Hirsch-Fye Quantum Monte Carlo Utilizing Graphics Processors

David Poliakoff: Millsaps College, Joseph Caprino: Rochester Institute of Technology

Hirsch-Fye Quantum Monte Carlo

Hirsch-Fye Quantum Monte Carlo (HFQMC) is used to model interacting electron systems, such as the Hubbard Model. It uses a Green function to describe the motion of electrons through space and time. HFQMC works with the Green function by evolving it through a Markov Chain Monte Carlo process. While most other Monte Carlo algorithms spend much of their time proposing moves, HFQMC traditionally spends the majority of its time updating the Green function after accepting proposed moves. Our goal this summer was to improve existing HFQMC code by extending it to effectively use graphics processing units.

The Cycling HFQMC Markov Process (MP)

Proposal and Acceptance Process (PA): computationally cheap

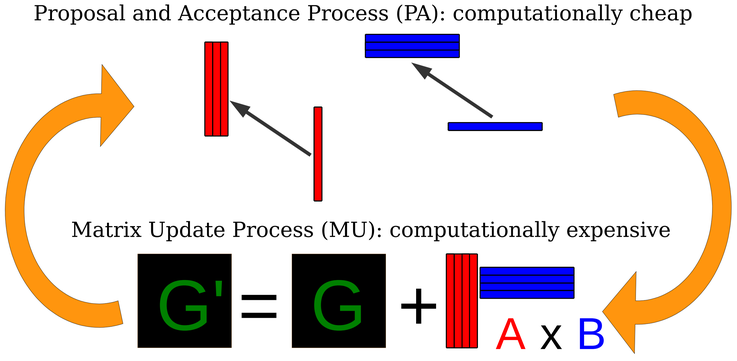

Fig. 1: How the HFQMC Markov process evolves the Green function, G. Note that A and B should be at least 32 wide to increase data reuse.

The Graphics Processing Unit

- Originally designed for computer graphics
- Now more capable of other types of computations
- Can do more floating point operations per second (FLOPS) than the CPU
- Require a programming paradigm which differs from typical CPU programming
- Uses are limited and specialized

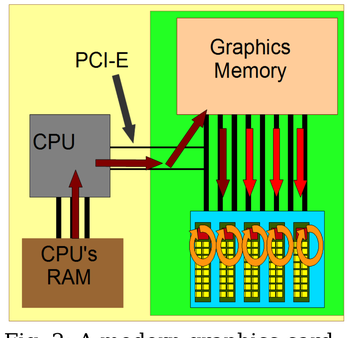

Fig. 2: A modern graphics card in a computer system

Materials & Methods

- Hardware: Nvidia GTX460 1GB, Nvidia M2070
- Programming Languages: CUDA, FORTRAN 90
- Other Software & Techniques: HDF5, PGI Accelerator, CuBLAS Streams

Results & Conclusions

- Started to develop code to read/write data between modules of code
- Found PGI Accelerator is not optimal for the development of this code
- Found CuBLAS streams had severe issues with scalability
- Determined, and have yet to implement, an algorithm for effective use of the capabilities of Nvidia GPUs

Fig. 3: Right are several independent Markov processes (MPs). This could suggest an execution model of each MP having its own completely independent *computer* process on the GPU. However, by testing with other code, many independent processes running on the GPU were found to cause a decrease in performance.

It is not efficient to have different Markov Chains performing different operations simultaneously. We conclude that we must partition proposals and updates such that the two steps do not happen concurrently. This can be achieved by first proposing and accepting changes to all MPs at the same time. This step can be repeated until the associated updates are sufficient to use most of the GPU resources. Only then will the updates be done, and done simultaneously. By cycling between these two mutually exclusive steps, we avoid the problem of concurrent proposals and updates. See figure 4 for a graphical description.

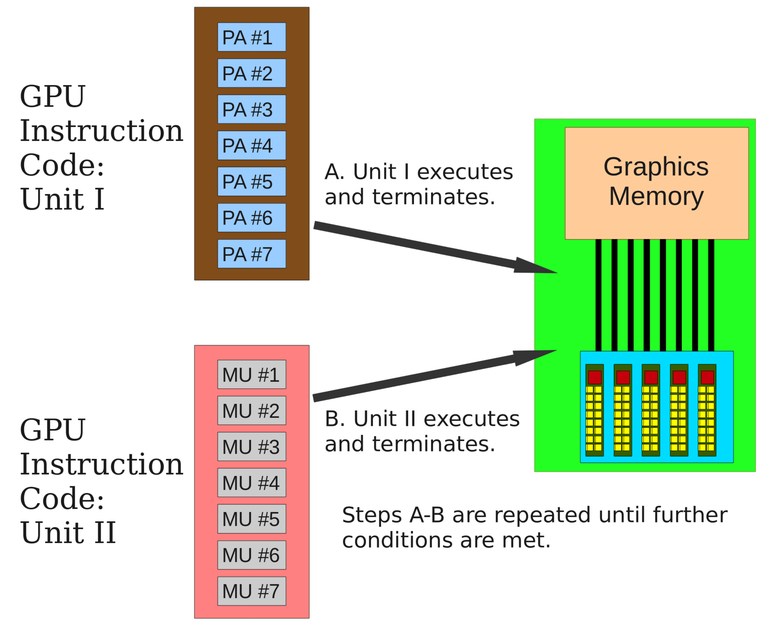

Fig. 4: A solution for the order of kernel execution of the Markov processes on the GPU

Future Work

- Implementing techniques for Hirsch-Fye GPU code
- Optimizing HFQMC GPU code
- Writing thin rectangular matrix-matrix multiply for GPU
- Assessing benefit of offloading more of program to GPU
- Profiling code to see where it spends its time

Acknowledgments

NSF grant OCI-1005165 and the Louisiana State University Center for Computation and Technology

References

CUDA Fortran Programming Guide and Reference, V. 11.4. STMicroelectronics, Inc. (2011).

NVIDIA CUDA C Programming Guide, V. 4.0. NVIDIA (2011).

E. Gull, A. Millis, and A. Lichtenstein, A. Rubtsov, M. Troyer, and P. Werner, Rev. Mod. Phys. 83, 349 (2011).

A. Georges, G. Kotliar, W. Krauth, and M. Rozenberg, Rev. Mod. Phys. 68, 13 (1996).

K. Haule. “Quantum Monte Carlo”. Rutgers. (2006). <http://www.physics.rutgers.edu/grad/509/qmc.pdf>

Contact

David Poliakoff: david.poliakoff@gmail.com
Joseph Caprino: jgc6224@rit.edu