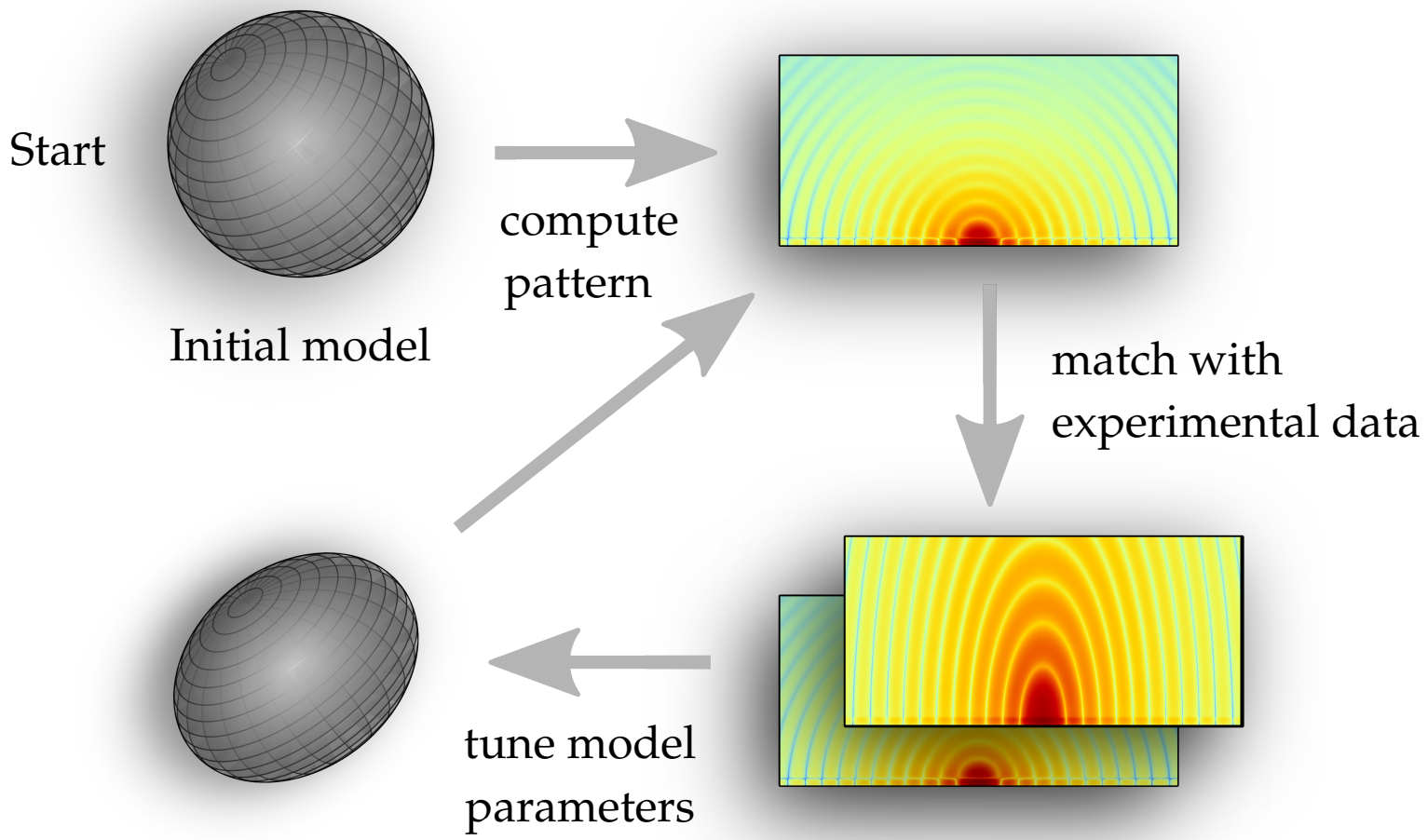
Why Simulate GISAXS Patterns?

- Structure prediction through fitting. Example:



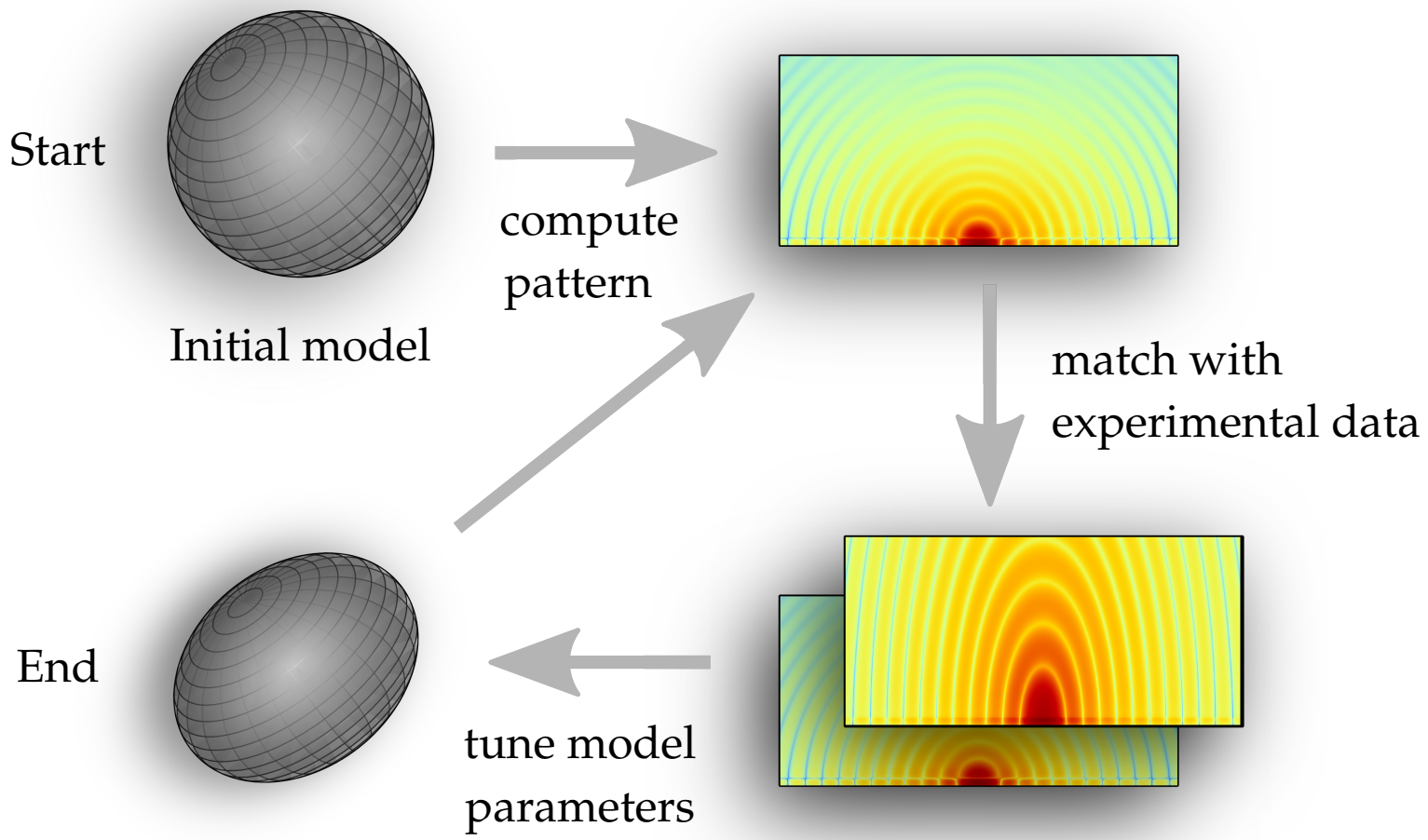
Why Simulate GISAXS Patterns?

- Structure prediction through fitting. Example:



Why Simulate GISAXS Patterns?

- Structure prediction through fitting. Example:



Why Parallelize and Accelerate?

Mismatch of data generation and data processing rates:

- High measurement rates of current state-of-the-art light sources.
- Extremely inefficient utilization of facilities due to mismatch.
- *Example*: Detectors at Linac Coherent Light Source (LCLS) in Stanford can generate 100 MB/s. Collects 12 TB per week.
- *Example*: Next Generation Light Source (NGLS) at Berkeley Lab envisions even higher data collection rates.

Accuracy Requirements

- Global error is proportional to the largest triangle's circum-diameter → increase triangulation resolution.
- For constant relative discretization error, finer triangulation requires higher Q-grid resolution → increase Q-grid resolution.

Why Parallelize and Accelerate?

Mismatch of data generation and data processing rates:

- High measurement rates of current state-of-the-art light sources.
- Extremely inefficient utilization of facilities due to mismatch.
- *Example*: Detectors at Linac Coherent Light Source (LCLS) in Stanford can generate 100 MB/s. Collects 12 TB per week.
- *Example*: Next Generation Light Source (NGLS) at Berkeley Lab envisions even higher data collection rates.

Accuracy Requirements

- Global error is proportional to the largest triangle's circum-diameter $\rightarrow$ increase triangulation resolution.
- For constant relative discretization error, finer triangulation requires higher Q-grid resolution $\rightarrow$ increase Q-grid resolution.

Why Parallelize and Accelerate?

High Computational Requirements

- Computational complexity = $O(nt)$
  n = number of q-points, t = number of triangles.
- Number of triangles = $O(10^3)$ to $O(10^6)$.
- Q-grid resolution = $O(10^4)$ to $O(10^8)$ points.
- Compute $O(10^7)$ to $O(10^{14})$ form factors for one experiment.
- Perform $O(10^2)$ of experiments for one sample.

Science Gap

- Beam-line scientists lack access to fast algorithms and codes.
- Previously existing codes are limited in compute capabilities.
- Also, they are slow – wait for days and weeks to obtain results.

Why Parallelize and Accelerate?

High Computational Requirements

- Computational complexity = $O(nt)$
  n = number of q-points, t = number of triangles.
- Number of triangles = $O(10^3)$ to $O(10^6)$.
- Q-grid resolution = $O(10^4)$ to $O(10^8)$ points.
- Compute $O(10^7)$ to $O(10^{14})$ form factors for one experiment.
- Perform $O(10^2)$ of experiments for one sample.

Science Gap

- Beam-line scientists lack access to fast algorithms and codes.
- Previously existing codes are limited in compute capabilities.
- Also, they are slow – wait for days and weeks to obtain results.

Goals and Major Challenges

- Achieve near real-time computations.
- Perform large number of computations.
- Optimize already "embarrassingly parallel" computations.
- Use limited system memory.
- Minimize expensive communication and data transfers.
- Scale to state-of-the-art massively parallel systems (and be future ready).

A solution ...

HipGISAXS: A High-Performance GISAXS Code

- Solves many limitations of previous codes.
- Implements new flexible algorithms to handle
  - any complex morphology,
  - multi-layered structures, and
  - all sample rotation directions and beam angles.
- Implements parallelization methods:
  - Deliver high-performance on massively parallel state-of-the-art clusters of GPUs and multi-core CPUs.
  - Bring computational time down to just seconds and minutes.
- Written in C++ with MPI, OpenMP and NVIDIA CUDA.
- Flexible and modularized code for future extensions.

Computational Problem: The Form Factor Kernel

Input: 3 arrays, q_x, q_y, q_z of lengths n_x, n_y, n_z, resp., representing a Q-grid of resolution $n = n_x \times n_y \times n_z$, and
An array defining the triangulated shape surface as a set of t triangles.

Output: A 3-D matrix M of size $n_x \times n_y \times n_z$, where each $M(i,j,k) = F(q_i, q_j, q_k) = F(\vec{q}_{i,j,k})$.

Environment: p node cluster of GPUs/multi-core CPUs.

Computational Problem: The Form Factor Kernel

Input: 3 arrays, q_x, q_y, q_z of lengths n_x, n_y, n_z, resp., representing a Q-grid of resolution $n = n_x \times n_y \times n_z$, and
An array defining the triangulated shape surface as a set of t triangles.

Output: A 3-D matrix M of size $n_x \times n_y \times n_z$, where each $M(i,j,k) = F(q_i, q_j, q_k) = F(\vec{q}_{i,j,k})$.

Environment: p node cluster of GPUs/multi-core CPUs.

Computational Problem: The Form Factor Kernel

Input: 3 arrays, q_x, q_y, q_z of lengths n_x, n_y, n_z, resp., representing a Q-grid of resolution $n = n_x \times n_y \times n_z$, and
An array defining the triangulated shape surface as a set of t triangles.

Output: A 3-D matrix M of size $n_x \times n_y \times n_z$, where each $M(i,j,k) = F(q_i, q_j, q_k) = F(\vec{q}_{i,j,k})$.

Environment: p node cluster of GPUs/multi-core CPUs.

Computational Problem: The Form Factor Kernel

Input: 3 arrays, q_x, q_y, q_z of lengths n_x, n_y, n_z, resp., representing a Q-grid of resolution $n = n_x \times n_y \times n_z$, and
An array defining the triangulated shape surface as a set of t triangles.

Output: A 3-D matrix M of size $n_x \times n_y \times n_z$, where each $M(i,j,k) = F(q_i, q_j, q_k) = F(\vec{q}_{i,j,k})$.

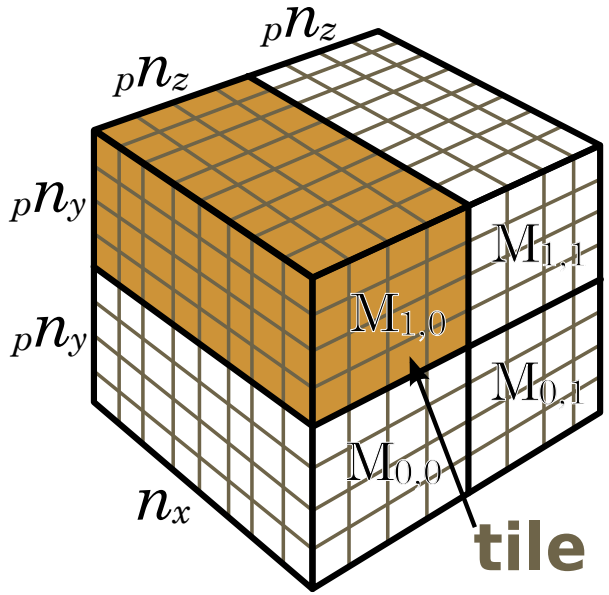
Environment: p node cluster of GPUs/multi-core CPUs.

On clusters of GPUs ...

Computation Decomposition Hierarchy

1. Across Multiple Nodes/Processes: *Tiling*

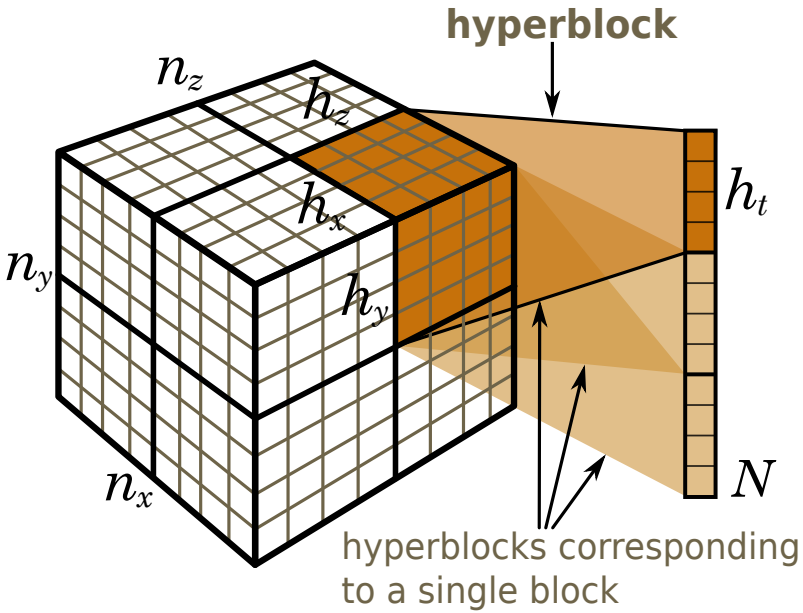
- Partition M along y and z dimensions into grid of $P = P_y \times P_z$ *tiles*.
- x dimension is typically small.
- Tile $M_{i,j}$ is assigned to node $P_{i,j}$.
- Tile data is distributed to respective nodes using MPI.



Computation Decomposition Hierarchy

2. Handle Memory Limitations: *Blocking*

- Data may not fit in device memory.
- Partition local tile along x, y, and z into *blocks* of size $h_x \times h_y \times h_z$.
- Partition triangle array into *segments* of size h_t.
- Represent combinations of blocks and segments as 4D *hyperblocks*.



- Process one hyperblock at a time on device.
- Hyperblocks result in partial sums. All partial sums for a block are reduced on host.

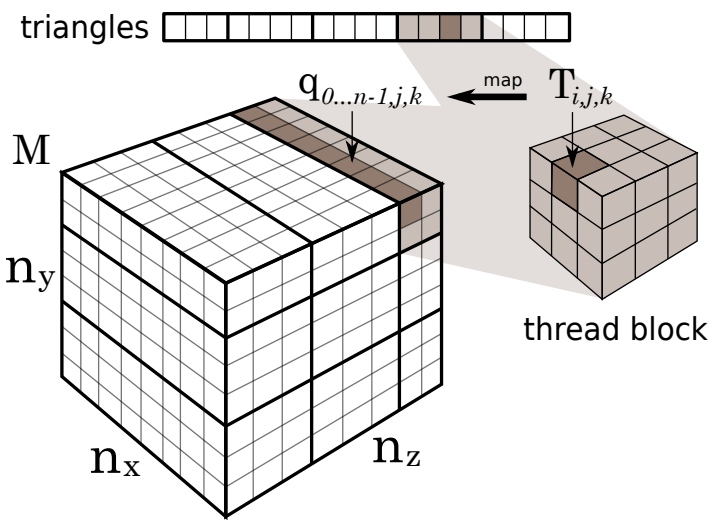
Computation Decomposition Hierarchy

3. Within device: *threading*

Phase 1 Local Computations.

- Partition along y, z and t into *thread blocks*.
- Compute over a triangle at all grid-points $\vec{q}$ in x dimension:

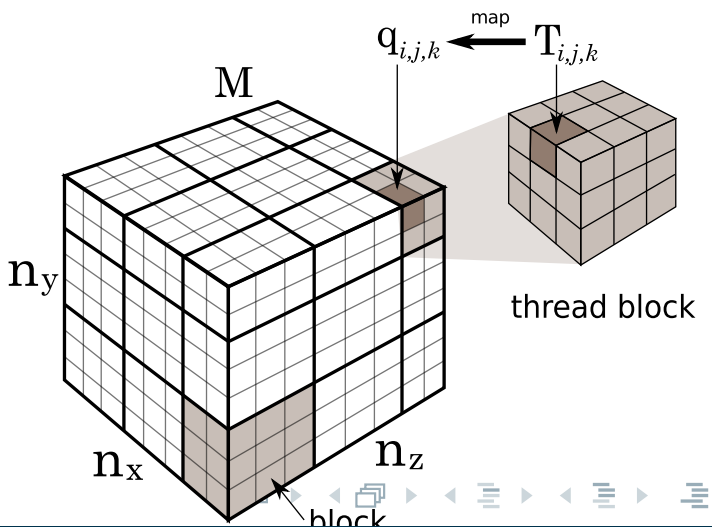
$$F_t(\vec{q}) = e^{i\vec{q}\cdot\vec{r}} s_t$$



Phase 2 Reduction.

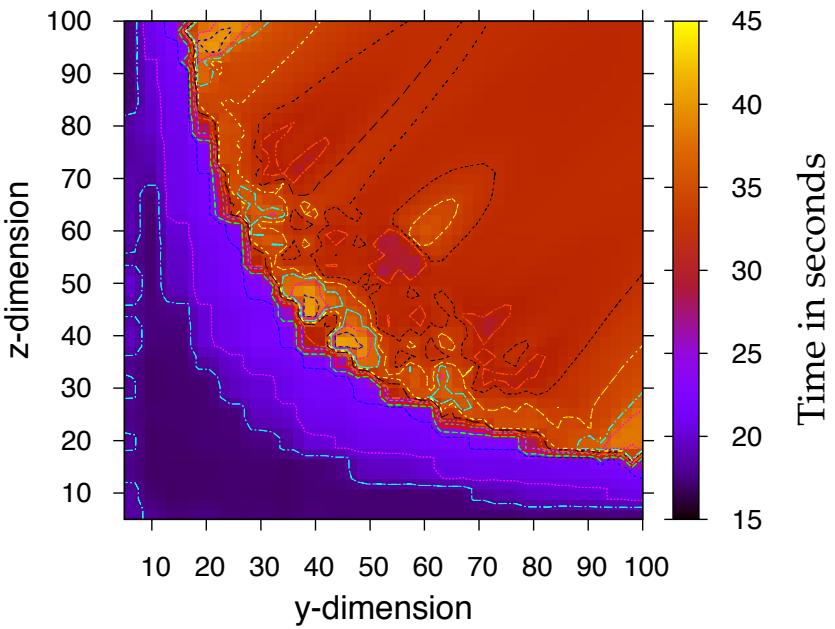
- Partition along x, y and z into *thread blocks*.
- Reduce all F_t at a grid-point $\vec{q}$:

$$F(\vec{q}) \approx \sum_{t=1}^{h_t} F_t(\vec{q})$$

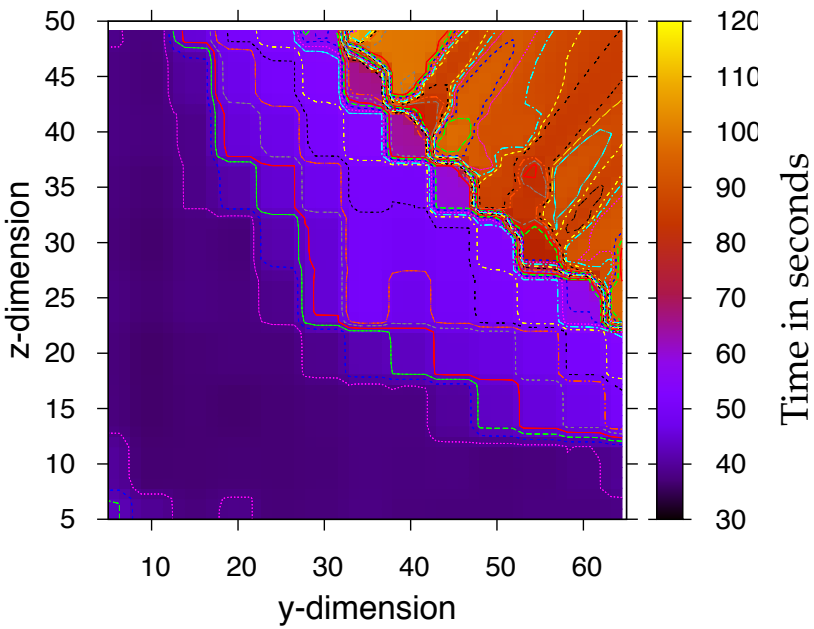


Optimizations: Choosing Hyperblock Size $h_x \times h_y \times h_z \times h_t$

- Crucial for high performance.
- Small size = low parallelism + large number of data transfers.
- Large size = transfer of large amounts of data.
- Find a good balance, explore the search space.
- Example heat maps of runtimes with varying h_y and h_z (4M q-points.)



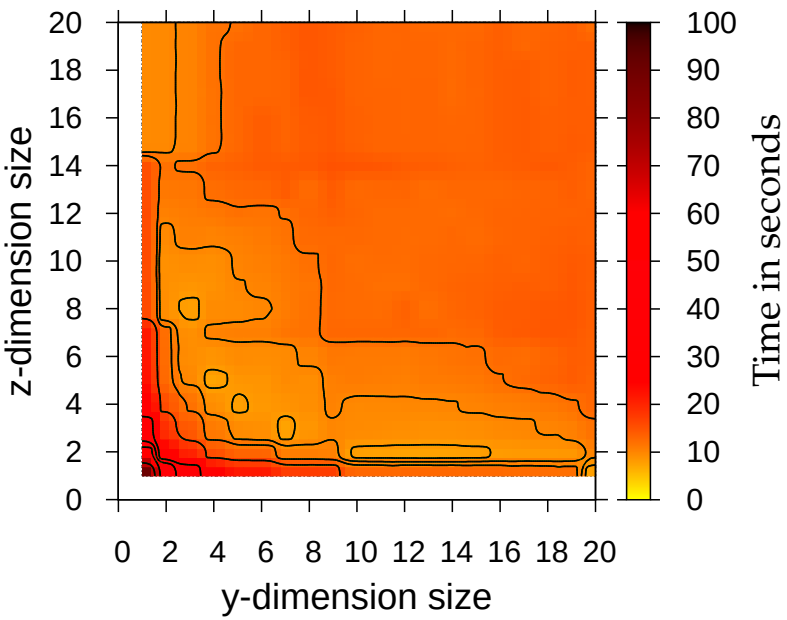
$t = 2,292$



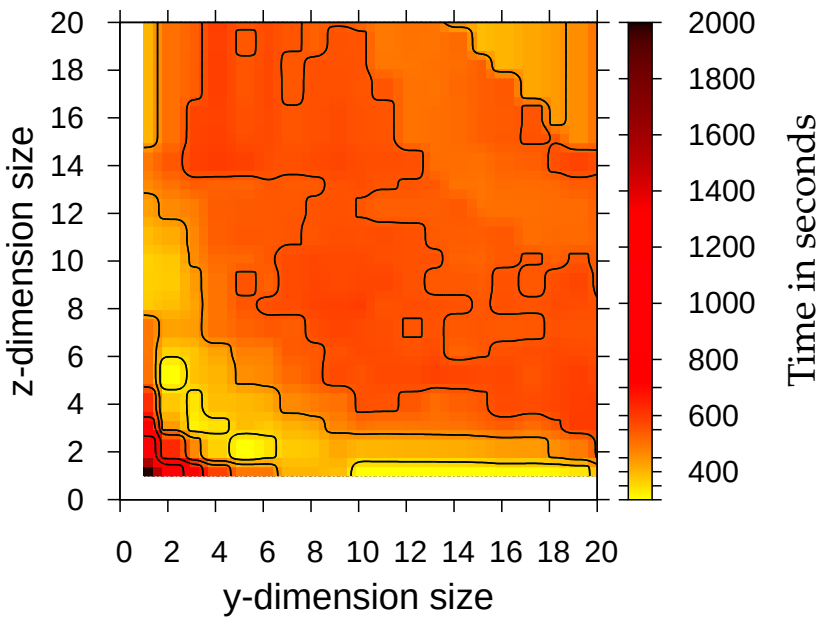
$t = 91,753$

Optimizations: Choosing Thread Block Sizes

- Also crucial for high performance.
- Small size = not enough threads in warps, or small number of warps.
- Large size = small number of thread blocks (less parallelism).
- Find a balanced size, explore search space.
- Example runtime heat maps with varying thread block sizes.



$t = 2,292$



$t = 91,753$

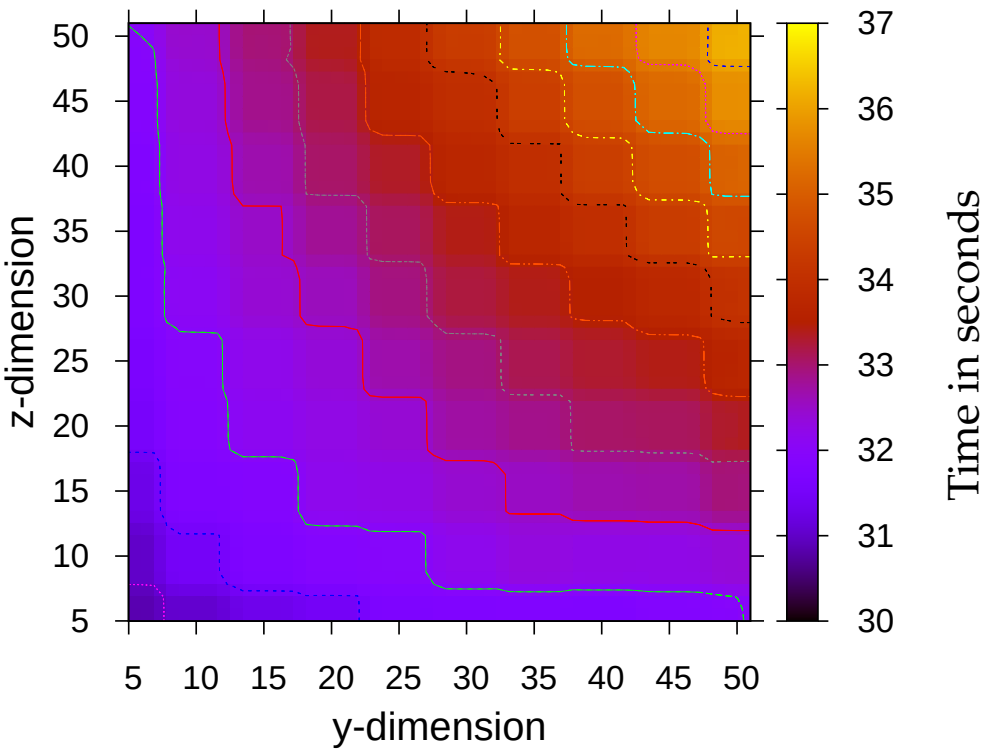
On clusters of multi-cores ...

Computation Decomposition Hierarchy

- A lot simpler!
- Partition into *tiles* and *hyperblocks*.
- Use MPI across tiles, and OpenMP within a hyperblock.
- Exploit NUMA design: choose number of MPI processes per node, number of threads per MPI process.
- Important to make effective use of caches. E.g.
  - Preserve input data locality – blocking.
  - Loop transformations.

Again, Choosing Hyperblock Size

- Not as crucial as for GPUs.
- Attributed to the large L1, L2 and LLC caches.



HipGISAXS performance ...

Experimental Environments

- **GPU Cluster:** *"TitanDev"*. Up to 930 nodes.
  - NVIDIA Tesla X2050 Fermi GPUs,
  - 6 GB device memory,
  - 1.15 GHz CUDA core clock,
  - AMD Opteron Interlagos 16 core CPU,
  - 32 GB main memory,
  - Gemini interconnects.
- **CPU Cluster:** *"Hopper"*. Cray XE6. Up to 6,000 nodes.
  - Dual AMD Opteron MagnyCours 12-core CPUs (total 24 cores),
  - 2.1 GHz clock,
  - 64 KB L1 and 512 KB L2 per core, 6 MB L3 shared by 6 cores,
  - 32 GB main memory,
  - Gemini interconnects.
- Single precision complex number computations.

Experimental Environments

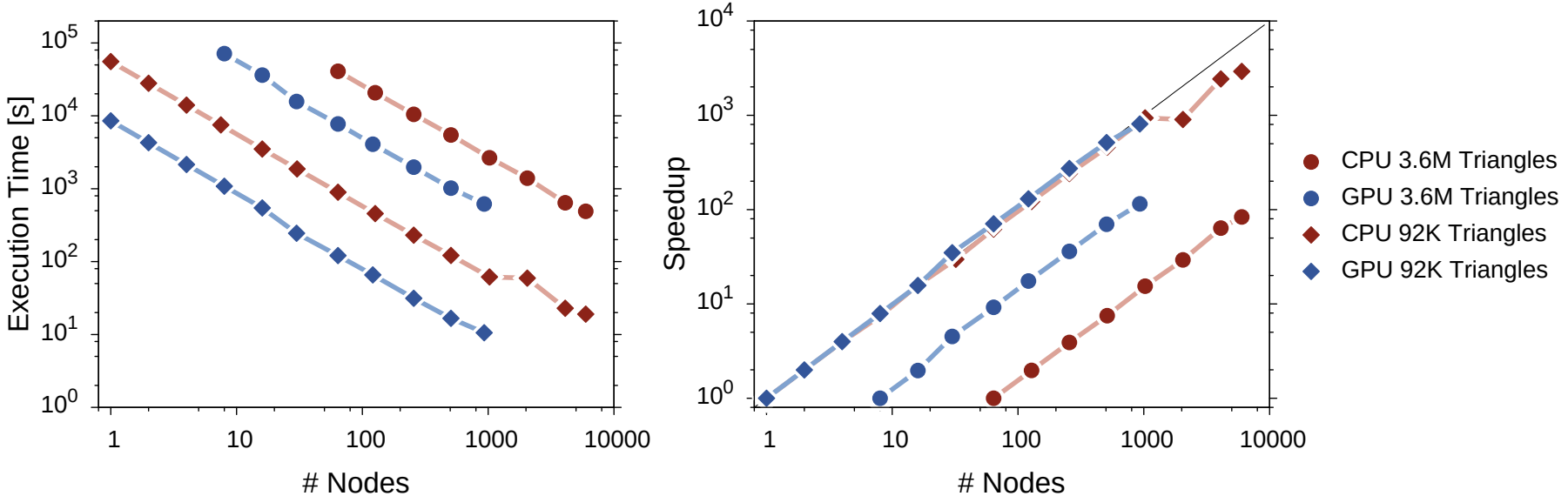
- **GPU Cluster:** *"TitanDev"*. Up to 930 nodes.
  - NVIDIA Tesla X2050 Fermi GPUs,
  - 6 GB device memory,
  - 1.15 GHz CUDA core clock,
  - AMD Opteron Interlagos 16 core CPU,
  - 32 GB main memory,
  - Gemini interconnects.
- **CPU Cluster:** *"Hopper"*. Cray XE6. Up to 6,000 nodes.
  - Dual AMD Opteron MagnyCours 12-core CPUs (total 24 cores),
  - 2.1 GHz clock,
  - 64 KB L1 and 512 KB L2 per core, 6 MB L3 shared by 6 cores,
  - 32 GB main memory,
  - Gemini interconnects.
- Single precision complex number computations.

Experimental Environments

- **GPU Cluster:** *"TitanDev"*. Up to 930 nodes.
  - NVIDIA Tesla X2050 Fermi GPUs,
  - 6 GB device memory,
  - 1.15 GHz CUDA core clock,
  - AMD Opteron Interlagos 16 core CPU,
  - 32 GB main memory,
  - Gemini interconnects.
- **CPU Cluster:** *"Hopper"*. Cray XE6. Up to 6,000 nodes.
  - Dual AMD Opteron MagnyCours 12-core CPUs (total 24 cores),
  - 2.1 GHz clock,
  - 64 KB L1 and 512 KB L2 per core, 6 MB L3 shared by 6 cores,
  - 32 GB main memory,
  - Gemini interconnects.
- Single precision complex number computations.

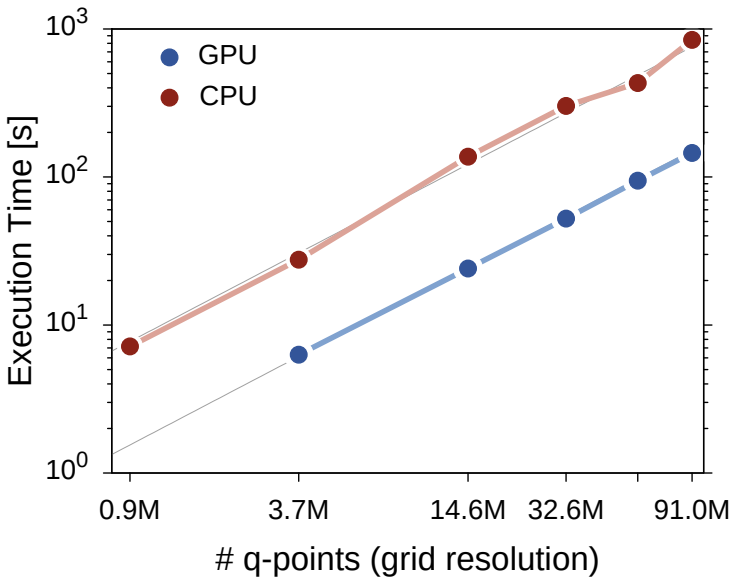
Scaling with Number of Nodes p: $O(\frac{nt}{\mathbf{p}})$

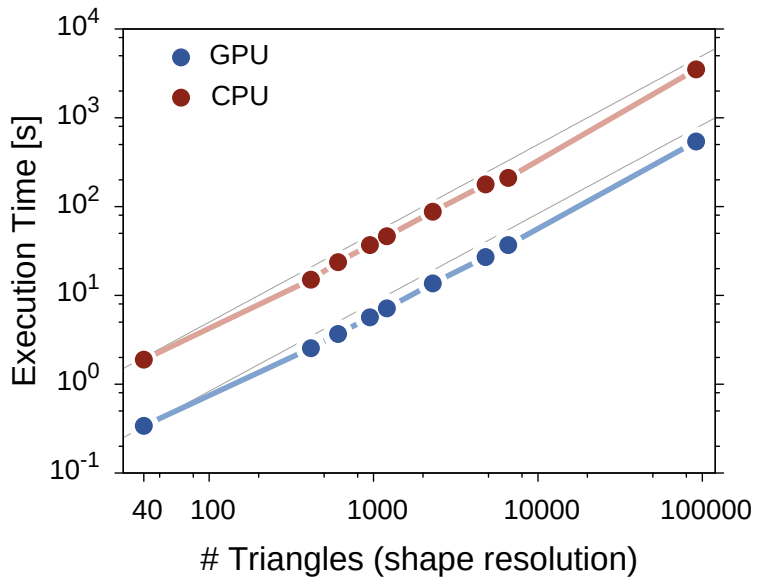
- GPU cluster = 1 to 930 nodes.
  - One MPI process per node. 16 OpenMP threads on host.
- CPU cluster = 1 to 6,000 nodes (24 to 144,000 cores).
  - Four MPI processes per node. 6 OpenMP threads per MPI process.
- Q-grid size = 91M q-points.
- Expected scaling = linear, observed = linear.



Scaling with Input Size n & t: $O(\frac{\mathbf{nt}}{p})$

- Q-grid resolution, n = 0.9M to 91M q-points (left).
- Shape resolution, t = 40 to 91K triangles (right).
- Number of nodes used = 4.
- Expected = linear, observed = linear.





Observations & Conclusions

Comparison	**GPU (930 Nodes)**	**CPU (6,000 Nodes)**
Single node speedup (wrt sequential code)	125×	20×
Performance ratio	1	6.25
Cluster speedup (relative to single node)	900× (96%)	5400× (90%)
Throughput (billion q-points per second)	999.98	941.07
Code base size ratio (LOC)	1.45	1
Development time person-hours ratio	4	1

Observations & Conclusions

- Implemented a high-performance GISAXS simulation code on GPU and multi-core clusters.
- Proper decompositions and optimizations are crucial for high performance.
- Brought down computational time from days and weeks to minutes and seconds.
- Allows simulation of much larger samples ($O(10^6)$ triangles) and with higher resolutions ($O(10^8)$ q-points) than previously feasible.

Future Work

- Further optimizations.
- Utilize new features of NVIDIA K20 GPUs.
- Scaling study on Titan.
- Implement more capabilities (e.g. sample slicing, analytical computations).
- Develop and implement fitting algorithms.

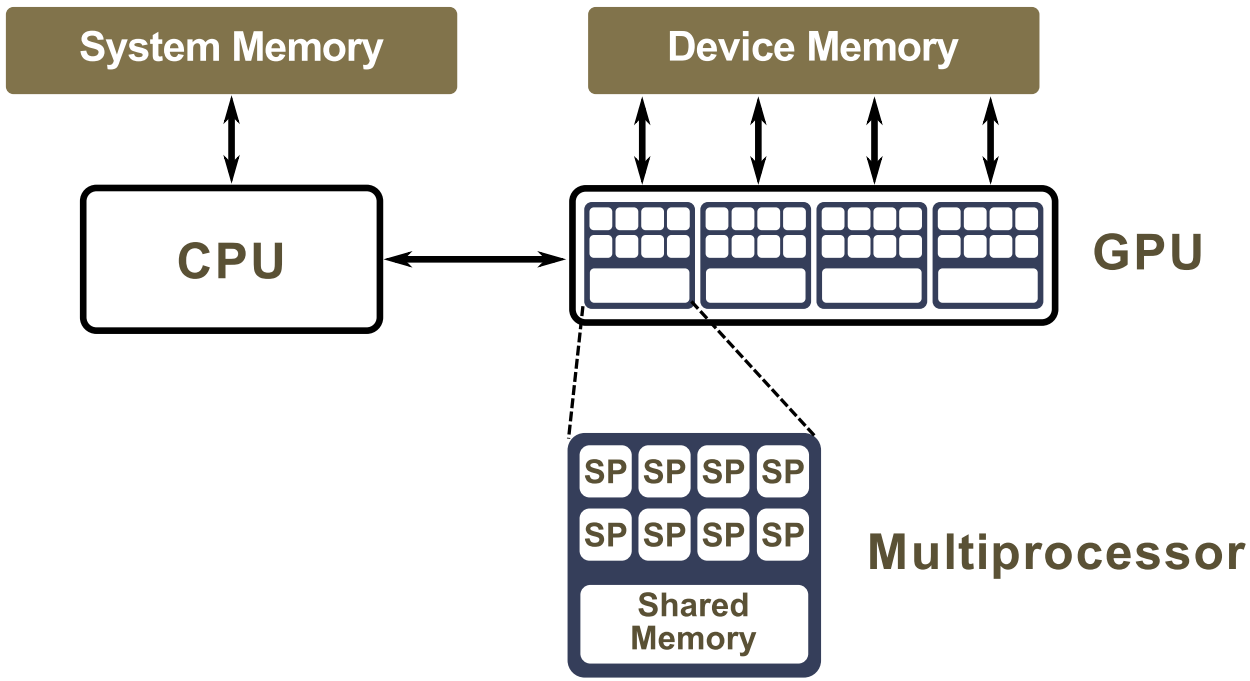
Acknowledgments

- Samuel Williams for his input on code analysis.
- This work is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
- Used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
- Used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

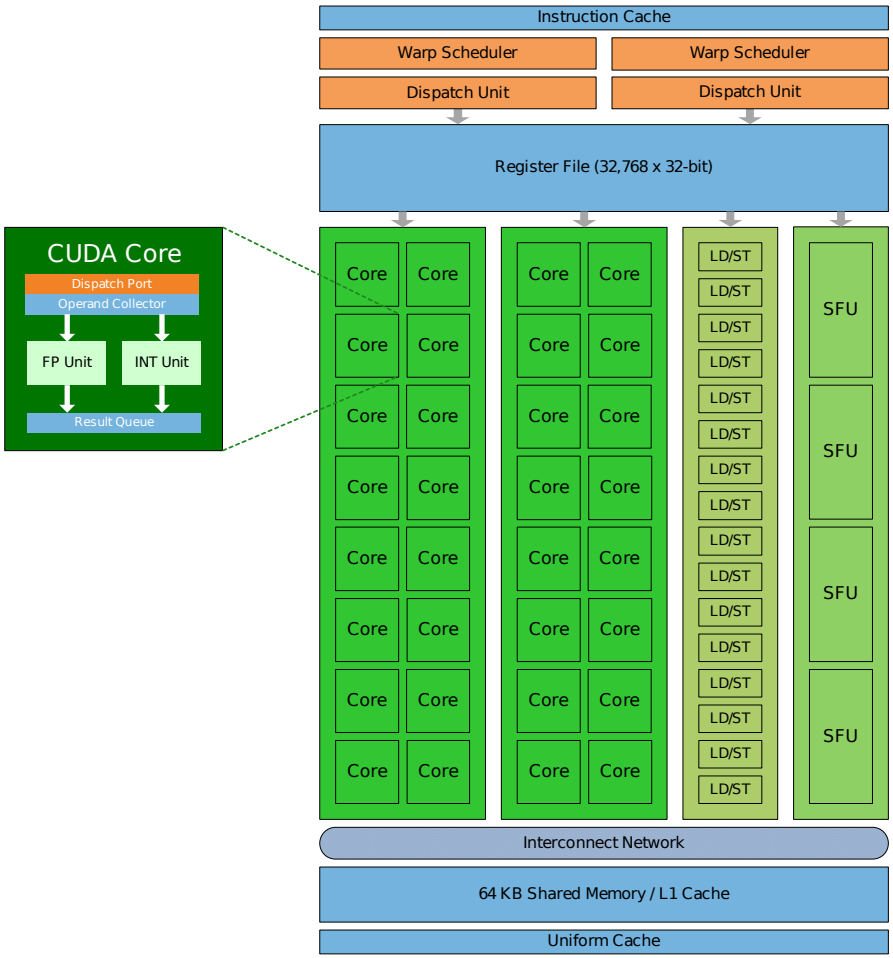
Thank you!

Graphics Processor

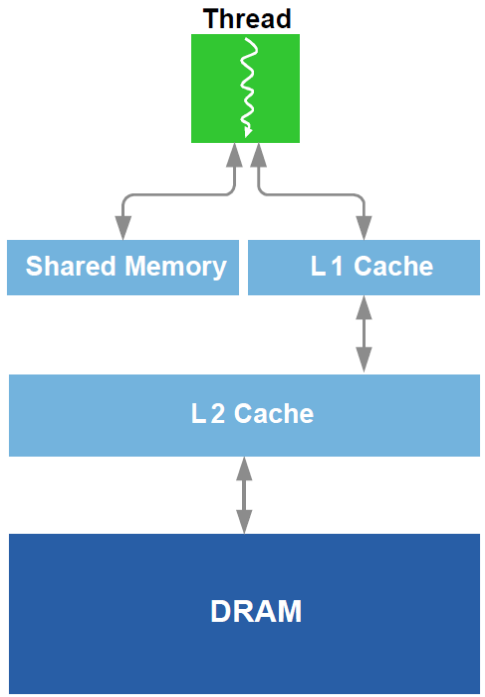
- Specialized processor; works in conjunction with a CPU.



Graphics Processor



Multiprocessor



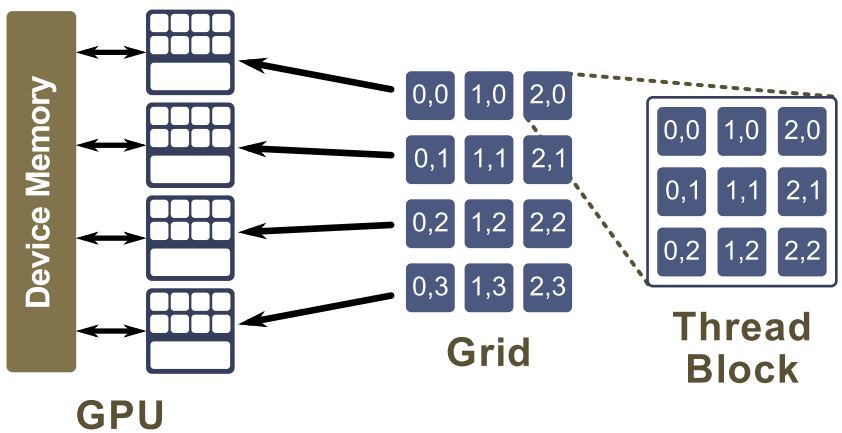
Memory Hierarchy

courtesy of "NVIDIA Fermi Compute Architecture Whitepaper."

NVIDIA CUDA and GPU Computing

- CUDA enables GPU computing.
- Heterogeneous serial-parallel, CPU-GPU programming model.
- Scalable – 100s of cores, 1000s of threads.
- Minimal extensions to C/C++ environment.

- Decomposed into a **Grid** of **Thread Blocks** containing **Threads**.
- Array of threads execute a kernel.
- Threads in a block can cooperate through on-chip shared memory.
- Threads in different blocks cannot cooperate with each other easily.
- Each thread block is scheduled to an SM.
- Thread blocks may execute in any order.

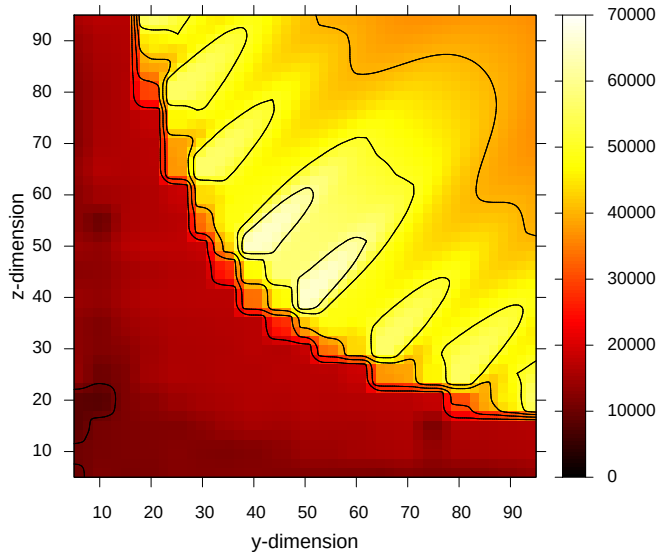


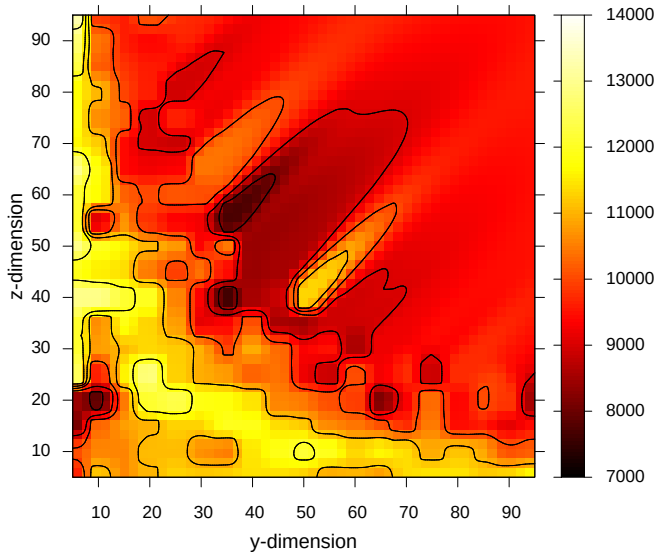
DWBA Theory

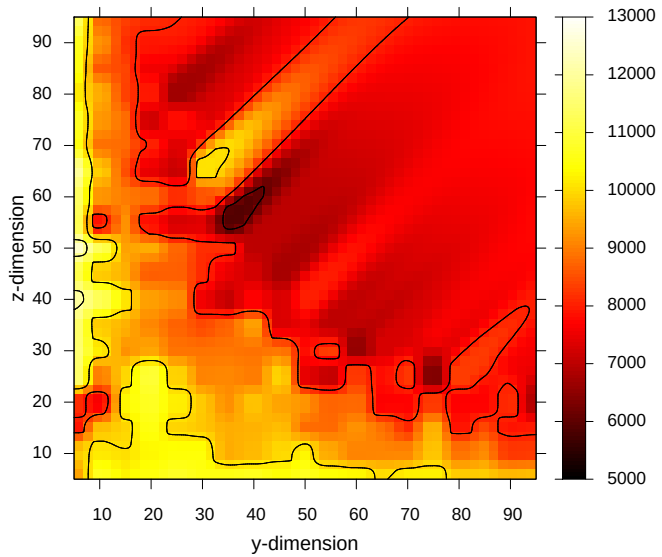
Code Optimizations: Examples

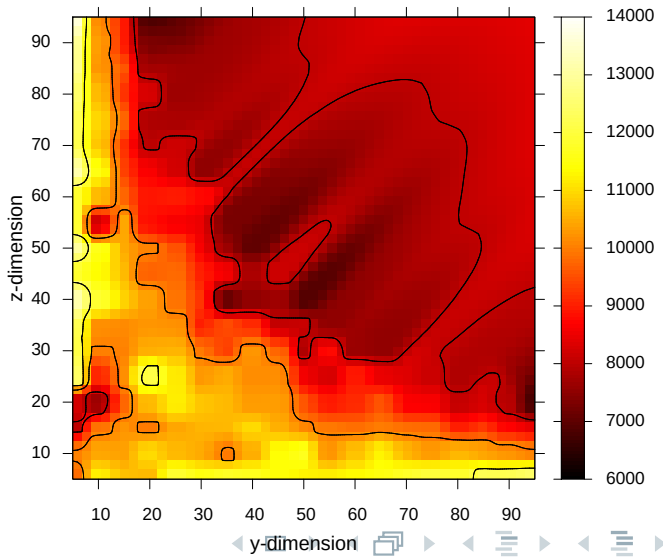
- Select optimal size for hyperblocks.
- Select optimal size for CUDA thread blocks.
- Memory optimizations:
  - Making full use of memory hierarchy; favoring shared memory (48K).
  - Using shared memory for input data reuse and output memory coalescing.
  - Padding and packing data to reduce number of memory transfers and ensure memory coalescing (6 to 3 per thread block).
  - Double buffering through multiple streams to hide device-host memory latencies.
  - Pinning host memory buffers.
  - Modifying memory access patterns to eliminate shared memory bank conflicts (24% to 0%).

Hyperblock Sizes for Various Kernel Implementations

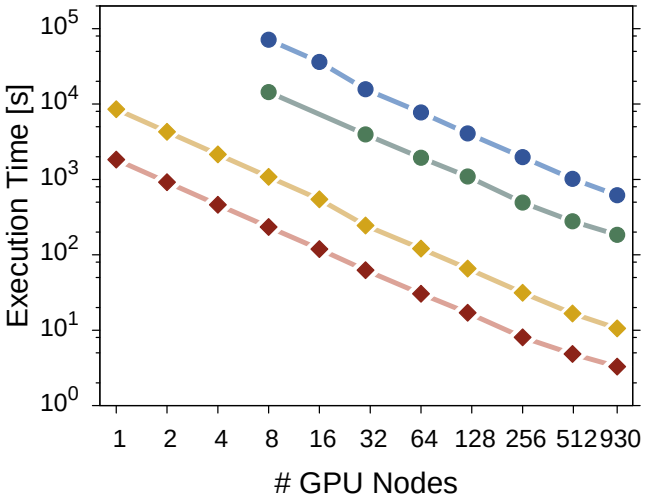


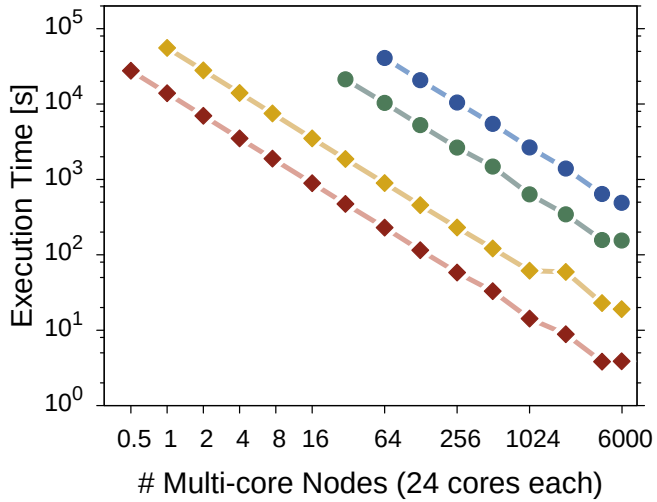


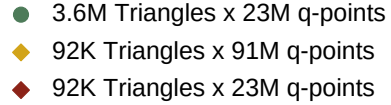


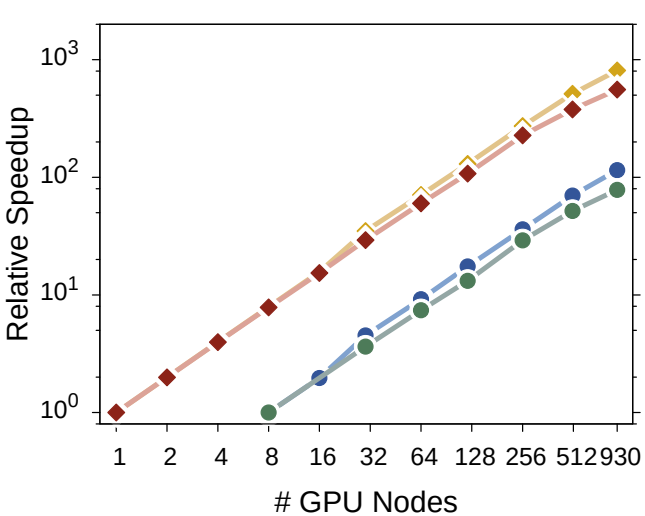


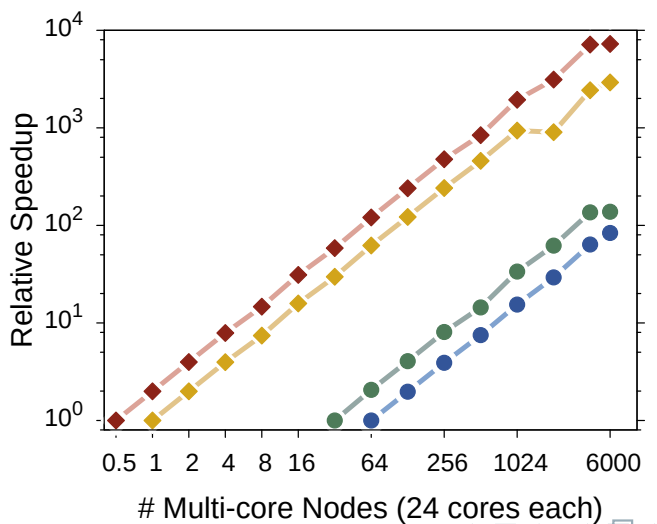
Scaling with Number of Nodes p: $O(\frac{nt}{\mathbf{p}})$

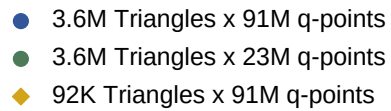




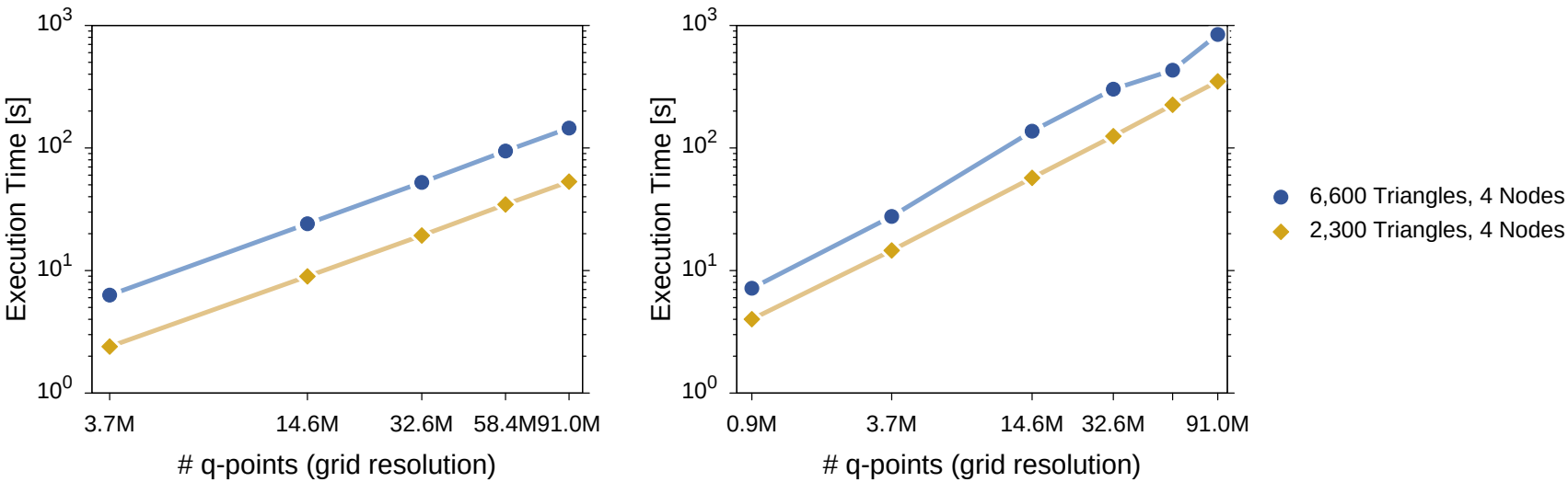








Scaling with Q-grid Resolution n: $O(\frac{\mathbf{n}t}{p})$



Scaling with Shape Resolution t: $O(\frac{nt}{p})$