Monte Carlo Neutron Transport on GPUs: Progress Report

Ryan Bergmann

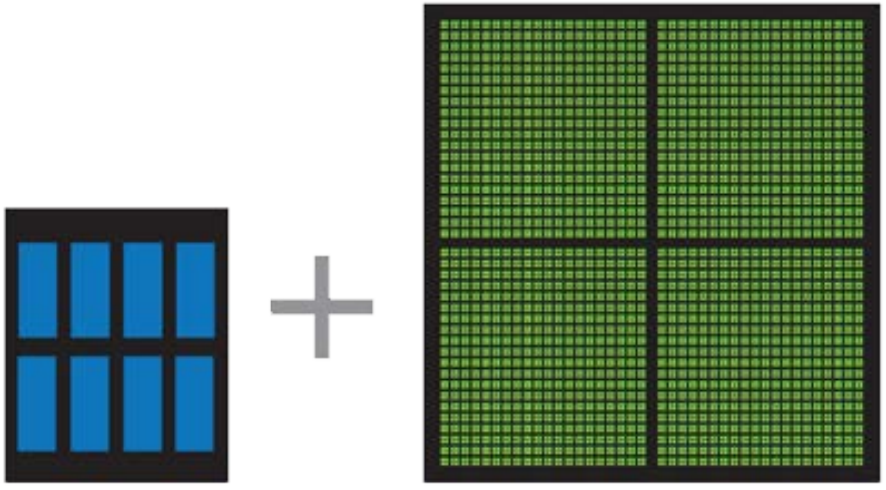
10/1/2013

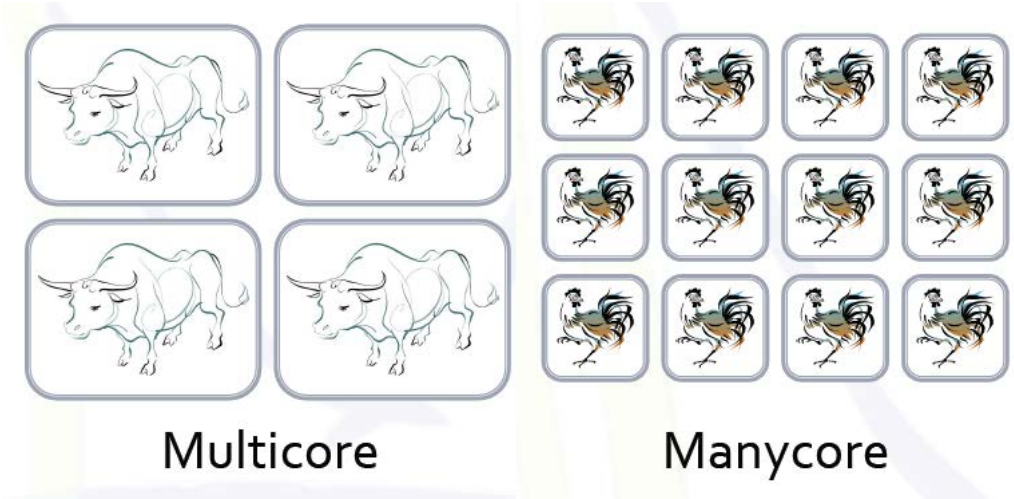
Outline

- What are GPUs?
- Why GPUs?
- Execution model
- Problems
- Solutions
- Preliminary work
- Ongoing work

What are GPUs?

What Are GPUs?

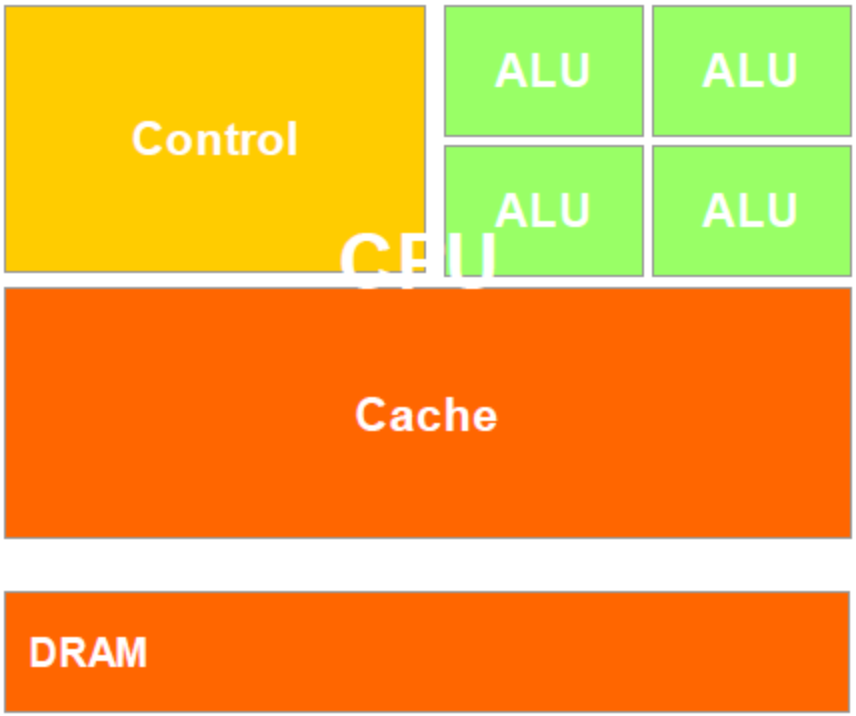


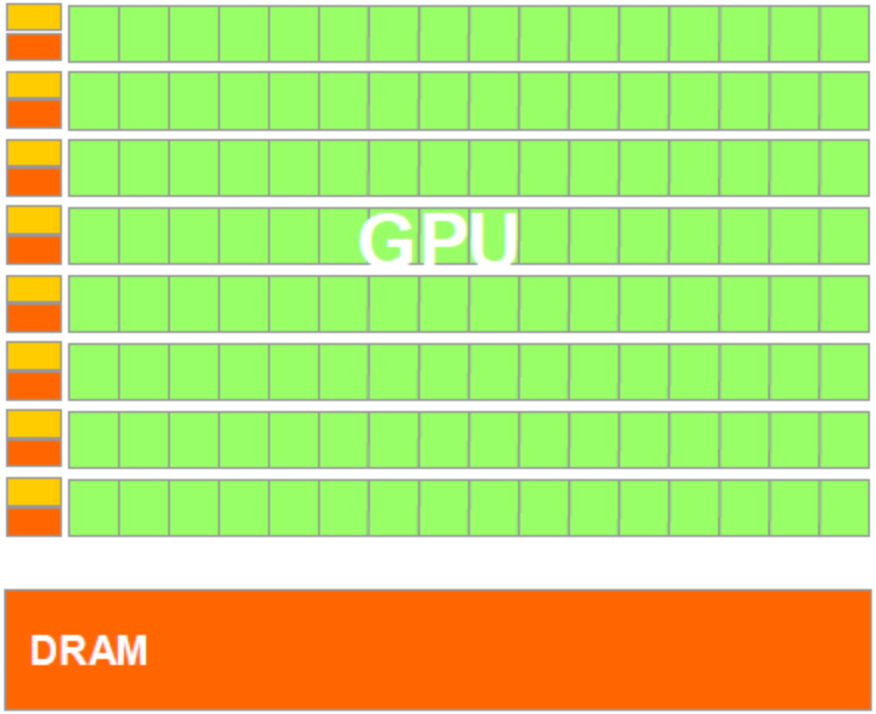


- In this case, really "GPGPU"
  - General Purpose Graphics Processing Unit
  - Not using them specifically for graphics
- "Massively parallel"
  - Geared for total throughput, not individual performance or complex tasks
  - Flock of chickens vs. yolk of oxen
  - Assembly line workers vs. master craftsman

Taken from Bryan Cantanzaro's CS-267 slides, Feb 2011.

GPU Transistor Usage





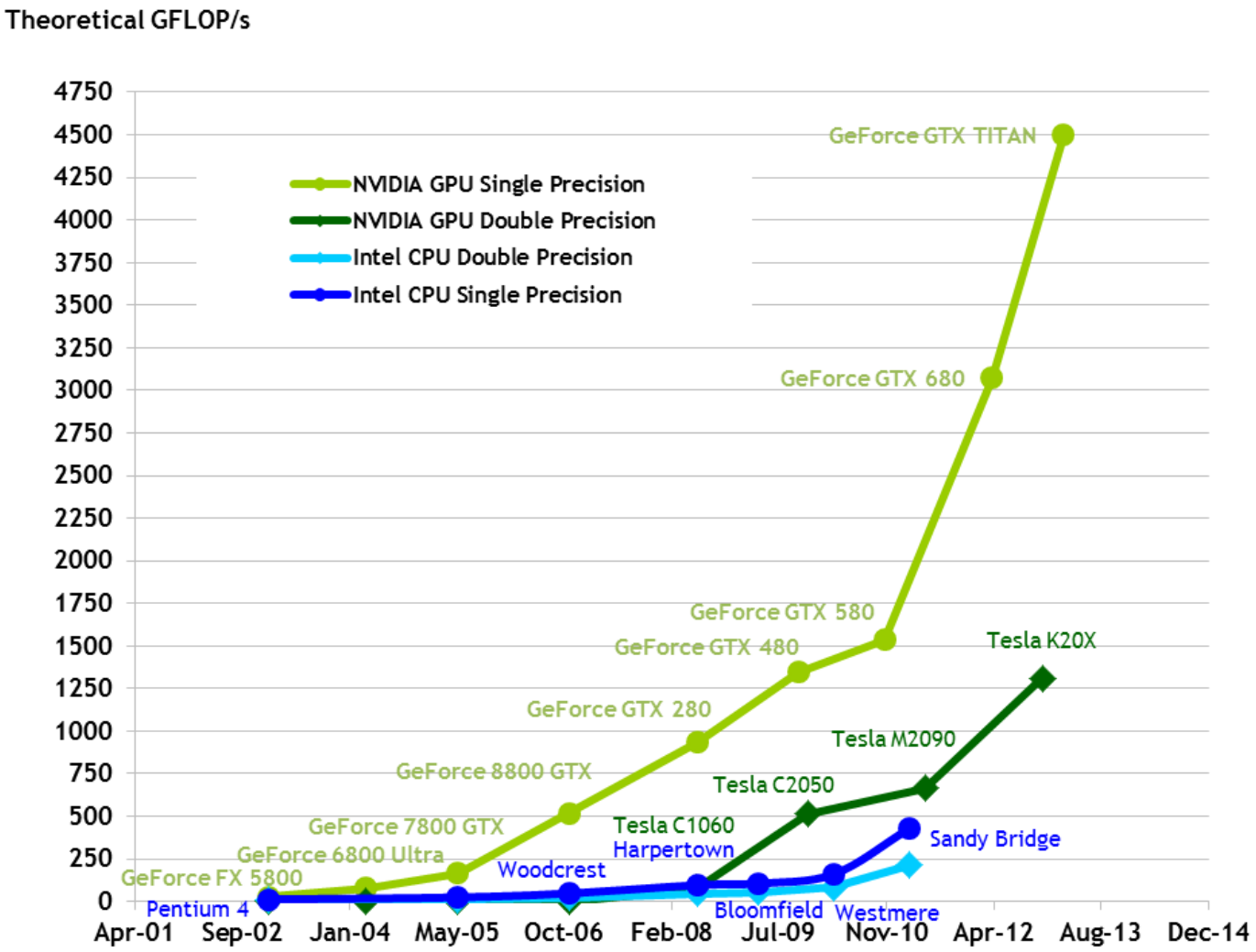
Graphic taken from Wen-mei Hwu's UCB CITRIS presentation, Jan 24-25, 2011.

Why GPUs?

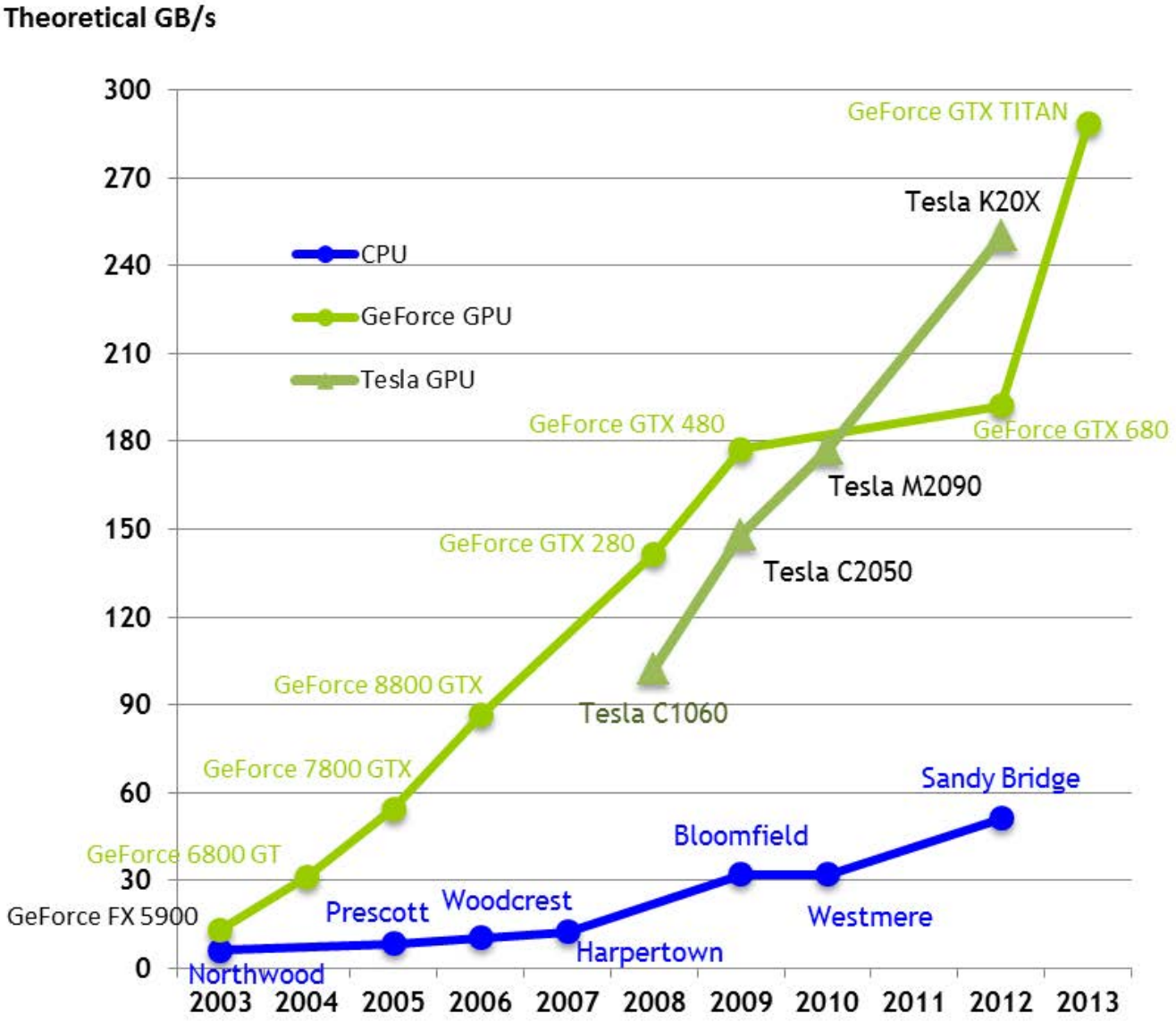
Why GPUs?

- Some high performance computing is going to GPU-heavy, shared memory systems
  - Top supercomputers use GPUs to gain efficiency
    - ~10x less energy per FLOP
  - GPUs can outperform CPUs in certain applications
  - If we want to target this new HPC hardware, codes need to be rewritten to do so
- GPUs are being used across all scientific fields
  - Major company endorsements guarantee future development

GPU/CPU - FLOPs



GPU/CPU - Bandwidth



CPU/GPU - Stats

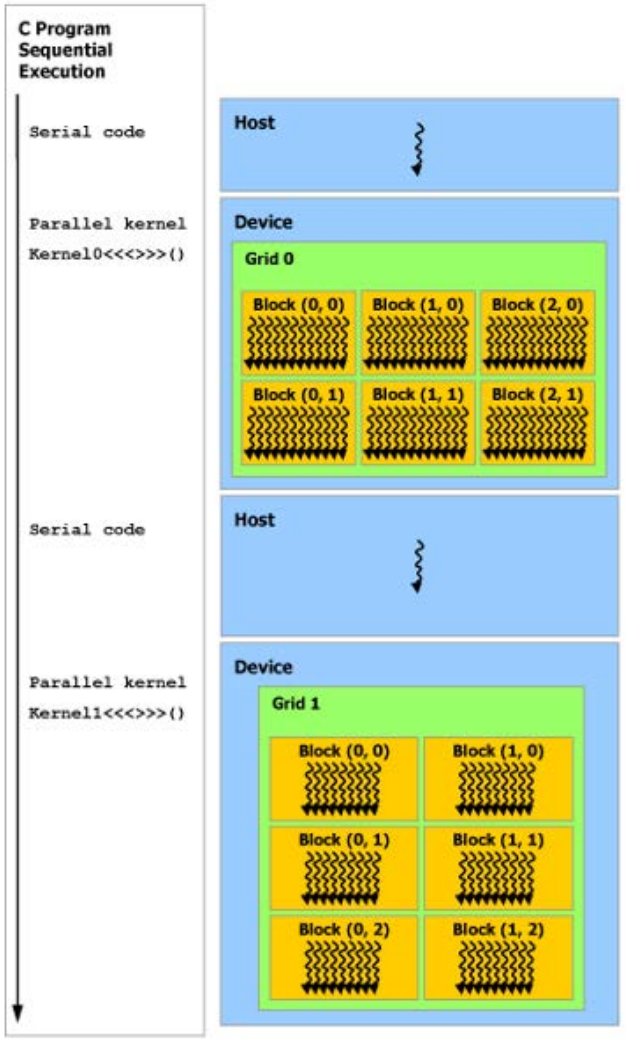
- Looks like GPU outstrips CPU
  - Higher bandwidth
  - Higher FLOPs
  - Lower cost
- Even more "starved for data"
  - Latency is very large
  - Bytes/FLOP is smaller
- Indicates that memory access could be a large problem

Feature	NVIDIA Tesla C2075	Intel Westmere EP (i7)
Physical Processors	14 cores, 2 issue, 16 way SIMD	6 cores, 2 issue, 4 way SIMD
Frequency	1.54 GHz	3.46 GHz
SP GFLOPS	1577	166
Global Bandwidth	192 GB/s	32 GB/s
Global Latency	200-800 clocks	~50 clocks
Bytes / FLOP	0.12	0.19
Register	2 MB	6 kB
L1 Cache	1024 kB	192 kB
Thermal Power	225W	130W
GFLOPS / Watt	7.01	1.28
Annual Cost / GFLOP ($.10/kW-hr)	$0.13	$0.69

Execution Model

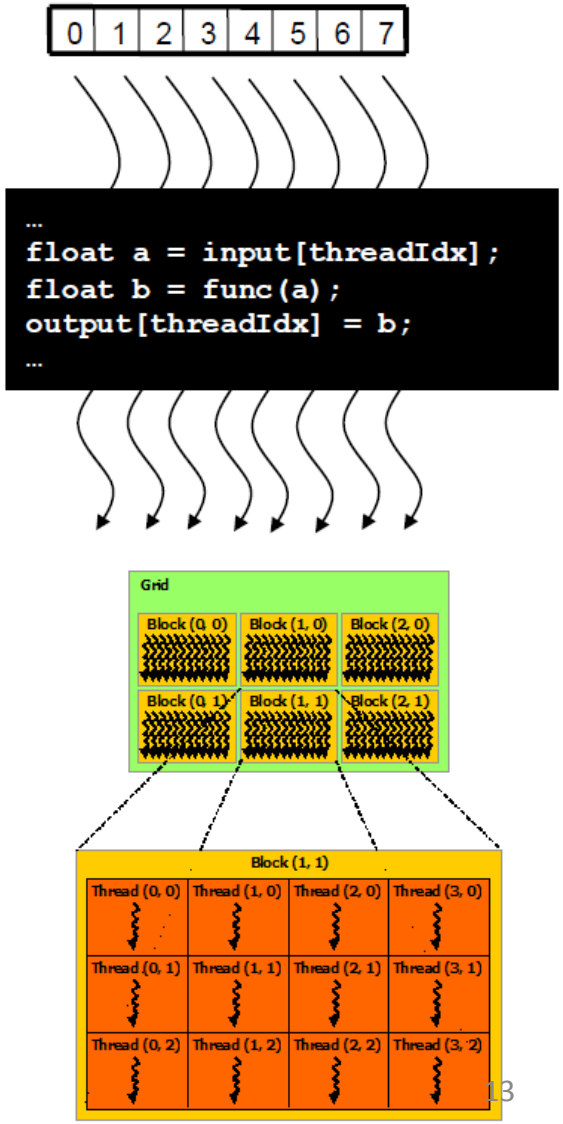
Execution Model (CUDA)

- SIMT = Single Instruction Multiple Thread
- "Data Parallelism" – the same operations are conducted on different pieces of data
- Takes advantage of regularity in instruction sets
- Easy to see how this would be advantageous in array operations
  - Matrix operations
  - Iterative methods



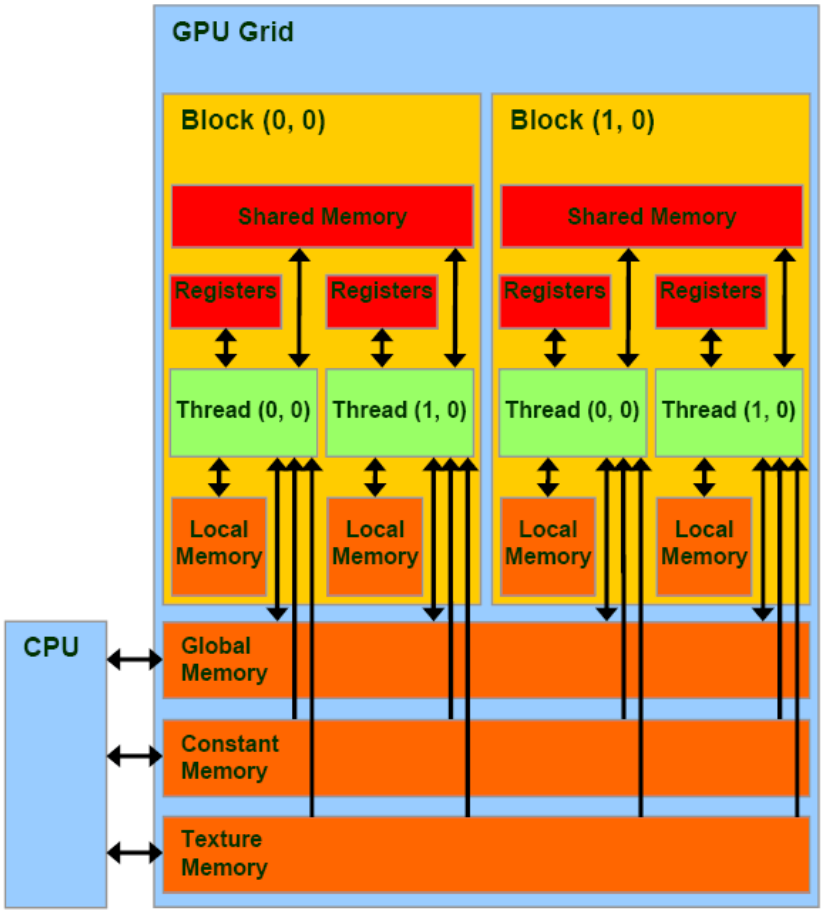
Threads

- Vector SIMD execution is abstracted by threads
  - Each thread in a “warp” executes the same instruction set on different data
  - Warps are 32 threads wide
  - Strict SIMT *not* enforced, divergence causes serialization
- Thread block execution is scheduled by hardware
  - Each thread must be completely independent of any others
  - Threads must be allowed to execute in any order
- Each thread and thread block has a unique ID which can be used to access different pieces of data.
  - tid = ThreadID + BlockID * BlockDim
  - data[tid]



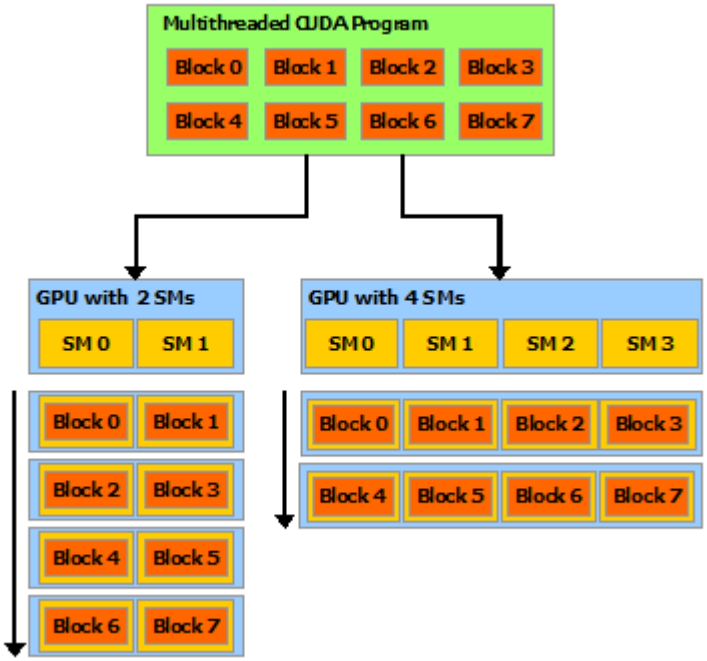
Memory

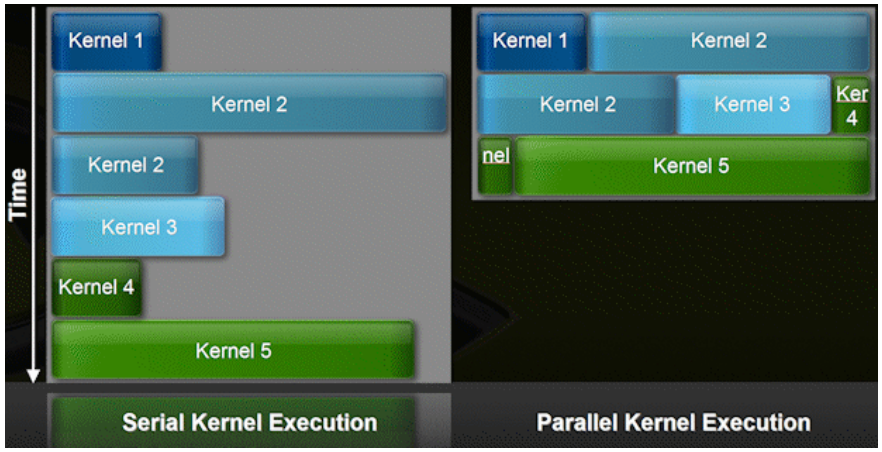
- Very stratified
  - Registers
  - Shared
  - local cache
  - Global
  - Constant
  - Texture
- All with different latencies, sizes
  - Much management needed to get efficiency
- Keep constant things in constant or texture memory
  - Cross sections!



Block Scheduling

- Streaming Multiprocessor (SM) execution abstracted by thread blocks
- Basic “unit of work” for a SM
- Thread blocks assigned to a SM at runtime, occupied a SM until complete
- Allows for concurrent kernels, mapping problems to different hardware



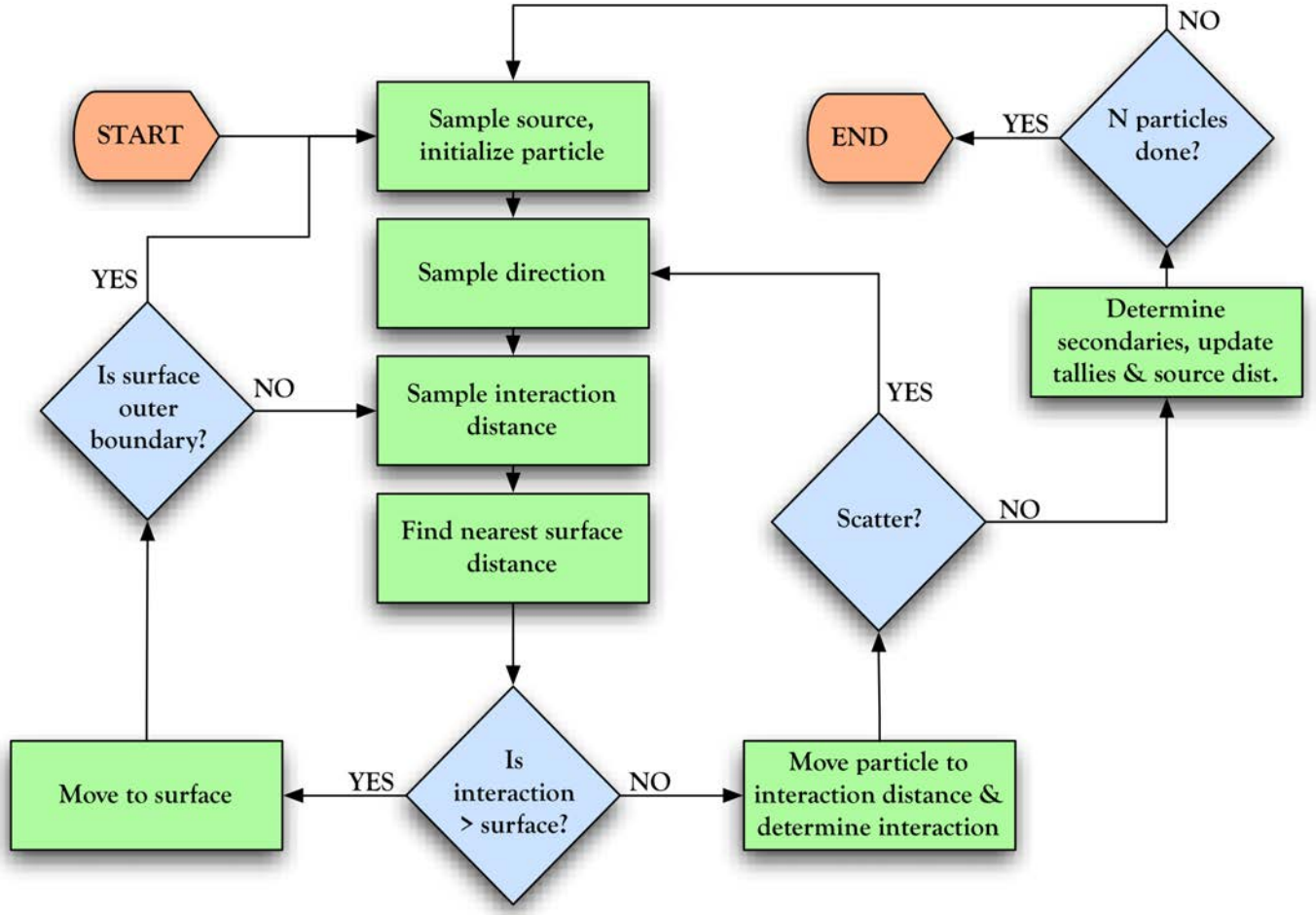


Problems

Problems

- Limited memory
  - 6GB max on or Tesla cards
- Keeping processors fed with data
  - Higher flops/byte than CPUs
  - CPUs already “data starved”
- Latency
  - Accessing global memory has a huge latency cost
  - Cannot randomly access data unless accepting this penalty is OK
- Reliance on SIMT
  - Divergent problems serialize warp execution
  - MC uses “if” statements based on random numbers

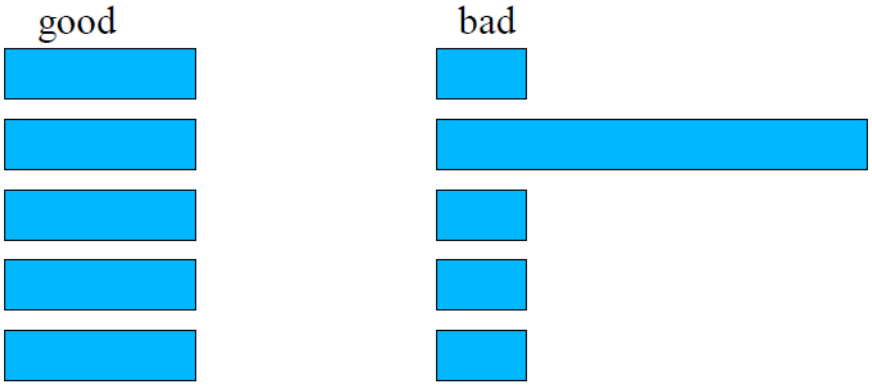
Task-Parallel MC



- What most CPU codes do
- "A particle per thread"
- Tasks are completely independent
  - Tasks can be at different states at any point in time
  - "task parallel"

Monte Carlo and SIMT

- MC method involves lots of 'if' statements based on random numbers
  - Creates many areas where thread control flow can diverge
  - Creates irregular memory access patterns
- Highly divergent problems can cause severe under utilization of GPU resources
  - Serialization of divergent threads
  - Warps to remain idle while waiting for longest thread to complete
  - Very random, non-coalesced memory access has large latency penalties



Graphic taken from Wen-mei Hwu's UCB CITRIS presentation, Jan 24-25, 2011.

Solutions 1

- Data-parallel algorithm
  - Increase memory access regularity as much as possible
  - Keep thread states close to keep warps from serializing
- Use NVIDIA OptiX to handle ray-tracing
  - Acceleration structures and I/O built-in
  - Can do “where am I,” surface detection, and material hash
- Unionized energy grids
  - Only a single search needed to find reaction type
  - Introduced gaps in data are interpolated
  - Higher memory footprint from redundant data
- Running large datasets
  - Hide latency in PCI communication with host
  - Limited by DRAM

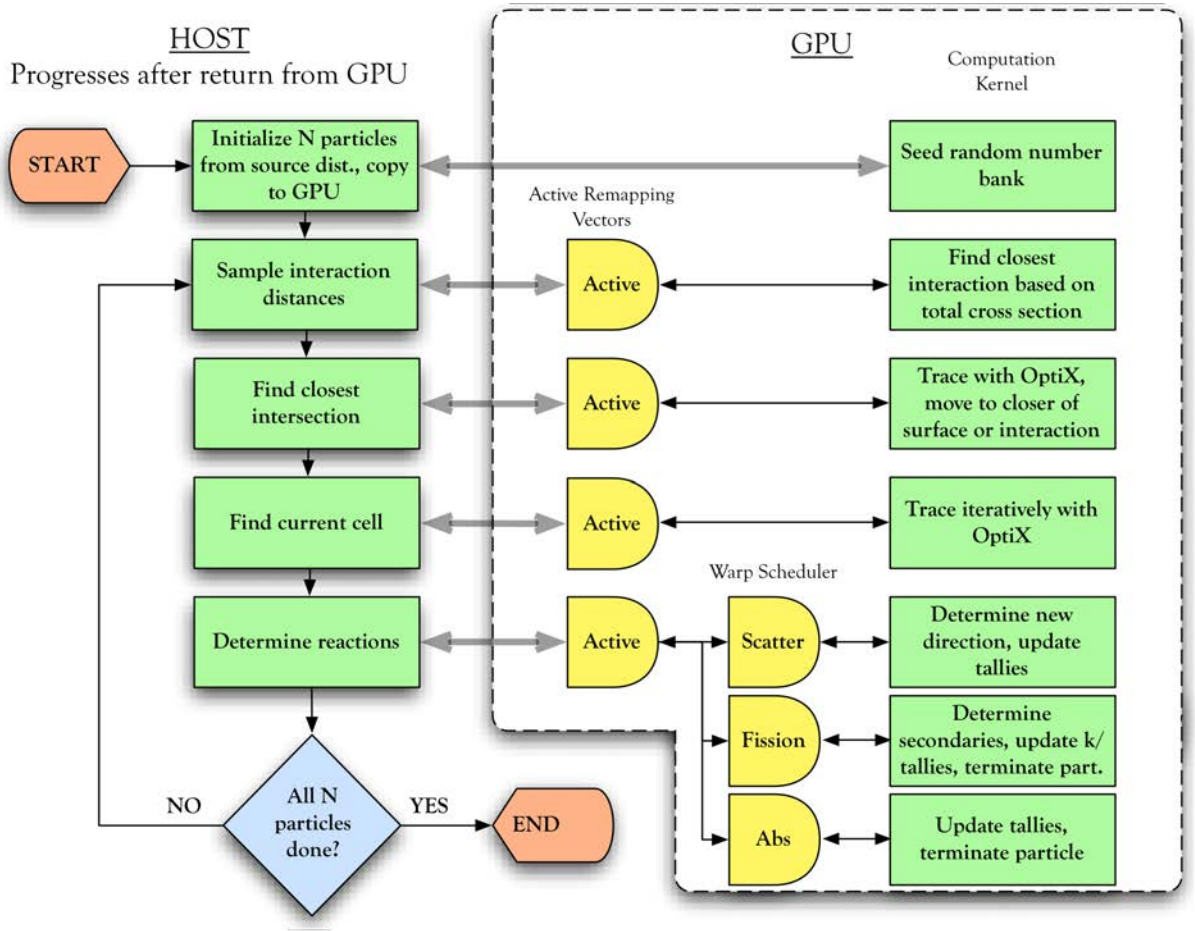
Solutions 2

- On the fly cross section processing
  - No need for multiple copies of data
  - Have to process at every interaction – hopefully hidden by latency (i.e. work would be done waiting for data to load anyway)
- Sorting by energy
  - CUDPP
  - Keep data loads as coalesced as possible
  - Cost of remapping references or of actually moving the data
- Overloading each multiprocessor
  - Give the warp scheduler lots of threads per block so it can keep warps coherent

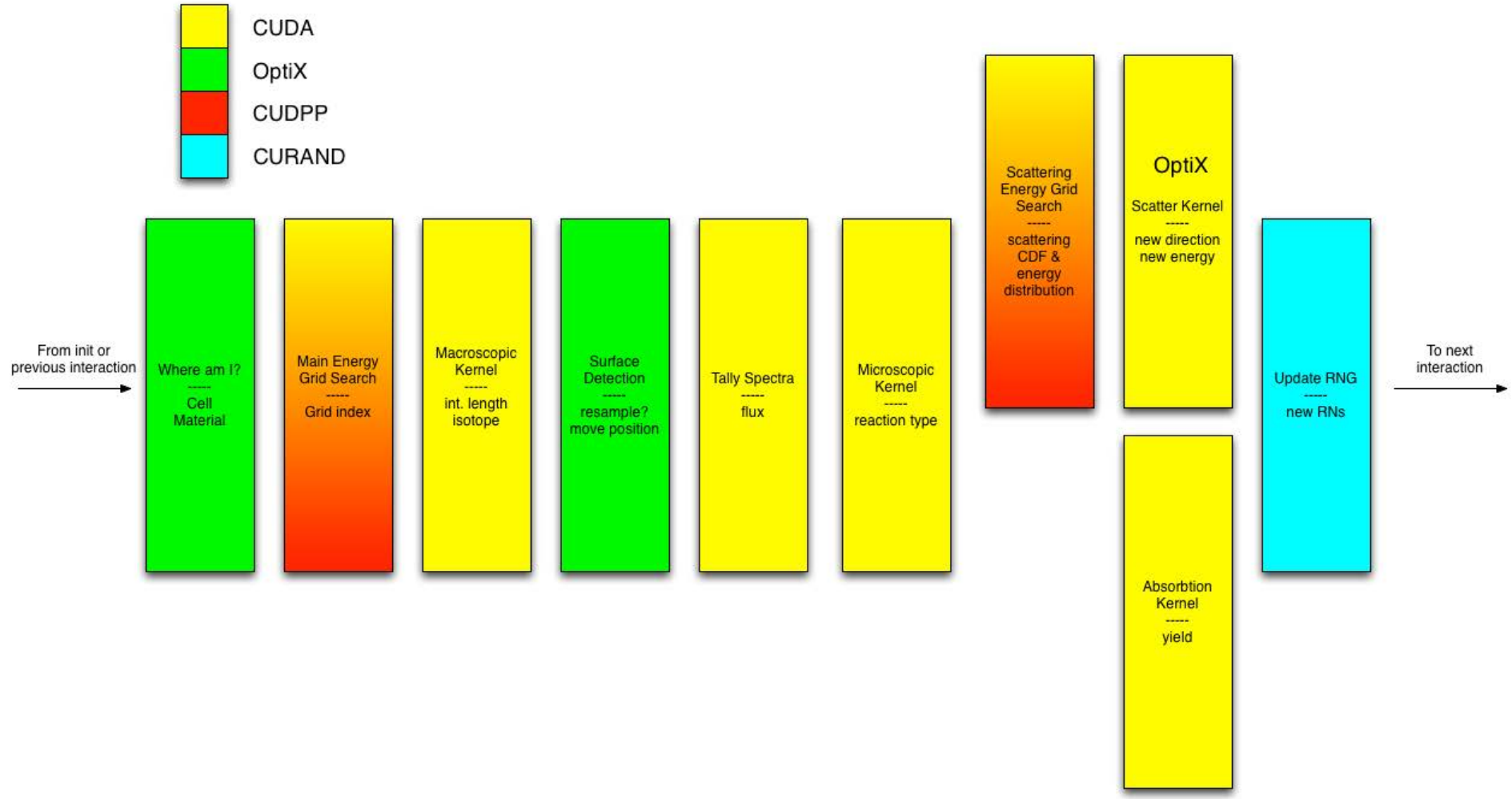
Data Parallel Algorithm

Data Parallel Algorithm

- Transporting an entire set of particles in lockstep
- Changes “a particle per thread” to “N particles shared by M threads”
- Data parallel in the sense that particles are undergoing similar reactions
  - Tasks are the same
  - Data acted upon is different
  - Also called “event-based”, historically



Detailed Data-Parallel Transport Pipeline



Ray Tracing with OptiX

Ray-Surface Intersections

Ray origin and direction:

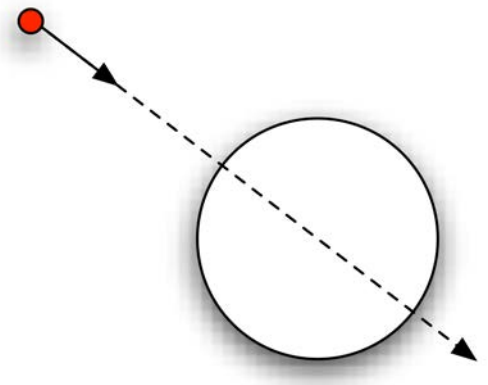
$$\vec{x}_o = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \quad \vec{x}_d = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

Parametric ray equation:

$$x(t) = x_o + tx_d$$
$$y(t) = y_o + ty_d$$
$$z(t) = z_o + tz_d$$

Sphere:

$$r^2 = x^2 + y^2 + z^2$$



Solve for t:

$$a = \vec{x}_d \cdot \vec{x}_d$$
$$b = 2\vec{x}_o \cdot \vec{x}_d$$
$$c = \vec{x}_o \cdot \vec{x}_o - r^2$$

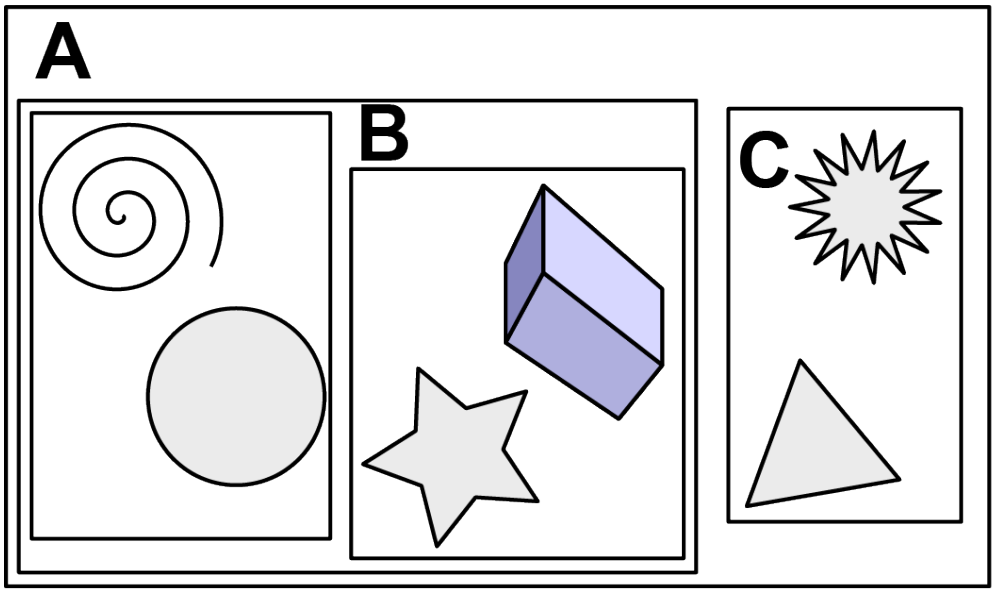
$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

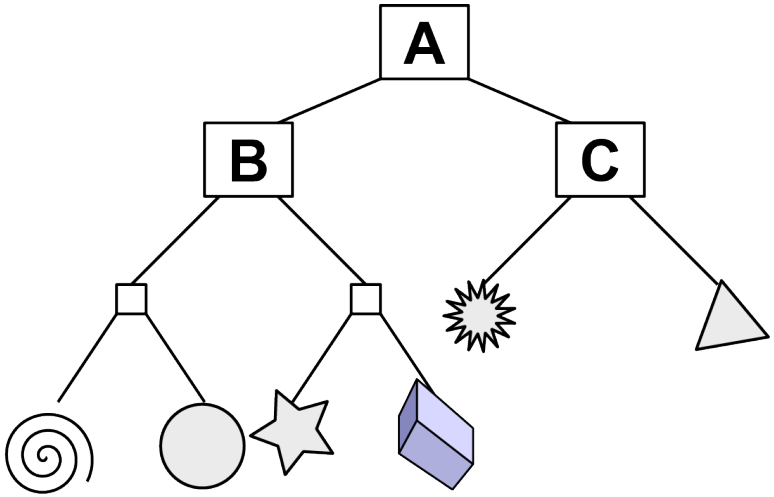
OptiX tightens an interval on t until it finds the closest intersection

OptiX Ray Tracing

- Ray Tracing API
  - Framework for building applications, not a premade library
  - Have to write all intersection programs, data structures
- NVIDIA knows what they're doing
  - I'm not a computer scientist (despite appearances sometimes)
  - I can't program it better than they already did

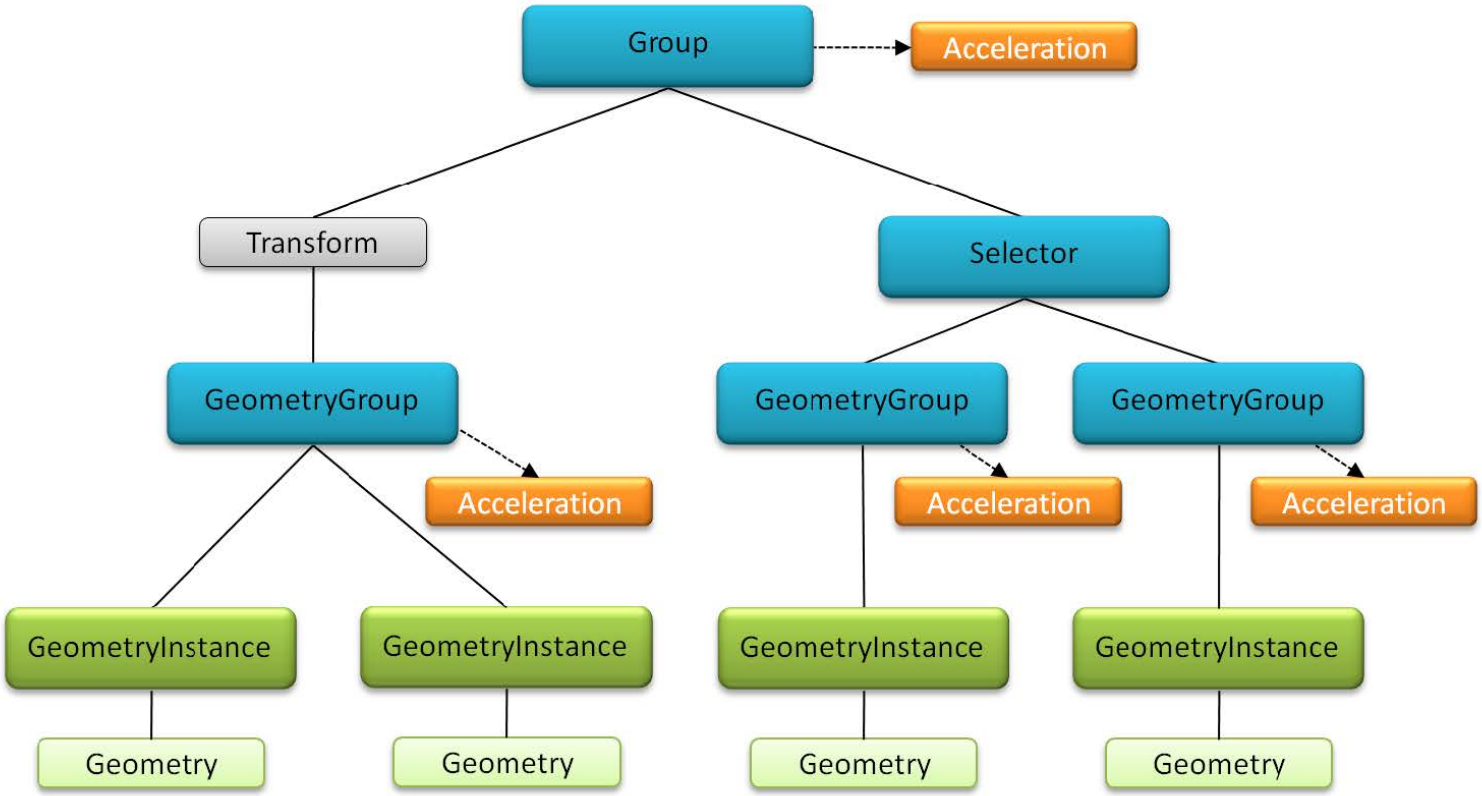
OptiX Acceleration Structures





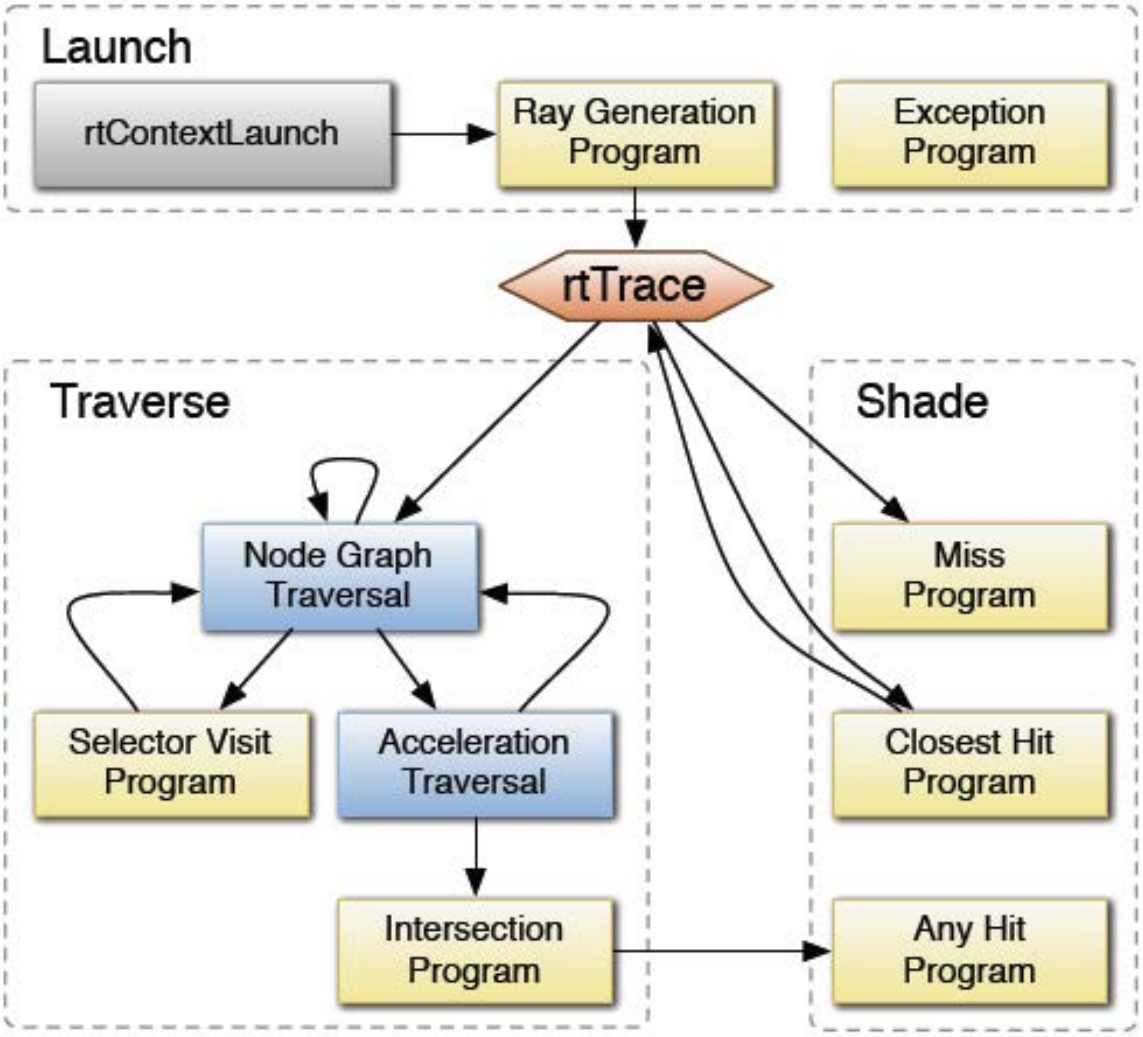
- Makes available many different qualities of Bounding Volume Hierarchy (BVH)
  - Object centric
  - Spatial redundancy
- k-d/binary tree
  - Spatial centric
  - Object redundancy
- Automatically built on launch
- For static calculations, would be a small initial cost for large improvement

OptiX Geometry Node Graph



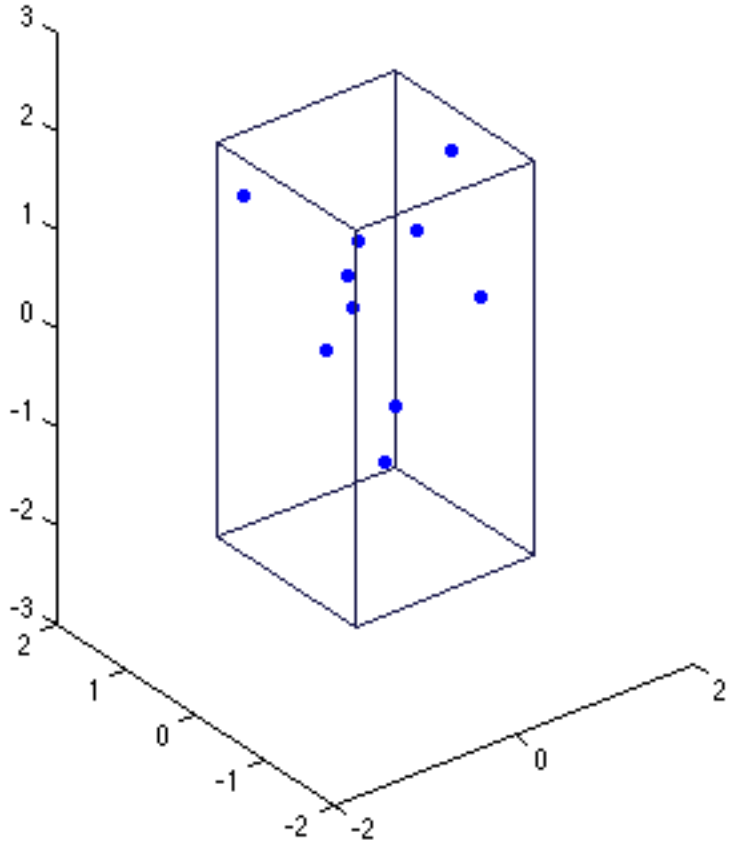
- Transform nodes used for instancing primitives
  - Arrays, etc. Like the "LIKE" card in MCNP

OptiX Execution Model



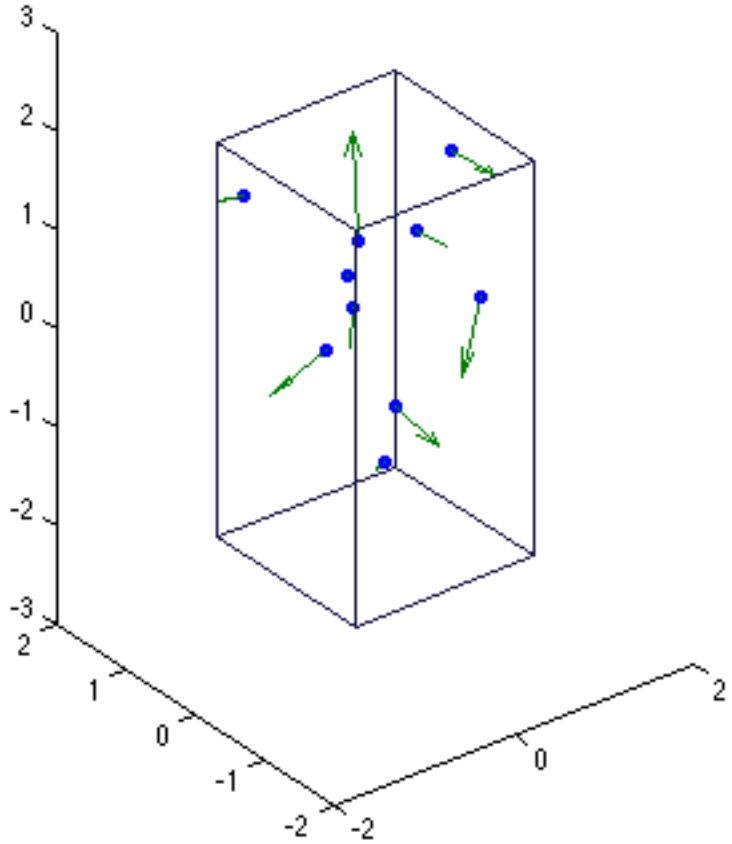
Example: Batched Ray Tracing Step

Source particle positions



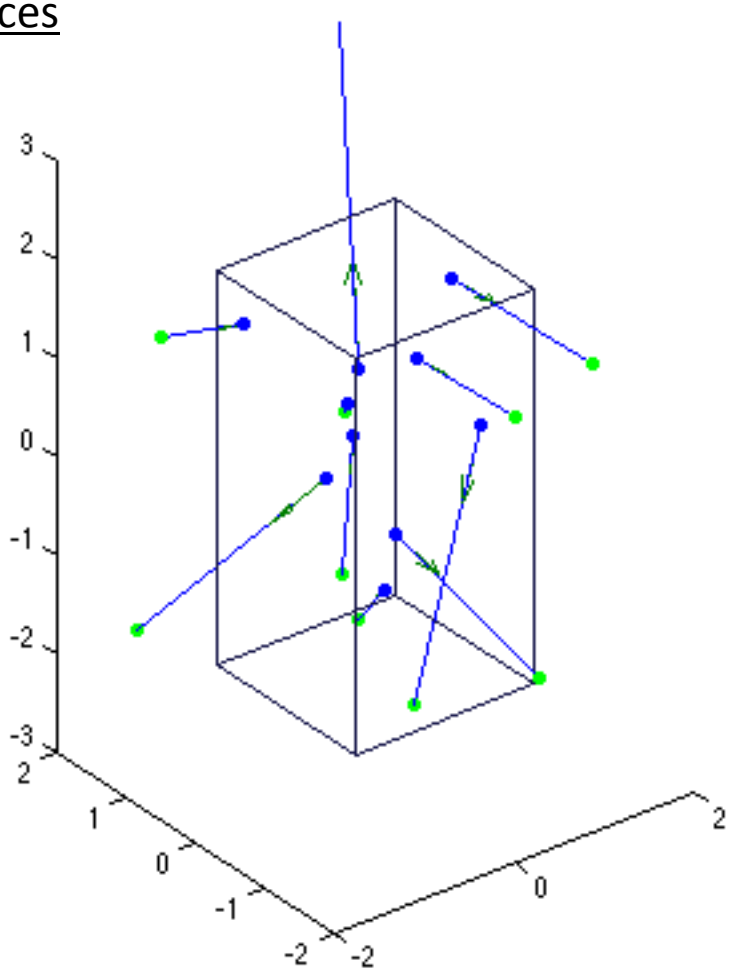
Example: Batched Ray Tracing Step

Source particle directions



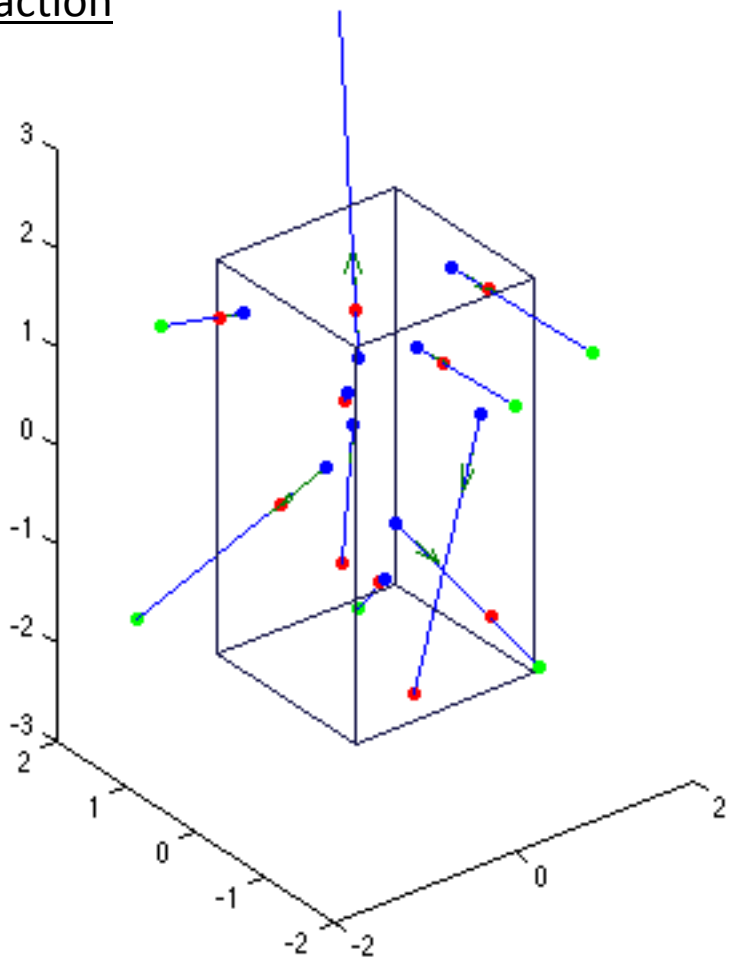
Example: Batched Ray Tracing Step

Sampled interaction distances



Example: Batched Ray Tracing Step

Move to boundary or interaction point, whatever is closer



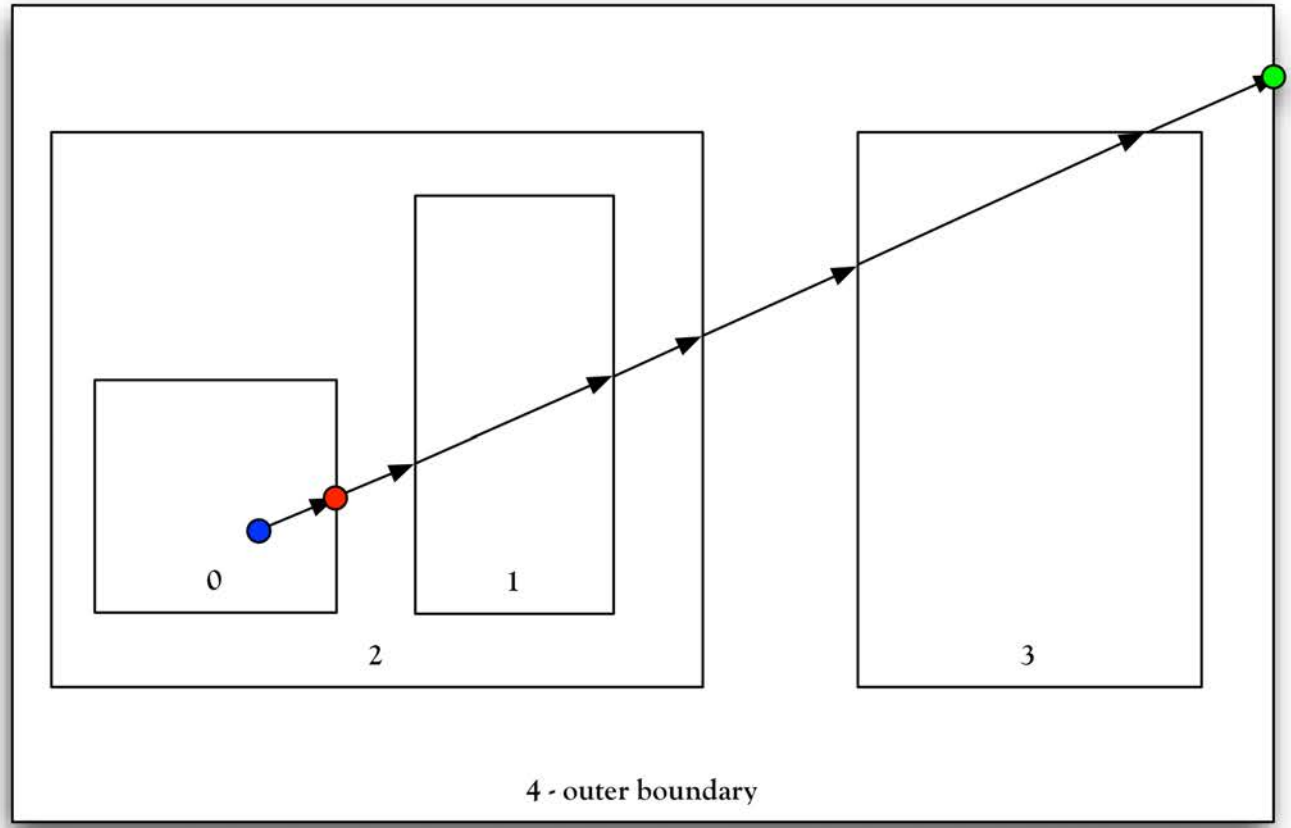
“Where Am I?” Algorithm

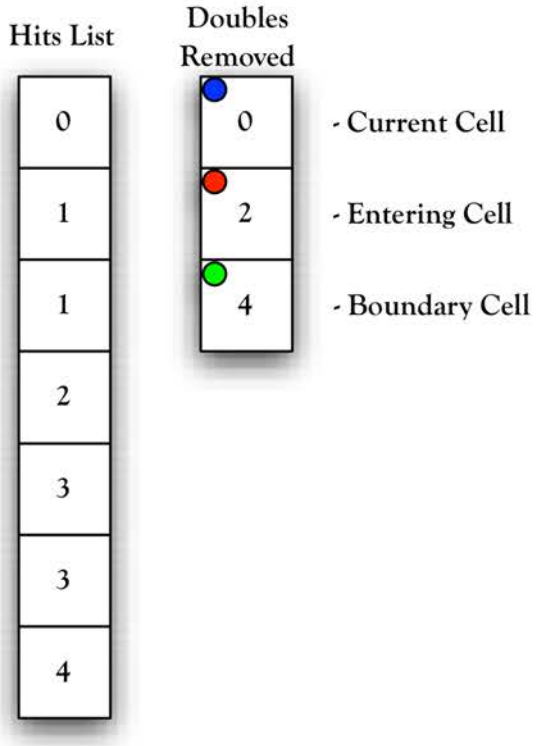
- As neutrons randomly walk across the spatial domain, they encounter boundaries
- Material properties must be updated at these boundaries (cross sections, temps, etc.)
- The cell that the neutron enters must be identified in order to look these properties up
- Neutron only carries current state data, routine must update properties

“Where Am I?” Algorithm

- Can be done inside ray tracing routine instead of separate routine that evaluates binary logic from all 2nd-order surfaces
- Needed since information only updated at surface intersection events

- **REQUIREMENTS**
  - All cells must be closed
  - All cells must be unique
  - No cells can overlap (result will be order-dependent unless set to same cell ID)



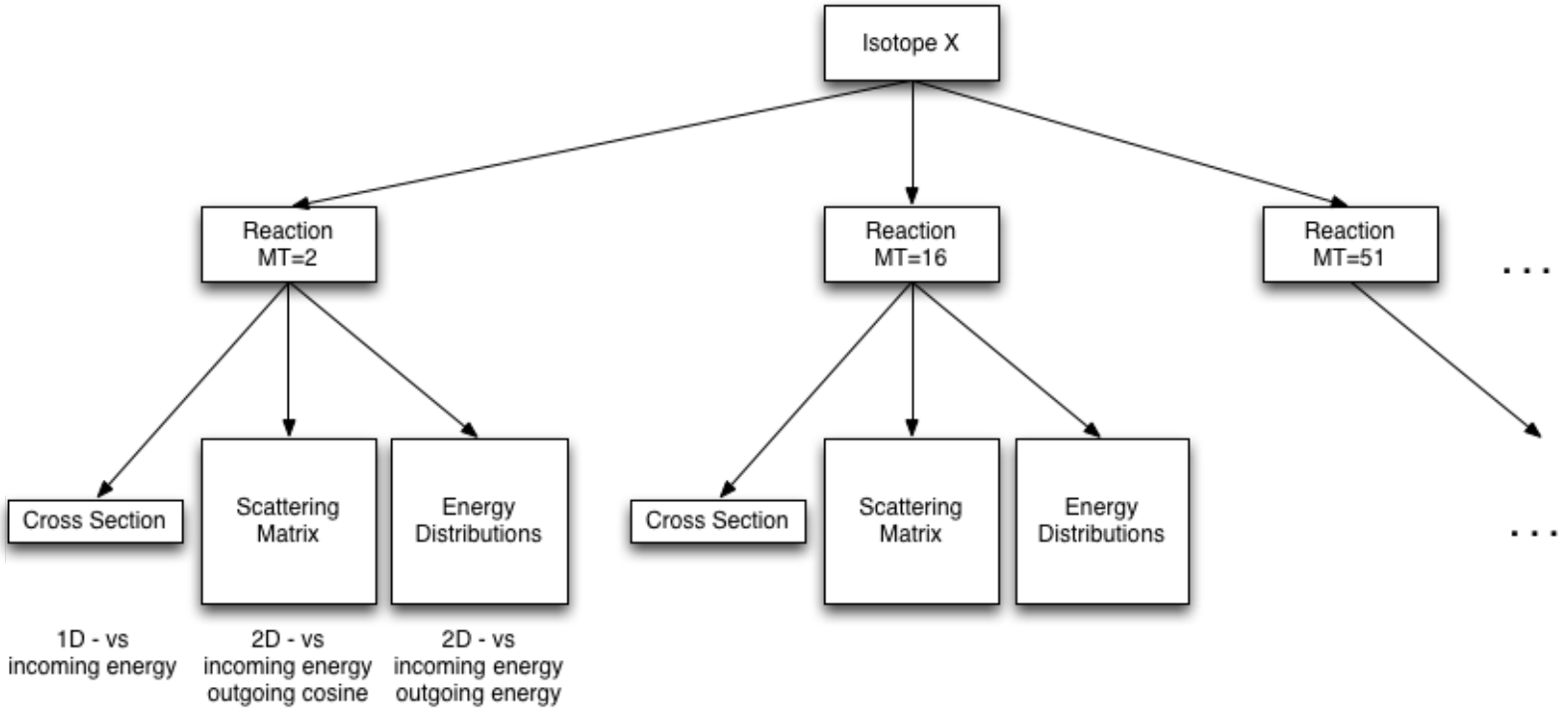


“Where Am I?” Algorithm

- Strengths
  - Done inside tracing routine
  - Can also do material hash
- Weaknesses
  - Scaling. Total will be query time (log(N)) multiplied by:
    - N in worst case (all objects on a line)
    - 1 in best case (only one object in line)
    - $\sqrt[3]{N}$ if spherically distributed

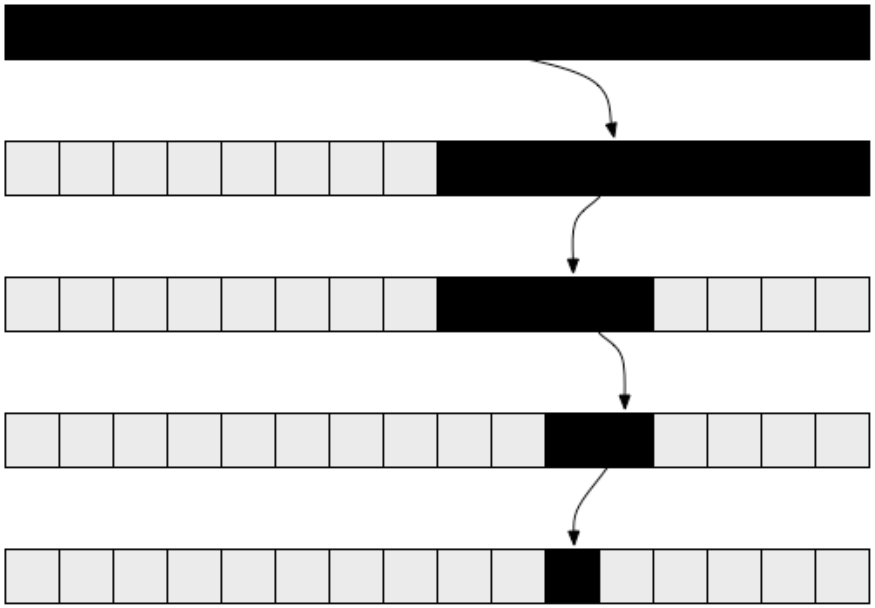
Unionizing Nuclear Data

ENDF Data Hierarchy



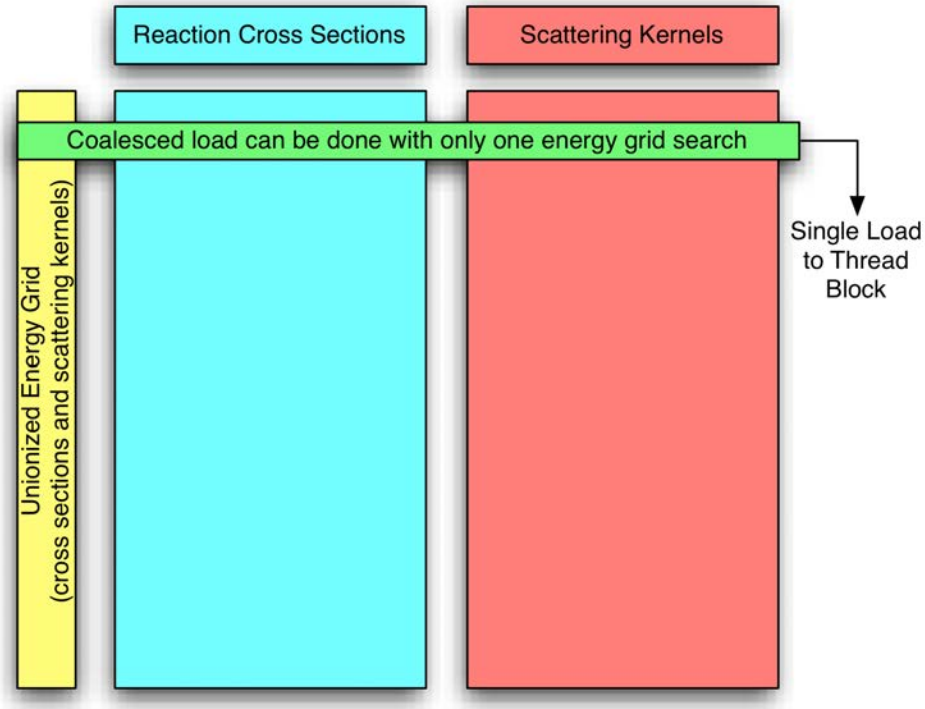
Grid Search

- Binary search
  - O(log(N))
  - No storage
  - Lots of global access
  - Good enough?
- Simple Lookup Vector
  - O(1)
  - Linearize so every bin is hit at least once by a line
  - dx = smallest difference in grid
  - 20MeV/dx ~ 10^{14} ☹☹☹
- Very high order polynomials?
  - Accuracy?
  - Time?
- Use line or polynomial as initial guess for binary search to decrease loads?
- Is there a way to efficiently hash a range of floats to an integer?



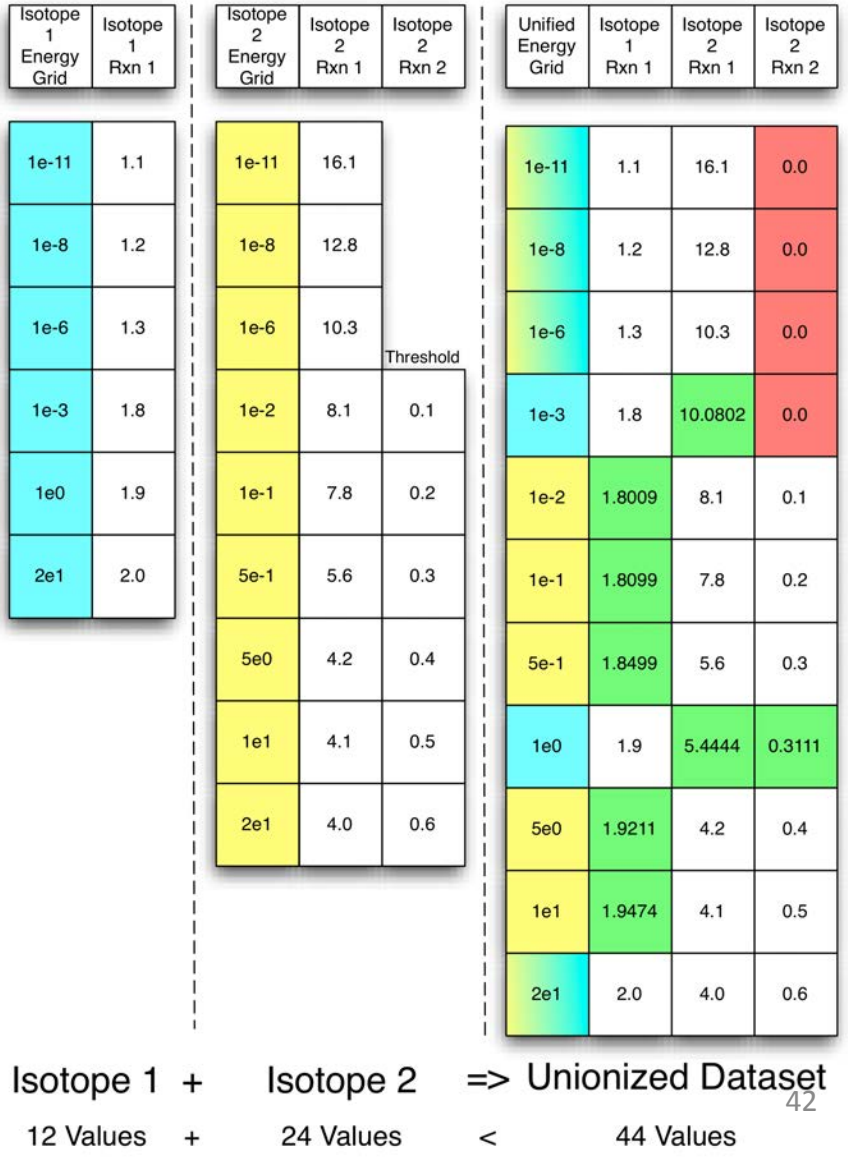
Unionized Energy Grid

- Arrange all the data at a given energy in a single row
  - All isotopes
  - Loads are coalesced
  - Load can be as fast as possible
- Unionize and interpolate gaps
  - No information is lost
  - Information is redundant, higher footprint
- Incoming energies in the scattering and energy distributions are also unionized as to not miss regime transitions



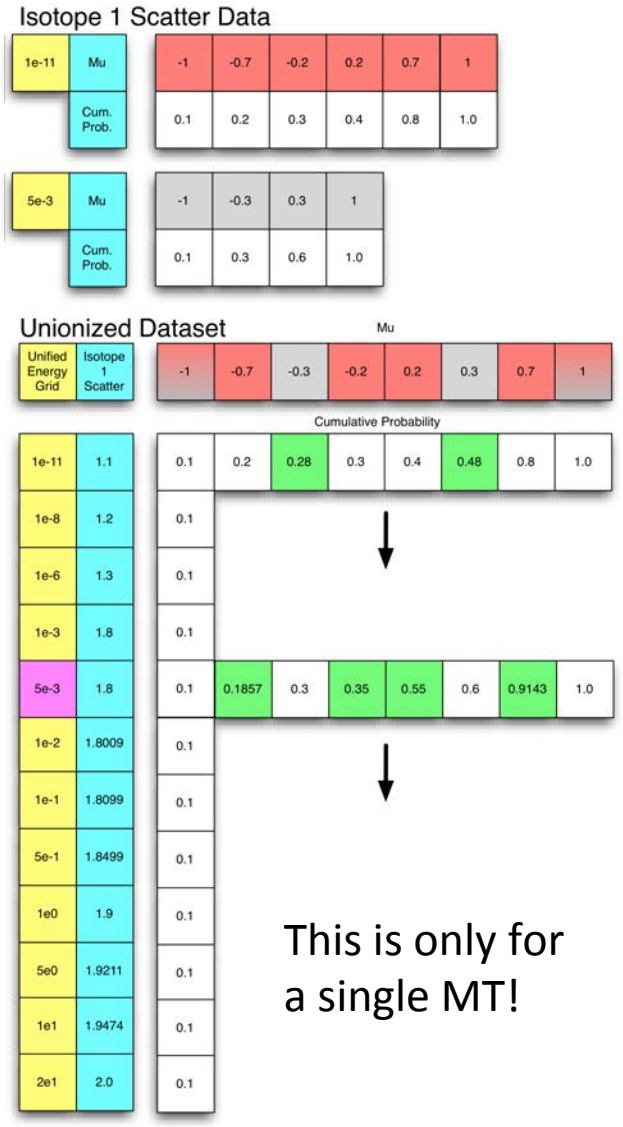
Unionized Energy Grid

- Transforms many 1D cross section vectors into a single 2D matrix indexed by:
  - Incoming energy
  - MT number



Unionized Scattering Matrices?

- Transforms many 2D scattering matrices into a 3D matrix indexed by:
  - Incoming energy
  - Outgoing mu
  - Reaction MT

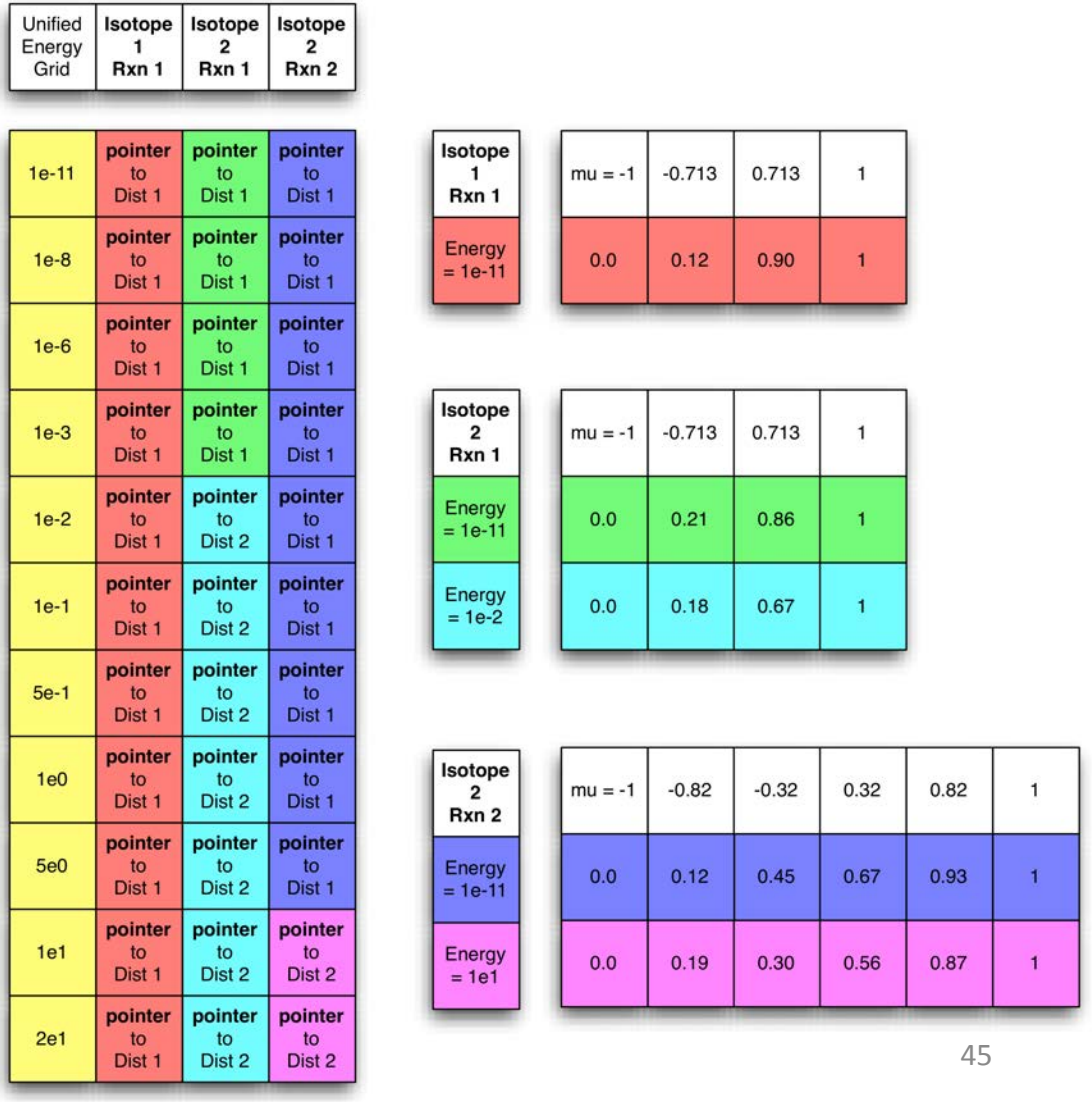


Unionized Scattering Matrices?

- Unionize with main energy grid?
  - Crazy large.
    - MainE * UnionizedCDF * TotalMT
    - U235: 30991 * 598 * 51 floats = 6.4GB!
- Unionize scattering independently
  - Still too large.
    - UnionizedScatterE * UnionizedCDF * TotalMT
    - U235: 103 * 598 * 51 floats = 12 MB
    - U235 and U238: 171 * 1072 * 97 floats = 68 MB
    - U235, U238, O16, H1: 1325 * 1237 * 170 floats = 1.1 GB
- Have to keep matrices in their current formats
  - Might be OK since when they are needed, the isotope and reaction is already known, so a global search is unnecessary
  - As long as the pointer to matrix can be looked up efficiently

Non-Unionized Matrix Access Via Lookup Table

- Cross sections are unionized
- Make a duplicate table that has the distribution pointer at that energy
- Similar to a linked list!
- Only triples the size of the cross section grid
  - 1 copy for angular
  - 1 copy for energy
- The whole table is needed anyway, so not unionizing is OK
- Only option, impact will be seen



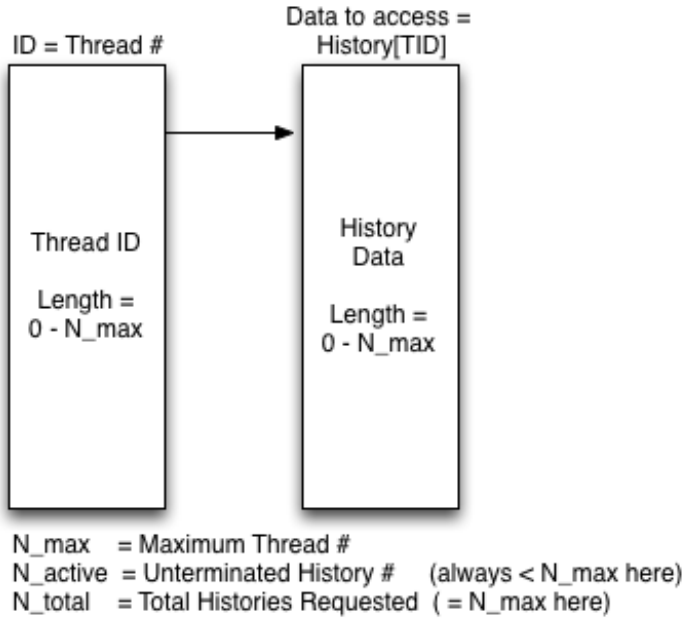
Running Large Datasets

Running Large Datasets

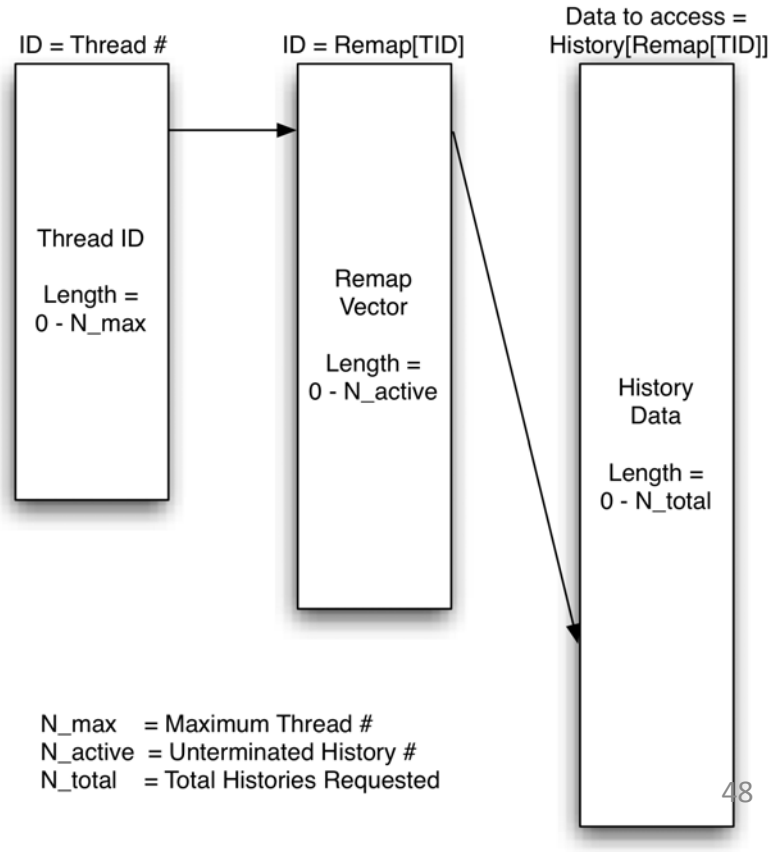
- Typical way to hide latency
- There is a maximum grid dimension size
  - 65,536 for Fermi GPUs
  - 2^{31}-1 for Kepler!
  - Means there is a maximum amount of addressable data
- Batched versions have static addresses between transport iterations
- It is possible to use a remap vector to keep the entire address space full of uncompleted data
  - Data can be larger than the grid size
  - This keeps the thread blocks occupied

Batched vs. Large Dataset

- Batched
  - Thread IDs map directly to data
  - Transported until all particles in dataset have completed
  - Run again until total number is completed



- Large dataset
  - Thread IDs map to data through remapping vector
  - After each transport step, completed particles are eliminated from remap
  - Transport continues until complete



On-the-fly Cross Section Processing

On-the-fly Cross Section Processing

$$\sigma_\gamma\left(T,E_g\right)\cong\sum_{i=1}^{N}\frac{a_{g,i}}{T^{i/2}}+\sum_{i=1}^{N}b_{g,i}T^{i/2}+c_g$$

- Needed if many materials and temperatures are requested
  - Due to limited memory
  - Same reason as CPU, just more extreme
- Methodology by Yesilyurt, et al. incoroporated in MCNP6
- Involves computing cross-sections from regression factors indexed by temperature, reaction, and energy (3D array)
- About 17 terms needed in regression
  - Allowed to vary to minimize size
    - Otherwise unionized dataset will be 17*Temperatures times larger
  - Only worth it if a discrete dataset would be much larger

On-the-fly Mixture Processing

- Macroscopic cross sections from mixture densities done on the fly
- Do not need copies for each mixture
- Does MCNP or Serpent do this?
  - Or do they precompute cross sections for each mixture?

Sorting by Energy

Sorting by Energy

- CUDPP provides fast parallel compaction routines
  - Takes two vectors, a “valid” mask and a set of data
  - Compacts the data into a new vector of sorted data which only contains “valid” members
- Can be used to sort the particle data
  - Thread blocks will access data that are nearest in energy
  - Will keep threads coherent longer for grid search
  - Will maximize data locality for loads

Overloading Each Multiprocessor

Overloading Each Multiprocessor

- The number of threads present in each block is variable, what number to use?
- Benefits
  - Warp scheduler has many particles to choose from, can keep warps coherent
- Drawbacks
  - Increases the memory usage per block, can run into limits
- Want to use as many as possible, the number that will fill the available memory

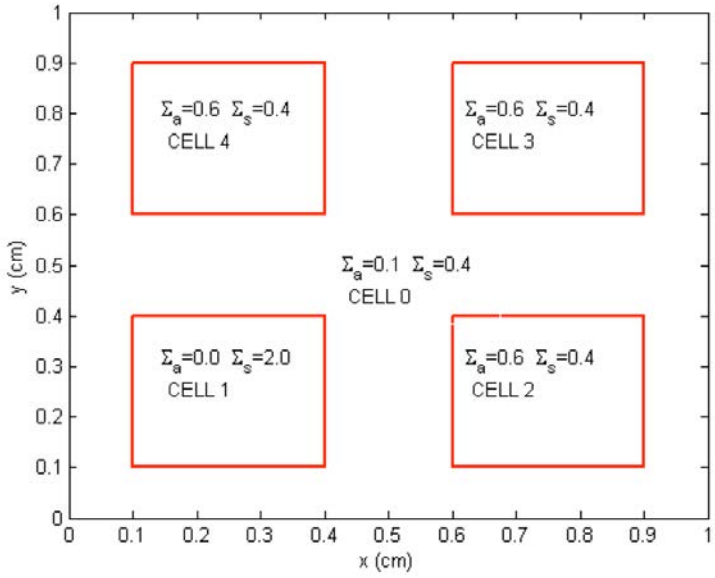
Preliminary Work

Preliminary Work

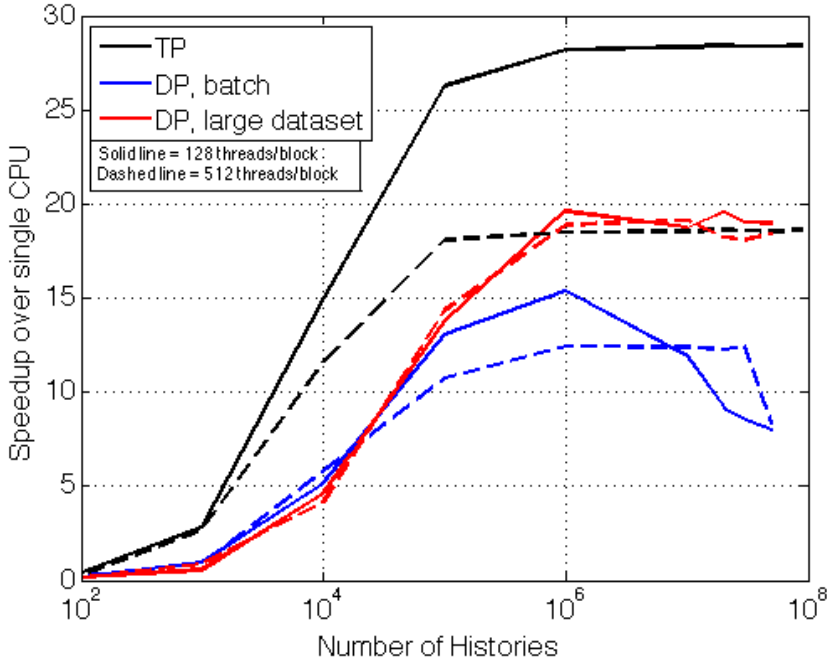
- Scattering problem
  - Compare CPU and GPU implementations
- OptiX benchmark
  - Can a package typically used as a renderer be used for MC?
  - Is it fast enough?
  - Does the “where am I” algorithm work?

Scattering Problem - Setup

- 4 versions written
  - CPU, task parallel, batch
  - GPU, task parallel, batch
  - GPU, data parallel, batch
  - GPU, data parallel, large dataset
- 2D, mono-energetic
- Two reactions
  - Scatter
  - Absorption
- Simple geometry
  - 5 cells
- Meant to highlight differences
  - Global access patterns
  - Speedups
  - Warp scheduler effectiveness



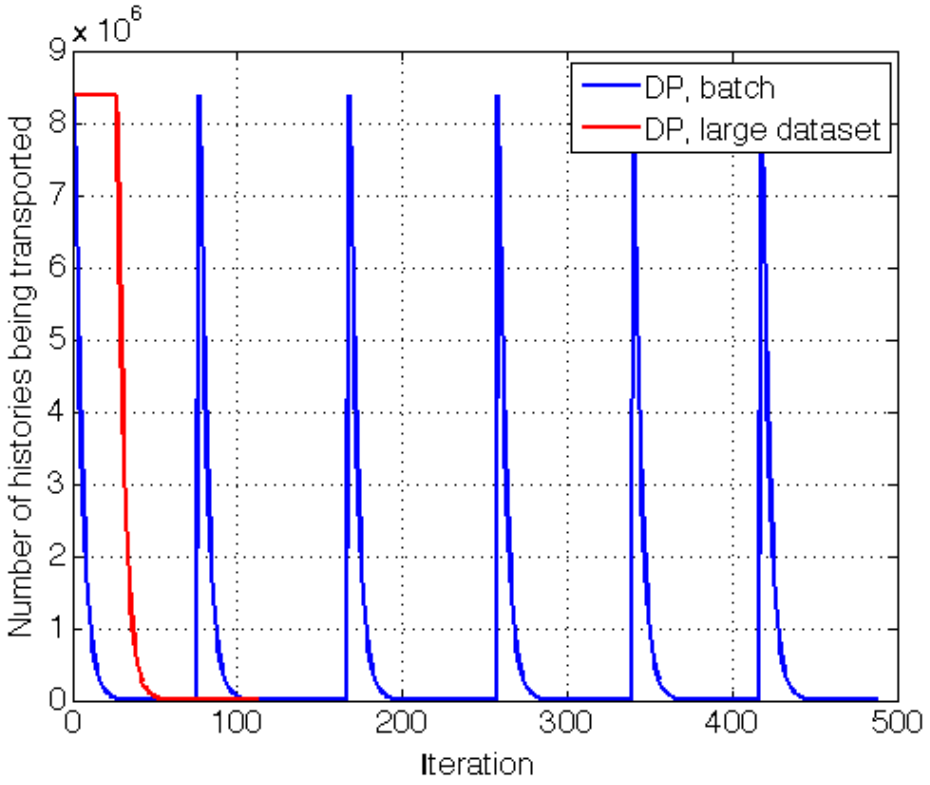
Scattering Problem - Speedup



GPU Parallelism Implementation	% Branch Efficiency	% Warp Occupancy
Task, 128 threads/block	92.4	45.4
Task, 512 threads/block	92.3	33.3
Data, Batch, 128 threads/block	92.8	47.0
Data, Batch, 512 threads/block	92.8	34.6
Data, Large Dataset, 128 threads/block	91.6	51.6
Data, Large Dataset, 512 threads/block	92.2	49.6

- Task-parallel faster than data-parallel except for higher threads/block
  - severely degraded at 512 vs. 256 threads/block
  - Due to warp occupancy?
- Increasing the number of threads per block doesn't seem to affect the data-parallel versions much.
  - Due to the threads all being in relatively close states?
- The large dataset approach is substantially faster than the batched version, which is expected.
- "Branch efficiency" is defined as the ratio of non-divergent thread control branches to total branches
- "Warp occupancy" is defined as the ratio of average active warps per active cycle to the maximum number of warps supported on a SM.
- All the versions have roughly the same branch efficiency
  - Suggests that the scheduler is able to keep the warps full of similarly acting threads for this test problem.
- Large dataset version has slightly higher average occupancy.
  - This is most likely due to the fact that threads are always mapped to active data and the scheduler has a larger thread pool to choose from on average.

Scattering Problem - Active



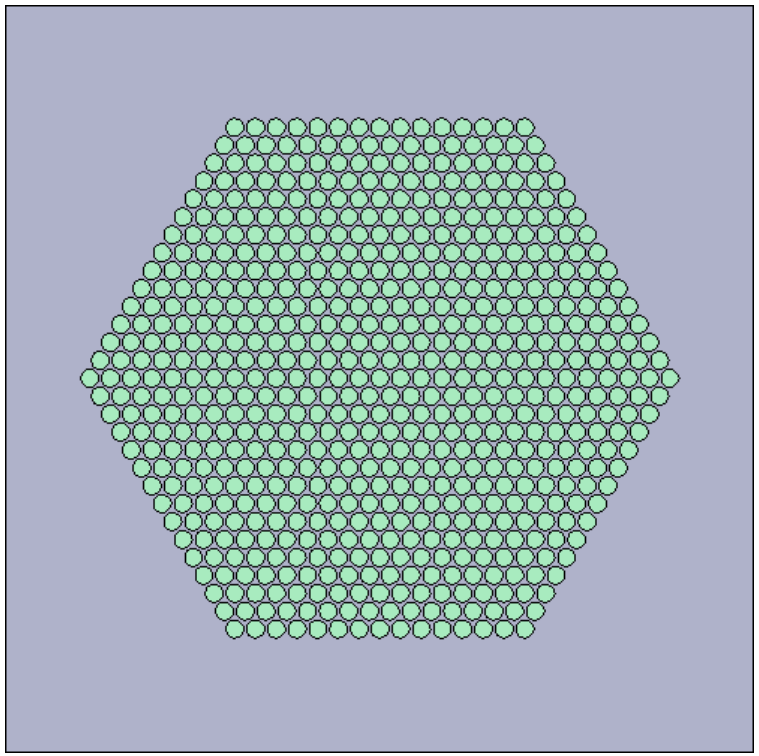
- SMs in the batched version spend much of their time under-utilized.
  - Once a particle is absorbed, the thread remains inactive on the GPU until all histories are completed and the batch is refreshed.
  - In the large dataset version, once a particle is terminated, the thread is immediately remapped to access a piece of active data.
- Remapping keeps the SMs busy until the very end of the problem when the number of remaining histories is less than the maximum thread number.
- Batched version suffers from having to go back and forth to the host more often (it has more iterations to complete all histories) and therefore will have a larger latency penalty.

Scattering Problem - Conclusions

- Main performance limit in the data-parallel versions is the latency incurred by constantly communicating with the host.
  - Since all versions have almost identical memory access patterns (but not instruction orders)
- In problems with more complicated reaction channels, threads/block would be a tuning parameter.
  - Thread divergence will also be a bigger problem

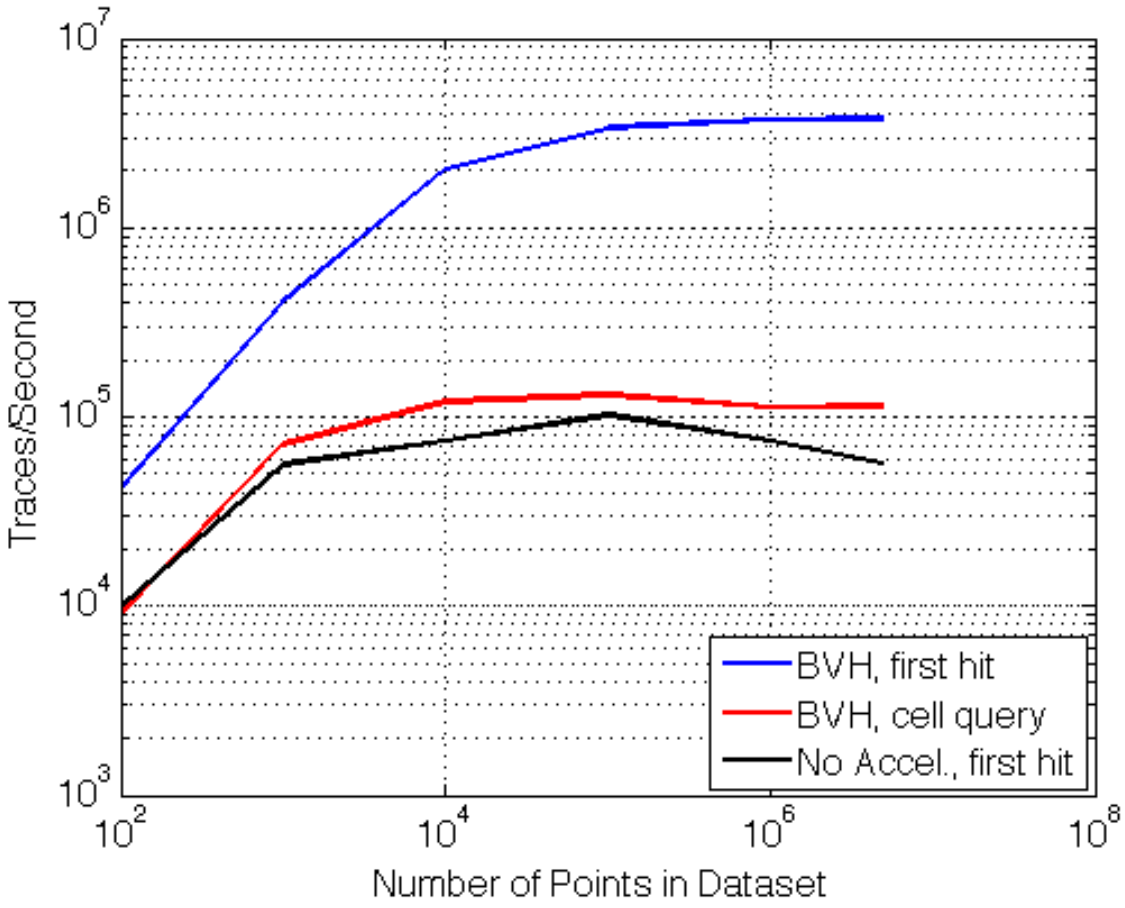
OptiX Benchmark - Setup

- Reactor-like geometry
  - Hex array of 631 cylinders (15 sided)
  - Hexagonal prism
  - Outer bounding box
  - Totals
    - 1270 planes
    - 631 circular surfaces



OptiX Benchmark - Results

- Intersection point only
  - 3 million rays per second can be done, but if the
- "Where am I?"
  - $1x10^5$ rays per second
  - Similar to first hit trace rate with no bounding volume hierarchy acceleration structure.
- The algorithm is implemented as a simple iterative loop
  - may be inefficient and incur many redundant calculations



OptiX Benchmark - Conclusions

- Flexible enough to perform MC ray tracing
- Can plot geometry with no problems
- Shows promise for providing enough ray tracing speed to be used in a full 3D Monte Carlo
  - Nothing to compare against for handling “where am I?” within ray tracing
- Again necessary to operate on large datasets to have good performance
  - data-parallel algorithm in general
  - reduces the impact of latency

Completed Work

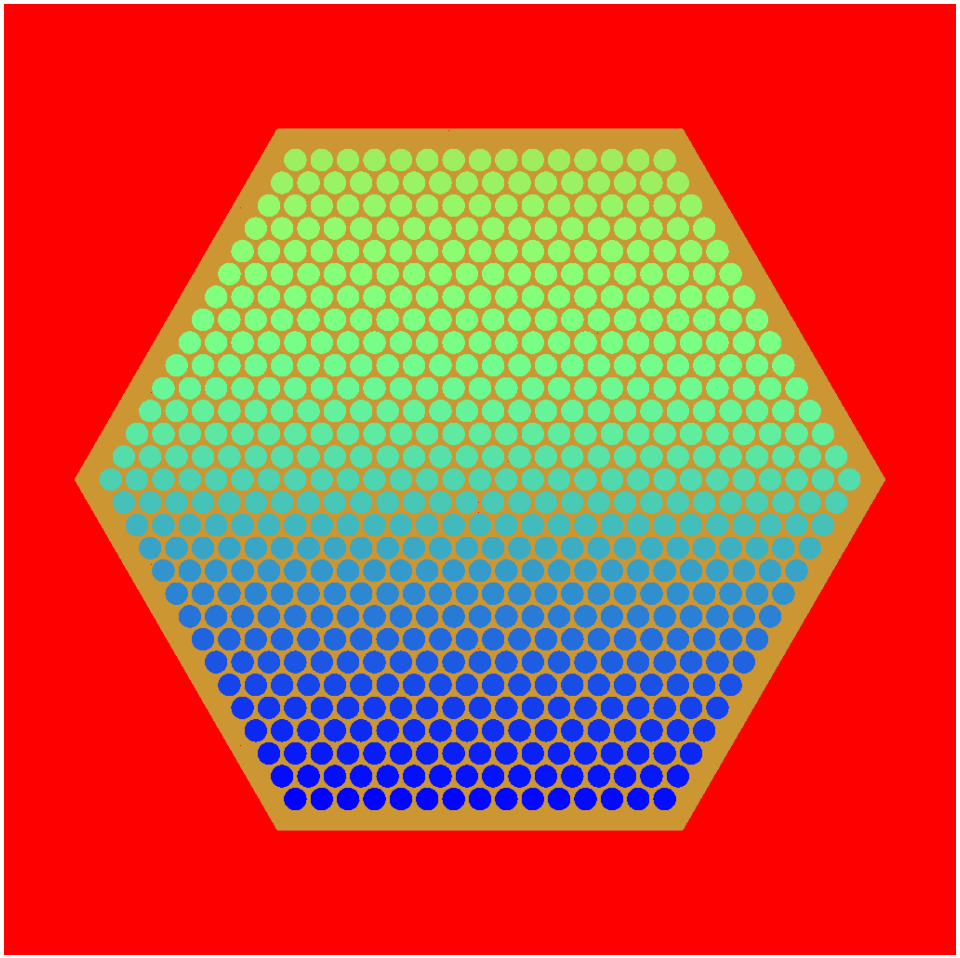
- Geometry routines are completely finished
  - Can plot at an arbitrary resolution at arbitrary slices
  - 3D rendering and/or real-time manipulation could easily be done with openGL
  - “Where am I?” works
  - Thank you OptiX for taking care of 5 of my 6 dimensions!
- Unionized cross sections
  - Using PyNE for loading and parsing
  - Embedded python interpreter into C++
    - Unionize & format cross sections in Python, pass pointers to C++
    - Find and pass scattering and energy tables to C++, pointer arrays created and populated in C++
  - Scattering distributions as of today!

Geometry 1

Colors represent cell numbers

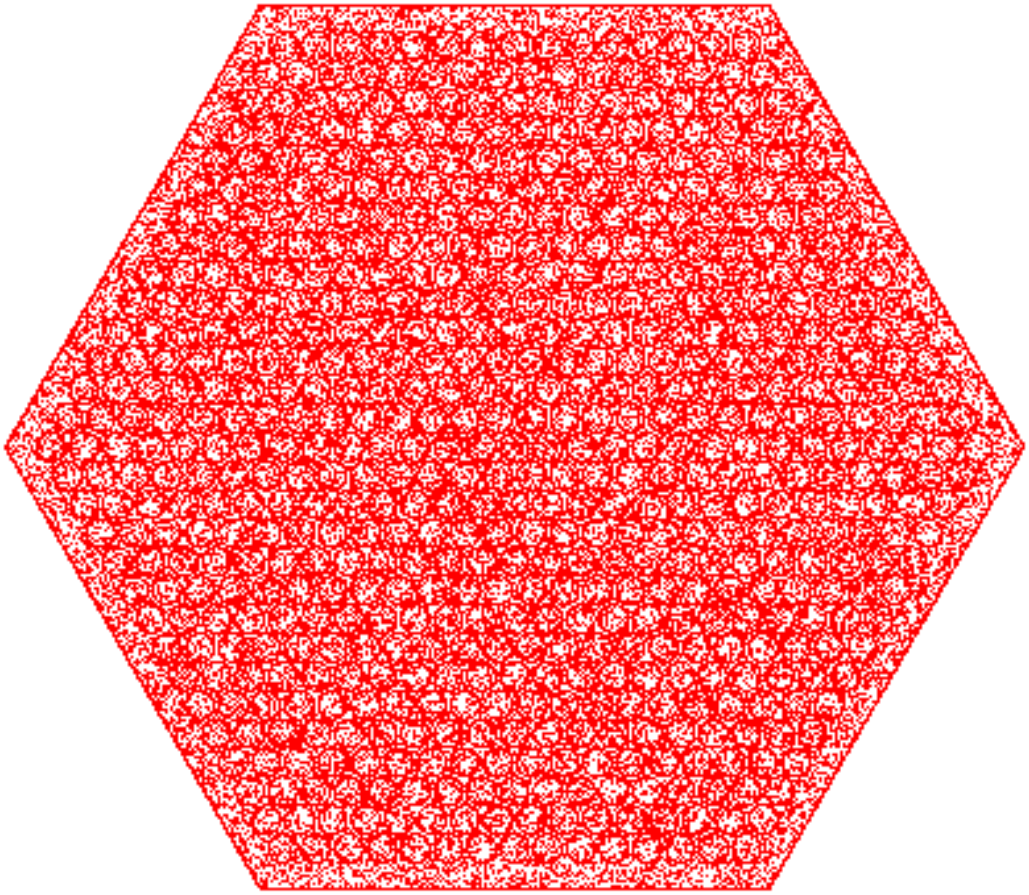
Can easily be set to show materials instead

Colormap can be set to other colors as well

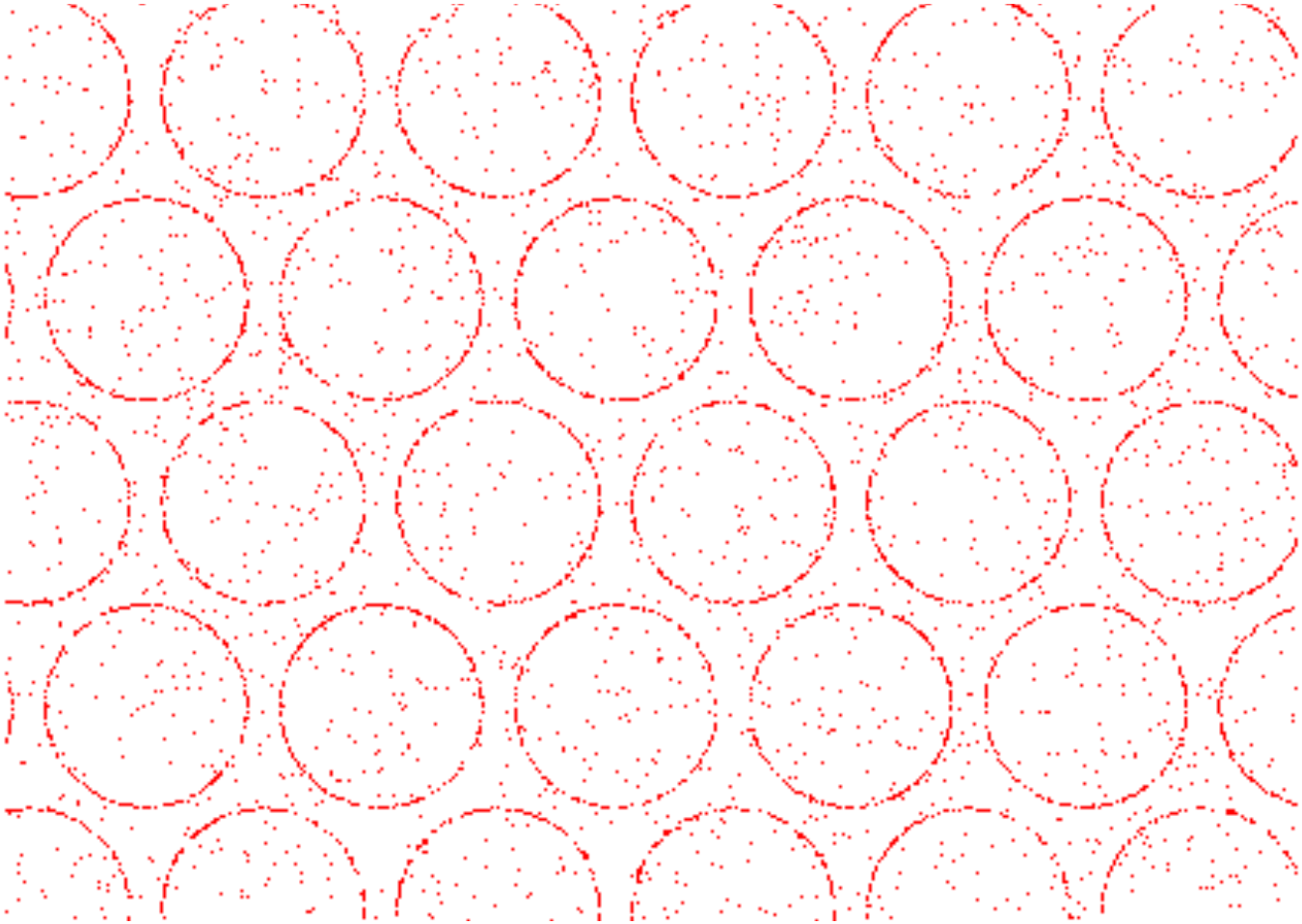


Z=0 slice

Geometry 1 - intersection points

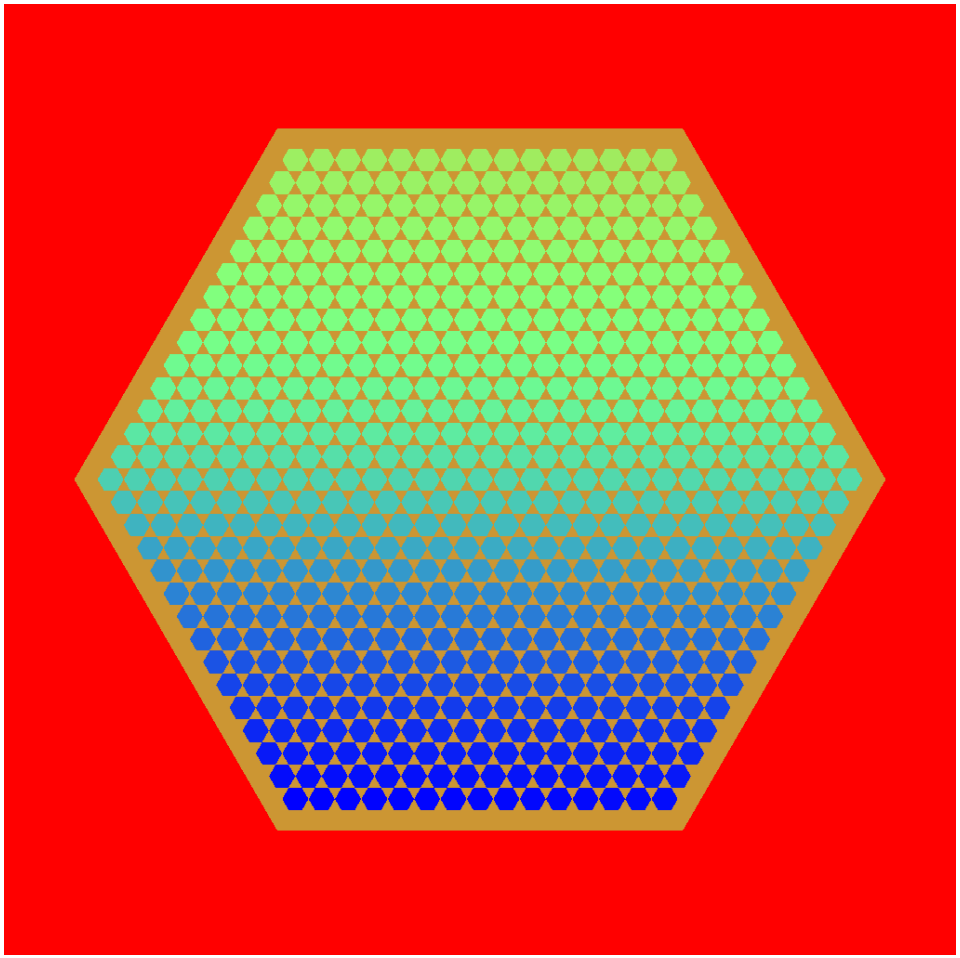


Geometry 1 - Intersection Points



Geometry 2

Doesn't look great since the hex units are the same orientation as the assembly



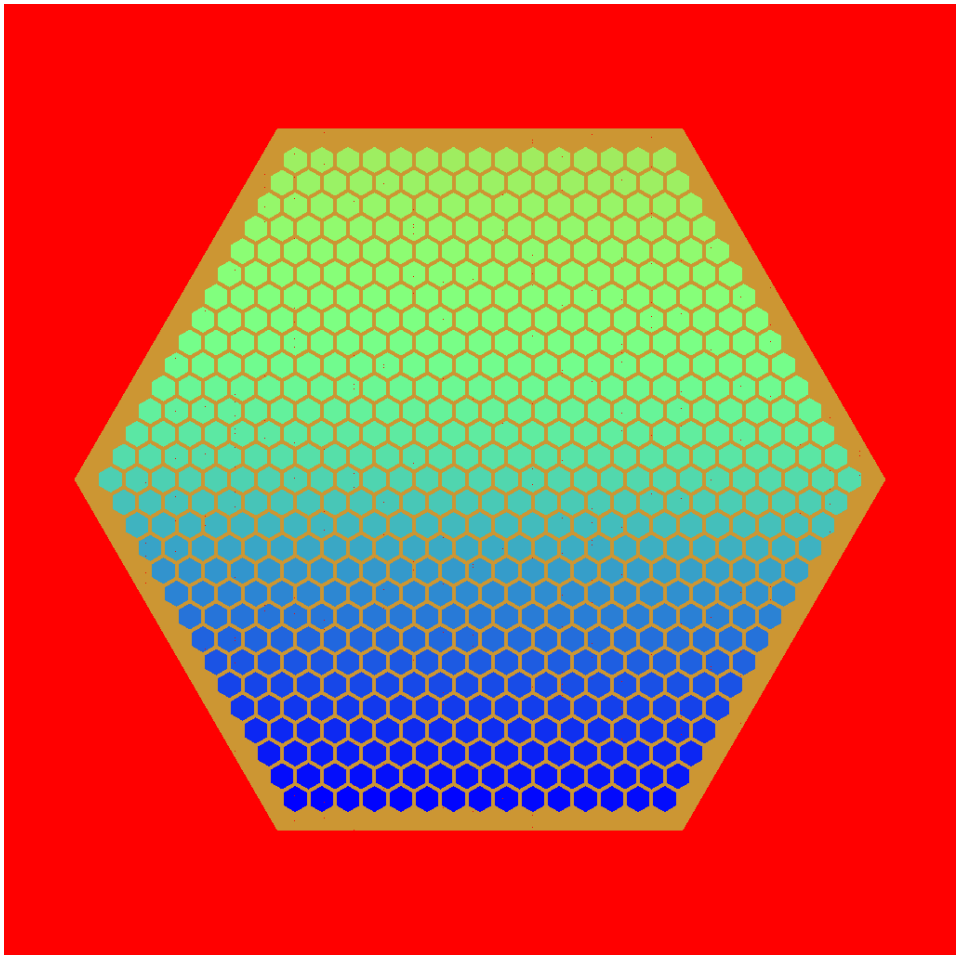
Z=0 slice

Geometry 2

BUT!

General rotations are easy due to the transform node instancing in OptiX!

I changed one number and one function prototype in my code to do this.



Z=0 slice

Ongoing Work

Ongoing Work

- Adding physics routines
  - Previous versions did not make accurate spectra
- Cross section routines
  - Energy distributions
  - Nu distributions
  - Verification (how?)
- Python wrappers for my C++ routines
  - Nikola will be helping with this
  - SWIG
  - Replace the main function as it only controls program flow, doesn't do any heavy lifting
  - Will allow a very integrated experience
    - Allow persistent sessions
    - Integrated plotting and analysis routines attached to the data

Thank You!

Training the Next Generation

This material is based upon work supported by the Department of Energy National Nuclear Security Administration under Award Number(s) DE-NA0000979

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or limited, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Questions?

Demo?