Standard Acts of Liaisons: Scientific Computing Success Stories

20 Years of Excellence in Computational Science
OLCF
Oak Ridge Leadership Computing Facility
1992—2012

Rebecca Hartman-Baker
Liaison Task Lead, Scientific Computing Group
Oak Ridge Leadership Computing Facility
hartmanbakrj@ornl.gov

Outline

- About Liaisons
- Liaison Role
- Success Stories

About Liaisons

- PhD-level scientists with expertise in computation
  - Astrophysics, biology, chemistry, climate, computer science, engineering, materials science, mathematics, nuclear physics, plasma physics, etc.
  - Experienced computational scientists with one thing in common
- Liaisons matched with INCITE projects based on science, mathematical, and algorithmic expertise
  - Can’t always match for science first, e.g., I am not a chemist, but I am familiar with their math and algorithms
- Our motto: Whatever it takes!

Liaison Role

- Liaisons are collaborators whose unique expertise with leadership-level computers will enhance your experience and help you get more science done
- Levels of liaison support
  – Level 1: User support +
  – Level 2: Paratrooper – fix a specific problem in your code, O(1 month)
  – Level 3: Embedded member of code development team and science collaborator

Liaison Role

- Typical liaison activities
  - Profile code performance, providing feedback to code team
  - Code porting
  - Implement solutions to problems experienced by application scientists
  - Advocate for users regarding tools, libraries, etc.
  - Collaborate scientifically

Scientific Computing Success Stories

- Recent successes, chosen to illustrate our typical tasks
- Many more success stories where these came from
- Next year: your code?

Code Profiling

- Code profiling: similar to energy audit of your home
  - Look at code's behavior, find bottlenecks, time sinks (leaking windows, old weather stripping)
  - Make recommendations for improvements to your code (replace windows, install a new water heater)
- Tools we typically use for this (you can try it too!):
  - Vampir and VampirTrace
  - Craypat

VampirTrace/Vampir

- VampirTrace: instrument code to produce trace files
  - Compile with VT wrapper, run code and obtain trace output files
- Vampir: use to visualize trace
  - Run in interactive job of nearly same size as job that produced trace files
  - Server on interactive job serves as analysis engine to local front-end

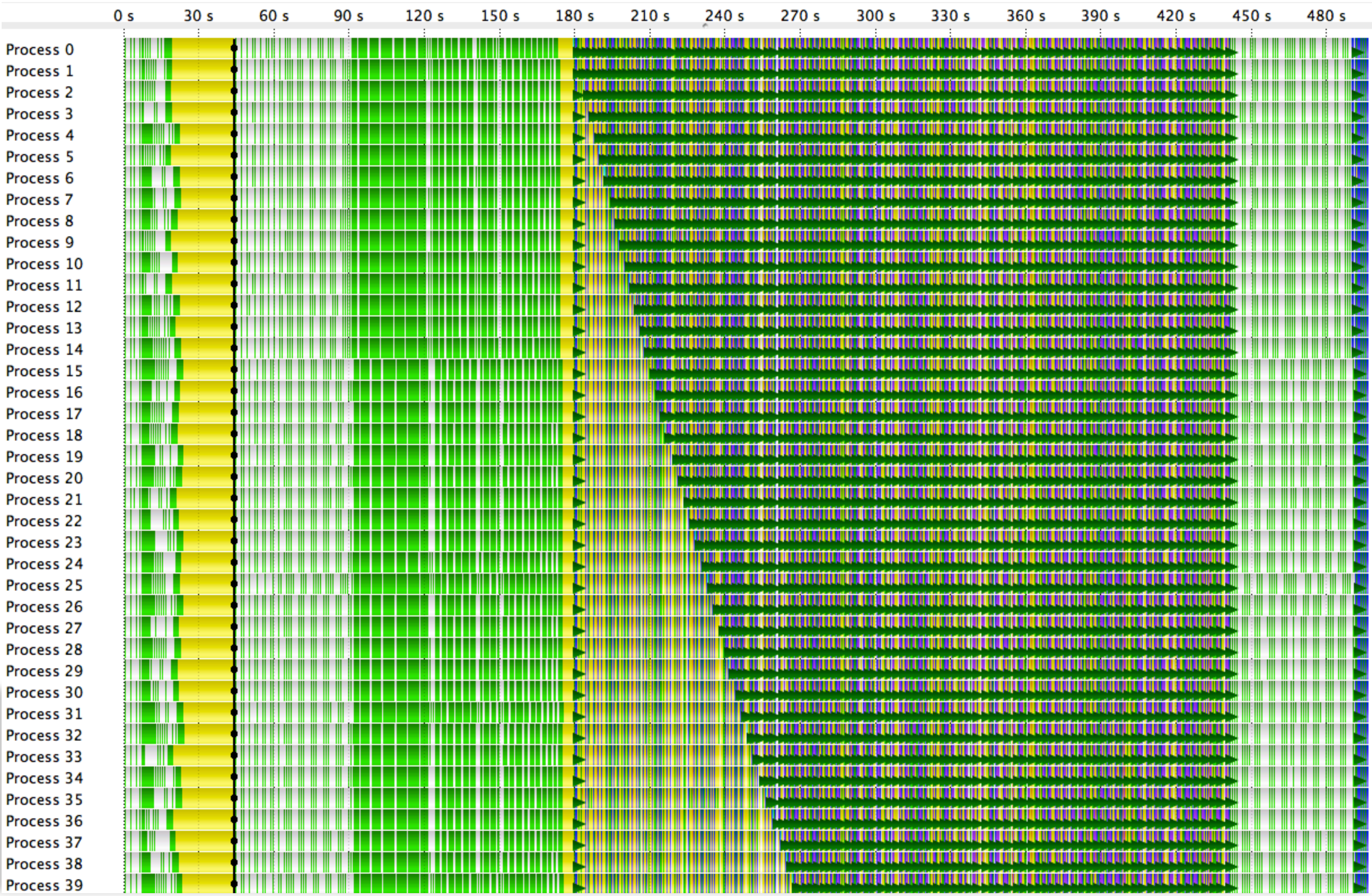
CrayPAT

- Package for instrumenting and tracing codes on Cray systems
- Run instrumented code to obtain overview of code behavior
- Re-run with refined profiling, to trace most important subroutines

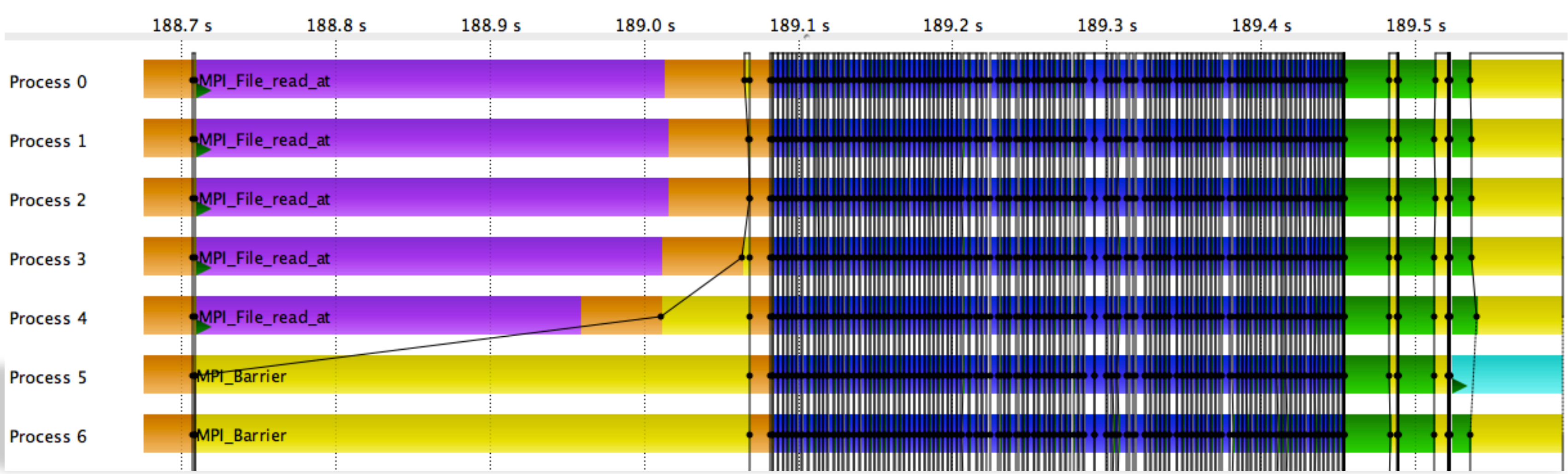
Code Profiling: Example 1 (Bigstick)

- I profiled Bigstick, a configuration interaction nuclear physics code
- Promising for large-scale nuclear configuration calculations, because it does not require as much memory
- But, does not scale well
- My task: find out why
- Used Vampir/VampirTrace to create graphical representation of code performance

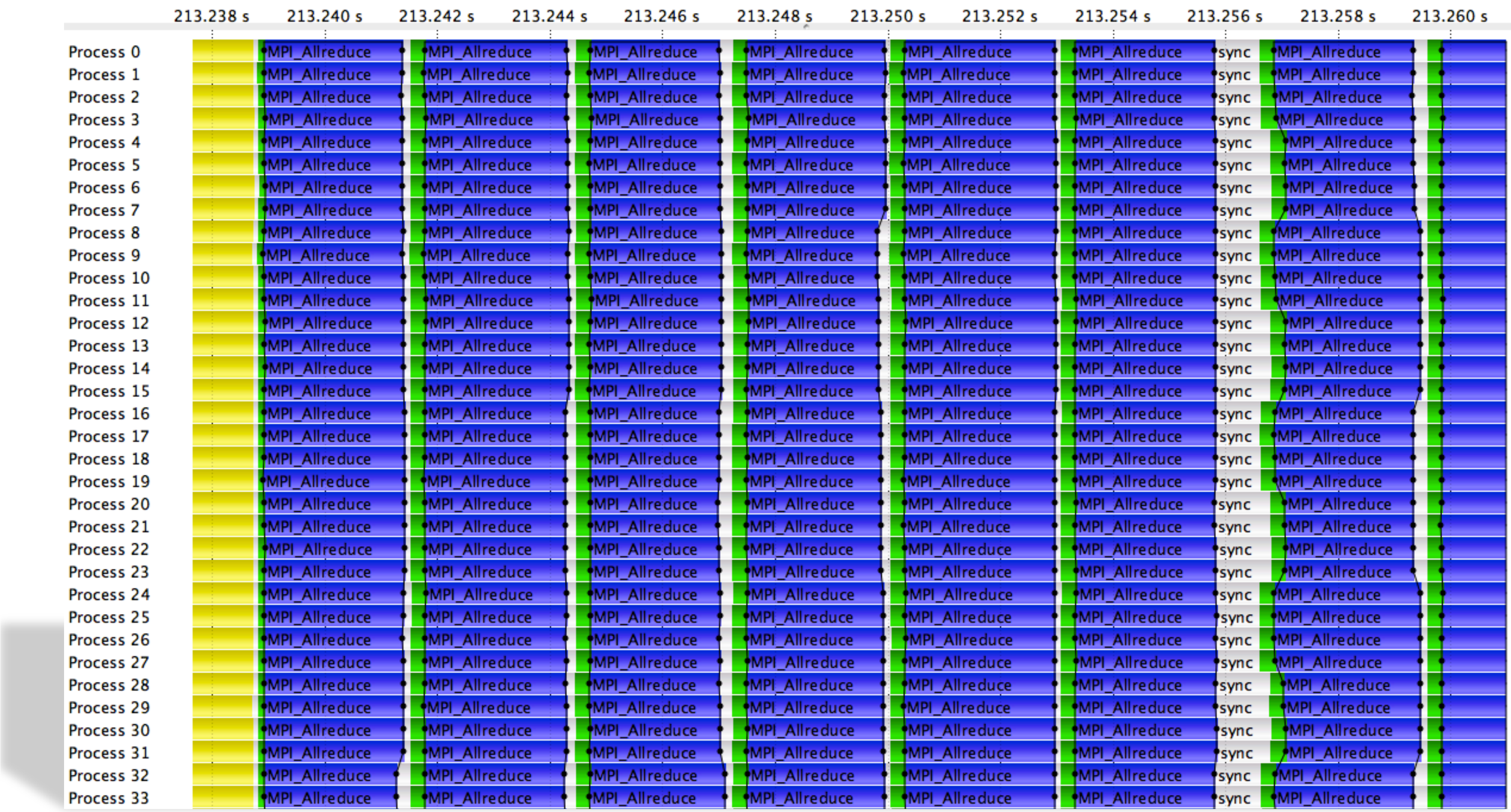
Top-Level Overview



Three Steps within Triangle



Block Reduce Phase



Bigstick: Analysis

- Triangular pattern in overview reminiscent of sequential algorithm applied across processors
  - Digging deeper shows in orthogonalization phase, processors held up by single processor writing to Lanczos vector file
  - Suggestion: reduce amount of orthogonalization performed
- Disproportionate time spent in `MPI_Barrier` (~30%)
  - Indicative of load imbalance
  - Barriers are within `clocker` subroutine, used for performance timings, obscuring evidence of load imbalance
- Majority of time in block reduce phase spent in `MPI_Allreduce`
  - Combining Allreduces could improve performance

Code Profiling: Example 2

- We first profiled j-coupled version of NUCCOR with CrayPat
- Discovered it spent >50% of its time sorting in test benchmark
  - Found it was using highly inefficient bubble-sort-like algorithm
  - Replaced “Frankenstein sort” with heapsort, reduced sorting to ~3% of time
- Asked collaborators what they were sorting, and why?
  - Their response: “We’re sorting something?”
- Removed sorting altogether, code worked just fine, and ran 30% faster on long benchmark

Code Profiling: Example 2 (continued)

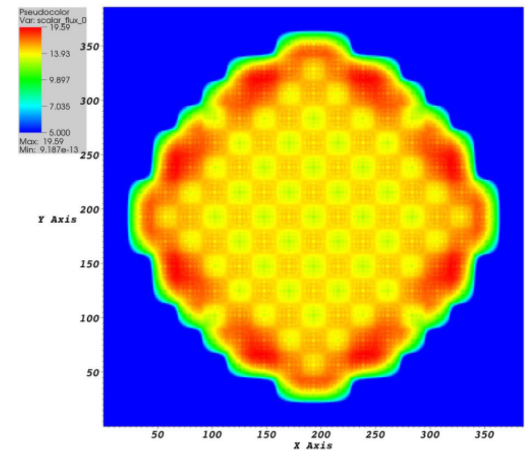
- We next profiled standard version
- Discovered it spent nearly 70% of time in single subroutine: `t2_eqn_store_p_or_n`
- This subroutine became focus of subsequent work with NUCCOR

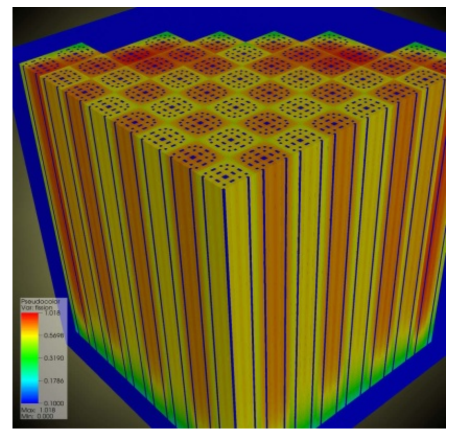
Code Porting

- Wayne Joubert, Denovo

What is Denovo?

- Denovo is a radiation transport code used in advanced nuclear reactor design
- It solves for the density of particle flux in a 3-D spatial volume such as a reactor
- In particular, it solves the six-dimensional linear Boltzmann equation (3-space, 2-angle, 1-energy)
- Denovo scales up to 200K cores on ORNL's 2.3PF Jaguar system.
- It is part of the CASL project (Consortium for Advanced Simulation of Light Water Reactors) and the SCALE code system (Standardized Computer Analyses for Licensing Evaluation)
- It was selected as an early port code for Titan

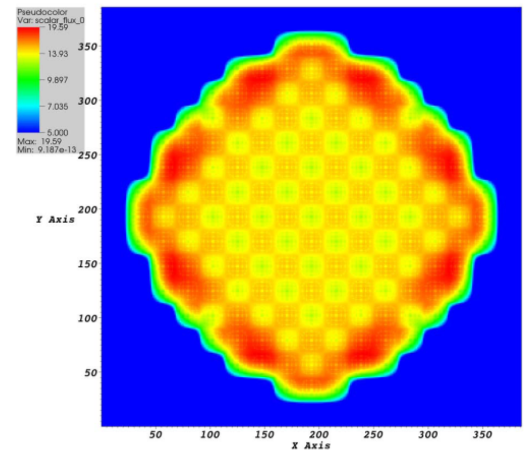


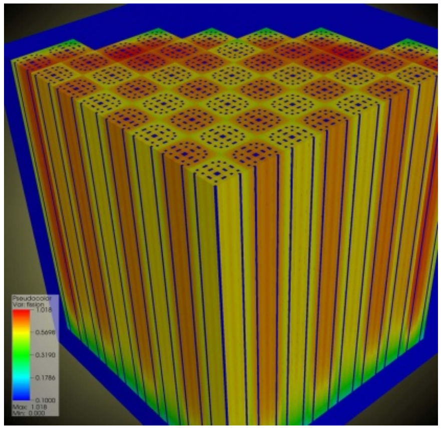


Oak Ridge National Laboratory

Denovo Algorithms

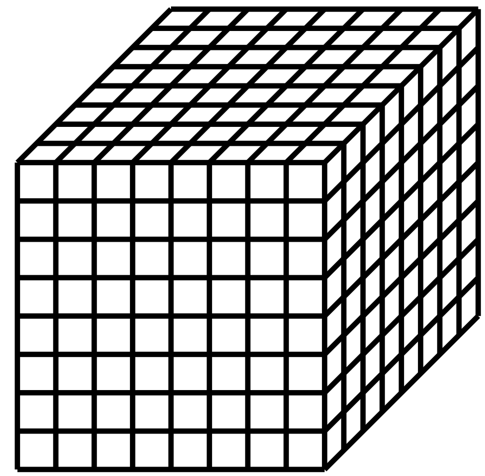
- Primary algorithms: the discrete ordinates method, 3-D sweep, GMRES linear solver and various eigensolvers, e.g., Arnoldi
- The execution time profile has a very prominent peak: nearly all the execution time (80-99%) is spent in a 3-D sweep algorithm.
- Because of this, the 3-D sweep must be the central focus of any effort to port Denovo to a accelerator-based system
- However, the sweep is a complex algorithm that is difficult to parallelize efficiently.

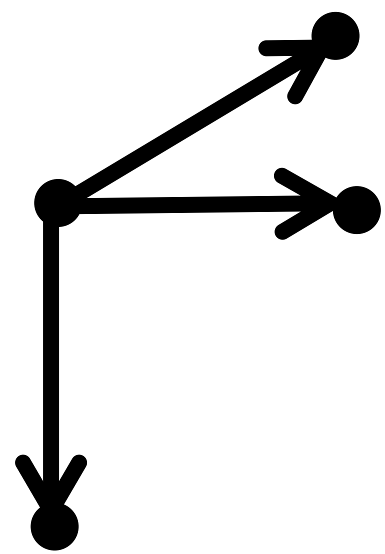




3-D Sweep Algorithm: Description

- Denovo is based on a 3-D structured grid
- The data dependency for the sweep operation is specified by a 4-point stencil
- The result at every gridcell is dependent on the result at the immediately lower gridcells in X, Y and Z.
- This induces a wavefront computation pattern – a sequence of diagonal planes sweeping in from a corner.
- Thus, results at the far side of the grid cannot be computed until results at the near side are completed
- For standard parallel grid decompositions, most of the processors will be idle much of the time



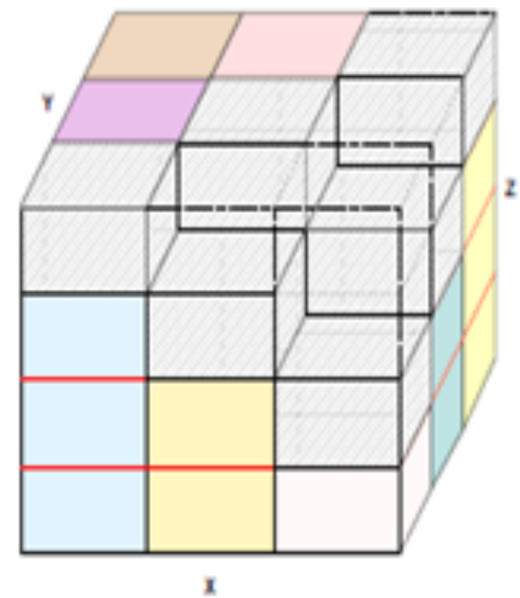


How to Program the Sweep on the GPU?

- Decide what language / parallel API to use to program the GPU.
- Options:
  1. CUDA: a minor extension of C/C++ for GPU thread programming, also available for Fortran 90
  2. OpenCL: a multi-vendor standard similar to CUDA
  3. Compiler directives: similar to OpenMP (PGI, CAPS, Cray, ...)
- Sweep is a complex algorithm, with many dimensions. Directives may not be flexible enough or expose enough hardware functionality to get the needed performance.
- NVIDIA support OpenCL, but going forward CUDA will be better supported and more in-sync with new hardware features.
- Thus use CUDA. Use C++ for consistency with Denovo base language.

Mapping the Algorithm to the GPU

We have many candidate dimensions for parallelism: space (3), energy, moment/angle, octant, and also unknown (4 unknowns per gridcell for this discretization).



We are told by NVIDIA that we need 4K-8K threads for the GPU to keep all GPU streaming multiprocessors busy and cover various latencies.

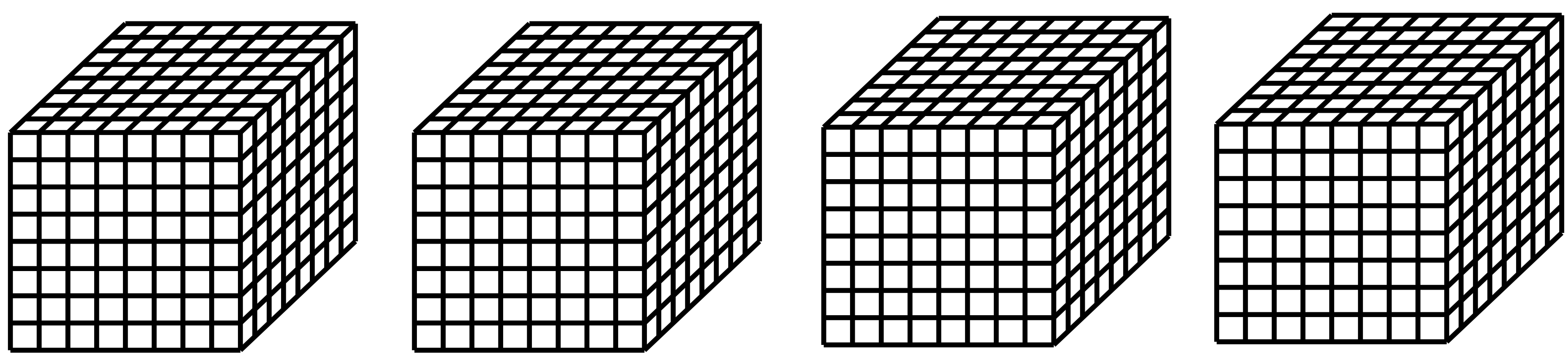
Also need the right kind of parallelism – proper decoupling of data.

Also must have good memory access patterns (reuse of data loaded from global memory, coalesced stride-1 memory references, good use of registers, shared memory, caches on the GPU).

<u>Approach</u>: explore each problem dimension for potential thread parallelism.

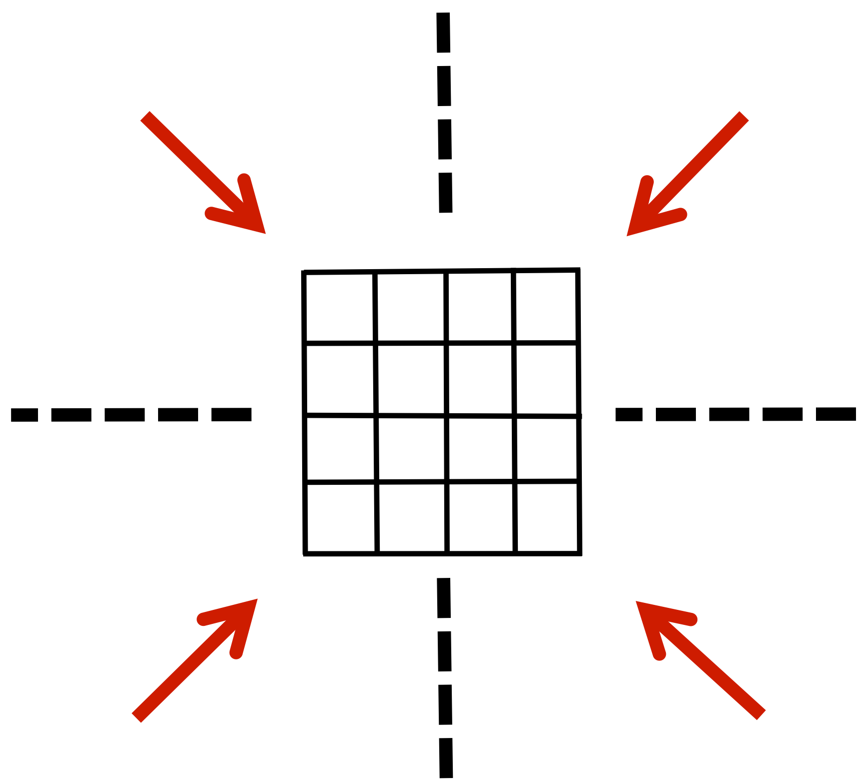
1. Parallelism in Energy

- Denovo exposes energy as a parallel dimension.
- Values for different energies are entirely independent in the 3-D sweep, thus the algorithm is embarrassingly parallel along this axis (!).
- Model problem has 256 energy groups – this helps, but we need to look further in order to get to 4K-8K threads.
- Also need to use some of this 256 for node parallelism.



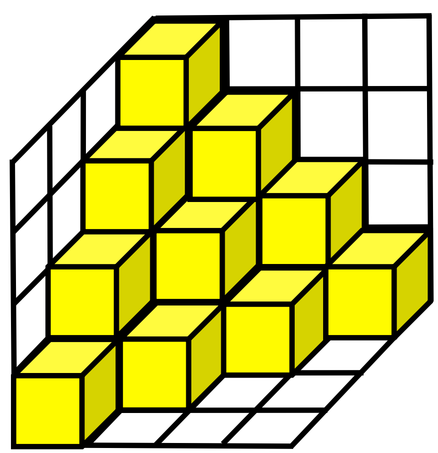
2. Parallelism in Octant

- Algorithm requires sweeps from 8 different directions.
- Sweep directions are independent, thus another 8X thread parallelism (!).
- Small issue: different octants update the same output vector, so we need to schedule properly to avoid write conflicts.



3. Parallelism in Space

- We have this recursion, as mentioned before, that makes the computations non-independent.
- However, the global KBA algorithm can be applied at this small scale (!).
- Set up block wavefronts, assign blocks to threads.
- Sync between block wavefronts.



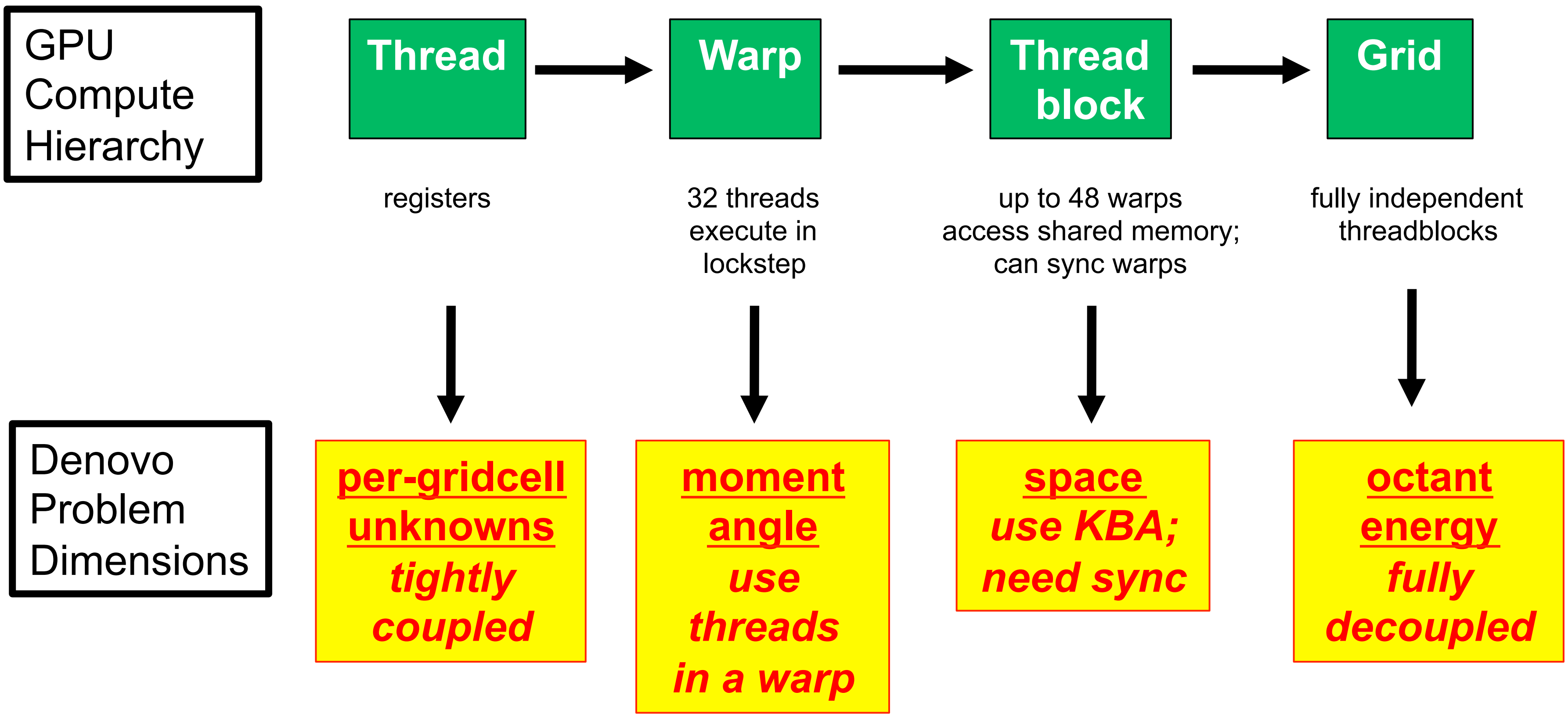
Performance

- With this parallelization scheme, code performed at only about 1% of peak flop rate, much lower than predicted by the performance model
- Reason: excessive use of registers caused spillage to main memory, thus poor performance
- Needed to find more/better axes of parallelism

4. Parallelism in Angle, Moment

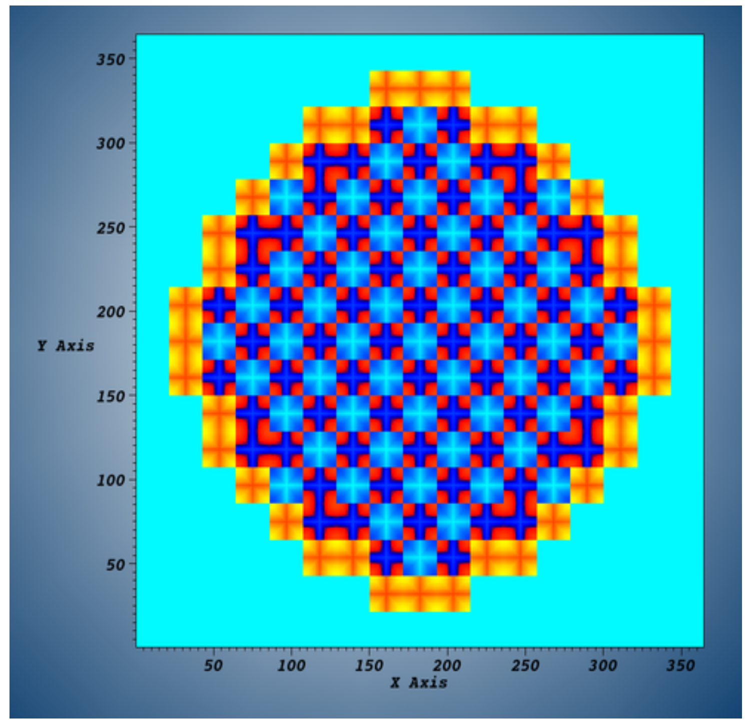
- A new strategy to parallelize the moment/angle axes at the gridcell level – map these axes to CUDA threads in-warp.
- Small dense matrix-vector products are perfect for thread parallelism – store vector in shared memory, relieve the register pressure.
- The two small matrices are the same across all gridcells (!), so they can be retained in L1 cache, to reduce a potentially high source of memory traffic.

Summary of Mapping of Dimensions



Results: Test Problem

- 32x32x128 gridcells
- 16 energy groups
- 16 moments
- 256 angles
- Linear discontinuous elements – 4 unknowns per gridcell

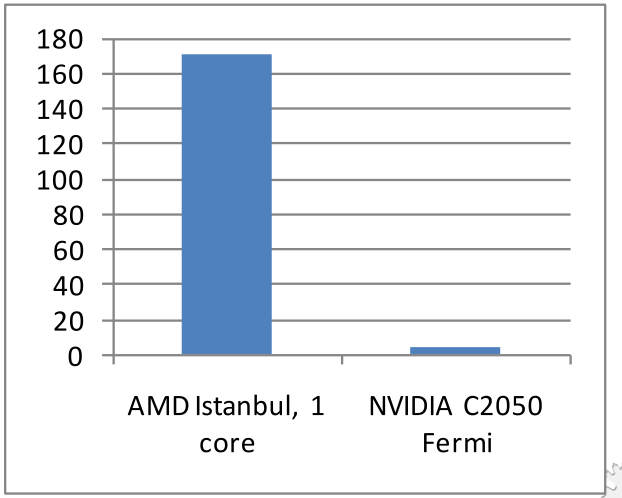


Results: Sweep GPU Performance

- Single core (AMD Istanbul) / single GPU (Fermi C2050) comparison

	AMD Istanbul 1 core	NVIDIA C2050 Fermi	Ratio
Kernel compute time	171 sec	3.2 sec	**54X**
PCIe-2 time (faces)	--	1.1 sec	
TOTAL	**171 sec**	**4.2 sec**	**40X**

NVIDIA Fermi is 40X faster than single Opteron core



Observations

- 40X faster than Istanbul core.
- Istanbul is 6-core, so Fermi about 7X faster than the entire Istanbul processor.
- For both CPU and GPU, code attains about 10% of peak flop rate – this is considered good for this algorithm.
- Expect more optimizations to be possible going forward.

Solving Code Problems

- Valentine and VPIC

VPIC

- 3-D electromagnetic relativistic particle-in-cell simulation code
- Ticket came in about VPIC failing at high core counts
- I asked newest colleague Valentine Anantharaj to take a look, with support from two experienced liaisons (Hai Ah Nam and myself)
- With Hai Ah's help, he isolated location of failure
- Libraries were incompatible because of version changes; recompiling with fresh environment and proper modules loaded eliminated the error

User Advocates

- Markus and Vasp
- Queue priority
- Software needs

VASP

- Vienna Ab-initio Simulation Package
- Licensed software, builds maintained by Markus Eisenbach
- Optimized builds that run more efficiently on Jaguar

Queue Priority

- Liaisons have science-area expertise, and understand unique needs of users
- Advocate and explain why exception is needed
- Advocate for users whose allocations have been exhausted

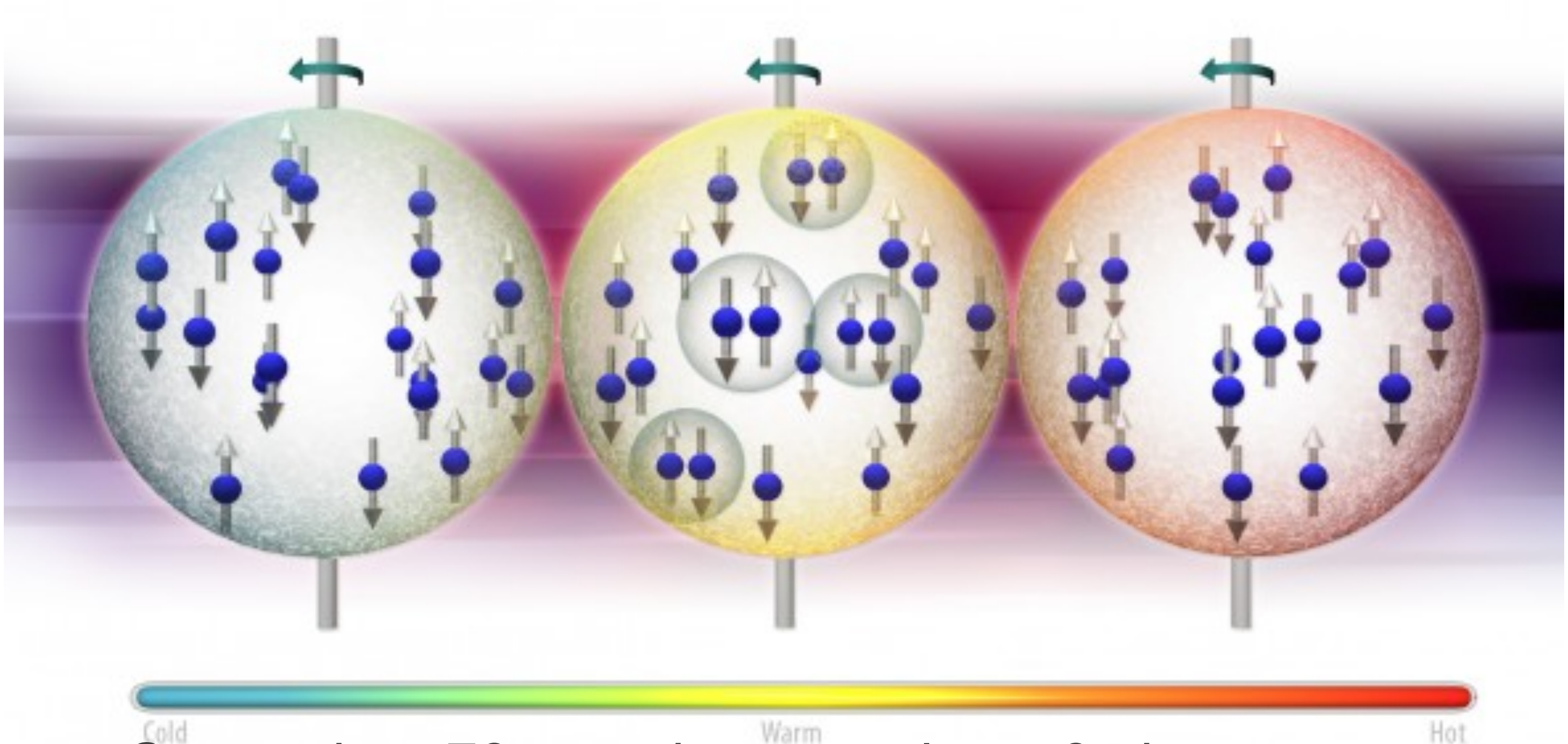
Software Needs

- Advocate for users' software needs
- Serve as translator between projects and Resource Utilization Council (RUC)

Scientific Collaboration

- Hai Ah Nam, Nuclear Physics

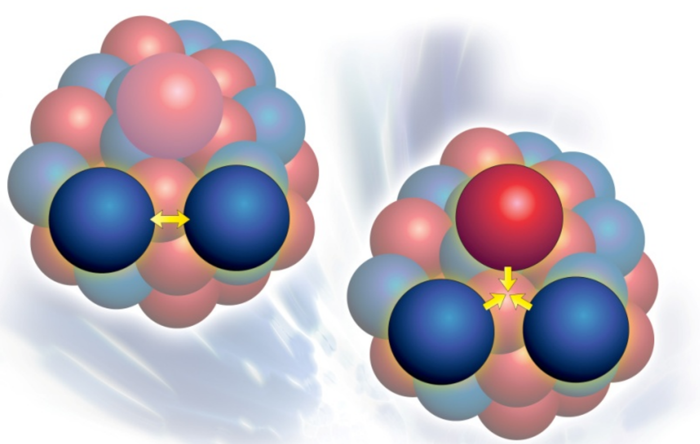
Nuclear Physics

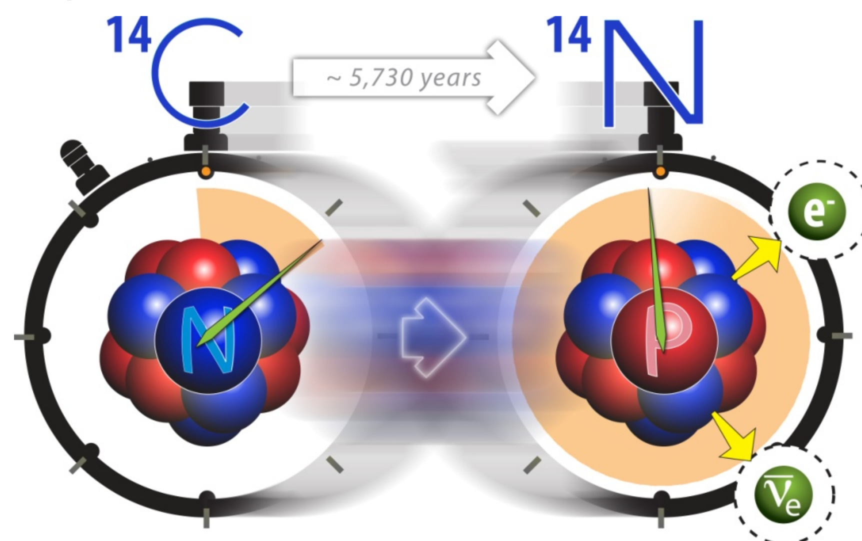


- Germanium-72 experiences at least 2 phase transitions
- Phase transitions a function of pairing of neutrons, rotation of nucleus, and temperature
- Reminiscent of superconductors

Nuclear Physics

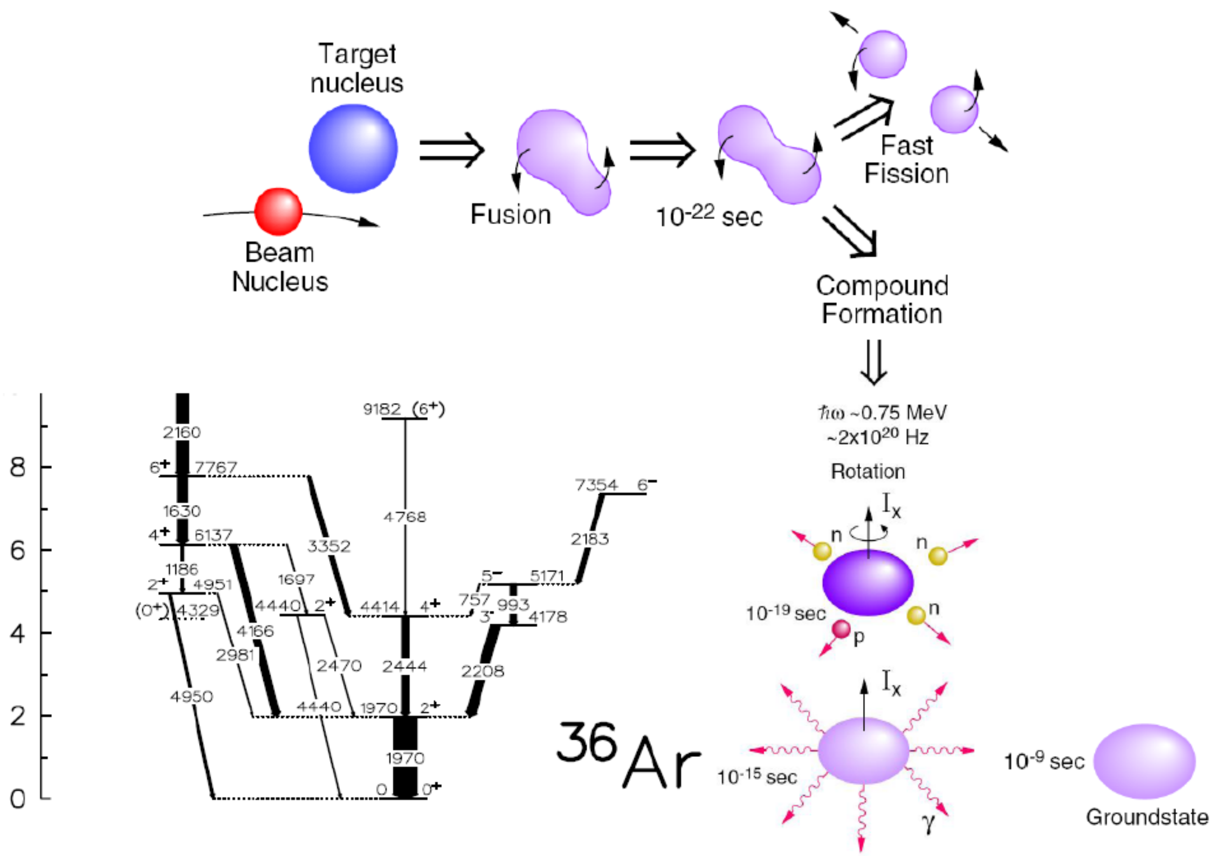
- Studying Carbon-14 half-life with three-body interactions
  - Easy and cheap to approximate nuclear forces with two-body interactions (p+p, p+n, n+n)
  - For better accuracy, need three-body interactions (3p, 2p+1n, 1p+2n, 3n) – using 2-body only underestimates half-life
  - BUT, adding 3-body interactions adds significant computational cost and memory requirements
  - Fortunately, on Jaguar this calculation is possible





Visualizing Spontaneous Fission of Heavy Nuclei

- Collaboration with A. Staszczak, UT/ Warsaw
- Project of summer student Elizabeth Morris
- In collaboration with visualization liaisons Dave Pugmire, Sean Ahern, Ross Toedte



Oak Ridge National Laboratory

Spontaneous Fission of Fermium-258

- Unusual, unexpected deformation of nucleus

Conclusions

- Liaisons are a valuable resource for INCITE projects
- Liaisons have unique HPC skills that projects can take advantage of
- Whatever it takes!

Acknowledgments

- Wayne Joubert
- Hai Ah Nam
- Scientific Computing group
- INCITE project participants

Questions?