Abstract

The Boltzmann equation provides high fidelity simulation of a diverse range of kinetic systems. A common application of this equation is to simulate neutral particle transport. An interdisciplinary team developed a novel algorithm, Quasi-Diffusion Accelerated Monte Carlo (QDA-MC), to solve the neutron transport equation. The algorithm decouples particle scattering and absorption to dramatically reduce the complexity of the stochastic simulation. The change provides good scalability and performance in both traditional multi-core/ many-node architectures and GPU-accelerated architectures. While we focus only on a one-dimensional system in this study, the lessons learned will be largely applicable to multi-dimensional systems as well.

Co-Design, What and Why?

- Traditional Scientific Code Development entails
  - Computational Scientist develop the code to reflect the physics of the problem
  - Followed by architecture specific optimization by the Computer Scientists and Engineers
- Frequently, fundamental algorithmic changes might be needed to get the best performance.
- Computer Scientists lack the specific understanding of the scientific domain to realize that such changes are feasible.
- A continued discussion process between computer scientists and computational scientists can incorporate algorithimic changes in the development process to get the best performance for different architectures.

1D Transport Equation

$$\mu\frac{\partial}{\partial x}\psi(x,\mu) + \Sigma_t\psi(x,\mu) = \frac{1}{2}\left[\Sigma_S\psi(x) + Q(x)\right]$$

- ψ is called the angular flux
- ϕ is called the scalar flux
- Σ_s is the scattering cross-section
- Σ_a is the absorption cross-section
- $\Sigma_t = \Sigma_s + \Sigma_a$ is the total cross-section
- Q is a fixed source
- μ is called the direction cosine

Classic Monte Carlo

Uses random sampling to determine how a sequence of events takes place. Can be used to solve the transport equation:

- Basic Algorithm:
  1. Source term, $Q(x)$ to determine particle starting locations, initial particle weight, direction and distance travelled.
  2. Determine whether particle is scattered or absorbed. If scattered, return to Step (2). Otherwise, end particle history.
  3. After particle simulation, tally angular and scalar flux.
- Disadvantages:
  May need billions of particles to converge, contain statistical noise, scattering of particles introduces thread divergence on GPUs.

Quasi Diffusion Accelerated Monte Carlo (QDA-MC)

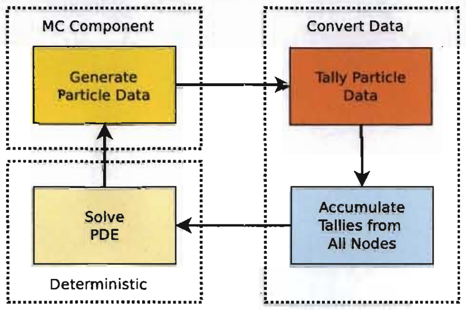
A novel iterative algorithm for the transport equation.

$$\mu\frac{\partial\psi^{HO}}{\partial x} + \Sigma_t\psi^{HO} = \frac{1}{2}\left(\Sigma_s\phi^{LO} + Q\right) \qquad \text{(HO)}$$

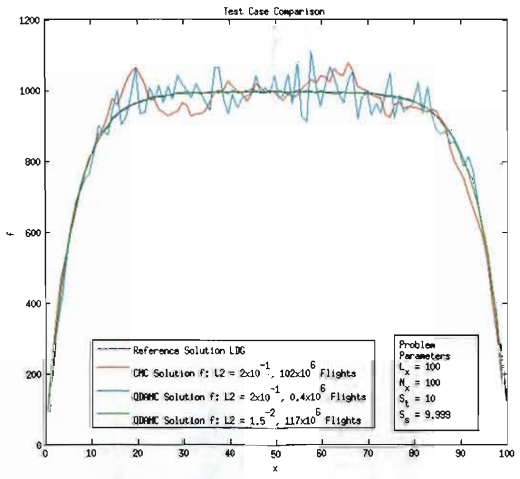
$$\frac{\partial}{\partial x}\frac{-1}{\Sigma_t}\frac{\partial E^{HO}\phi^{LO}}{\partial x} + (\Sigma_t - \Sigma_s)\,\phi^{LO} = Q \qquad \text{(LO)}$$

- Higher Order (HO): Stochastic solution in a purely absorbing medium
- Lower Order (LO): Deterministic approach to handle the scattering

QDA-MC Minimizes thread-divergence.



Test Case



Test Results: $\Sigma_t = 10$, $X = 100$						
	$\Sigma_s = 9.9$		$\Sigma_s = 9.99$		$\Sigma_s = 9.999$	
	CMC	QDAMC	CMC	QDAMC	CMC	QDAMC
CPU Time	75.65	1.71	358.2	1.44	843	1.53
Flights	3.908e6	5.5e5	19e6	5e5	43.8e6	5.5e5
Collisions	98.7	1	973.6	1	8945.9	1
Iterations	396	11	196	10	49	11

Note: For CMC we use 100 particles per cycle and for QDAMC we use 50,000 particles per cycle.

Multicore CPU

- High frequency, deep cache hierarchies, large but slow main memory.
- Run parallel processes for generation/tally of particles, reduce tallies at the end.
- Implemented both MPI and OpenMP version for distributed and shared memory systems respectively.

OPENMP and MPI Results

GPU Optimizations

- QDAMC has massive thread-level parallelism and minimal thread divergence. Well suited for GPUs
- Particle generation and tallying on the GPU
- Lower order system solved on the CPU
- The tallies done in "order" to reduce synchronization costs

GPU RESULTS

Conclusion

- The algorithmic change makes the Monte-Carlo step of QDAMC less thread divergent
- Good scalability seen with OpenMP and MPI
- Reduced thread divergence on GPUs, but synchronization costs for tallying still high
- For problems in higher dimensions
  - Complexity of Lower order system would increase. Might be more efficient to parallelize this step.
  - A hybrid OpenMP-MPI system might be more efficient in terms of memory footprint for larger problems
  - For CPU-GPU architectures, need to overlap computation on CPU and GPU for good performance. Further algorithmic changes to be considered.

Bibliography and Acknowledgements