CUDA Implementation of Monte Carlo Based Applications in Computational Chemistry

Akila Gothandaraman[1,3], Revati Kumar[2], Dongliang Yang[2], Kenneth D. Jordan[1,2], Gregory D. Peterson[3]

[1]Center for Simulation and Modeling (SAM), University of Pittsburgh

[2]Department of Chemistry, University of Pittsburgh

[3]Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville

Abstract

Architectures such as multi-core processors, Reconfigurable Computing (RC) using Field-Programmable Gate Arrays (FPGAs), and Graphics Processing Units (GPUs) have emerged as alternatives in the landscape of high performance computing (HPC). In our research, we are particularly interested in leveraging GPUs to accelerate **classical and quantum Monte Carlo** simulations in **computational chemistry**. GPUs have undergone a tremendous growth due to the ever-growing performance demands from the gaming industry. **NVIDIA's GPU solutions** presently provide **hundreds of processing cores** and **tremendous on-chip memory bandwidth** making them attractive for **general-purpose computing**. The **Compute Unified Device Architecture (CUDA)** programming paradigm allows us to exploit the computational power of the GPU without the need to invoke graphics functions, alleviating the difficulty of programming GPUs for general-purpose computing.

Algorithm

Monte Carlo methods involve sampling a number of configurations of a collection of particles and averaging the properties over a large number of samples. With increased computational resources, one can simulate larger physical systems for longer times.

We use the Quantum Monte Carlo (Variational Monte Carlo) to obtain the structural and energetic properties a cluster of inert gas atoms

Algorithm

REPEAT (for *n* iterations)

Step 1: Select a reference configuration, *R* (*x*, *y*, *z*) at random.

Step 2: Obtain a new configuration, *R'* by adding a small random displacement to all particles in the above configuration.

Step 3: **Compute the ground-state properties (potential energy, wave function) of the particles in the current configuration, *R'*.**

Step 4: Accept or reject the present configuration using the ratio of the wave function values.

UNTIL finished

Implementation Strategy: Identify the computationally intensive kernels of the algorithm – **O(N^2) potential energy and wave function calculation**. **We use pair-wise (two-body) model** for calculations of potential energy and wave function.

Classical Monte Carlo method is used to study water, from clusters to the condensed phase

Our project involves the implementation of an efficient algorithm to perform Monte Carlo simulations on the GPUs in order to study the phase diagram of water with the DPP many-body force field developed in our group [1]. As a preliminary step we are developing a Monte Carlo implementation of the simple TIP4P water model [2] on the GPU platform.

Algorithm

REPEAT (for *n* iterations)

Step 1: Select a reference configuration, *R* (*x*, *y*, *z*).

Step 2: Compute the energy for this configuration.

Step 3: Perturb the configuration in Step 1 with a random displacement (translational and rotational) to obtain a new configuration, *R'* and repeat Step 2.

Step 4: If the energy difference between *R* and *R'* satisfies a certain criterion, we accept the configuration and energy in Step 3.

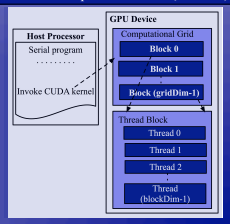
UNTIL finished

We wish to extend this to non-pairwise additive systems where the energy difference calculations involve calculating the total potential energy before and after the perturbation (DPP water model).

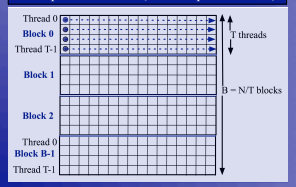
Implementation

Implementation of the Monte Carlo Applications on NVIDIA GPUs using Compute Unified Device Architecture (CUDA)

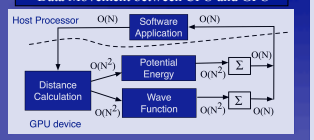
CUDA Computational Grid (1D Grid)



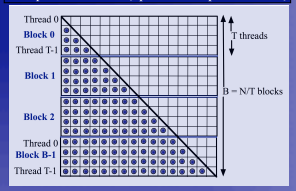
Computational Grid (naïve implementation)



Data Movement between CPU and GPU



Computational Grid (optimized implementation)



Quantum Monte Carlo (QMC)

- O(*N*) atom positions copied from host memory to device memory using CUDA runtime
- **Execution Configuration:** 1D grid of blocks and block of threads
- Subset of rows of the matrix (*N*/*T*) to each thread block
  - Each thread calculates the interactions between its atom and all other atoms
- Different levels of memory hierarchy on the GPU (16 KB shared memory per block)
  - A thread block loads positions of *T* atoms from device memory to shared memory
- In-place reductions of row-wise function values on GPU to produce O(*N*) results
- Final reductions on the CPU to produce O(*1*) potential energy or wave function
- We consider **naïve** (entire matrix evaluation) and **optimized** (upper/lower triangular matrix evaluation) implementations as well as experiment with entirely **single-**, entirely **double-** and **mixed- precision** (single-precision for function evaluations and double-precision for accumulations) algorithms.

Classical Monte Carlo

The CUDA kernel uses the same grid setup as the QMC with the following differences,

- Different number of floating-point operations in the kernel
- We compute the total Lennard-Jones and electrostatic forces from a cluster of water molecules

Results

Quantum Monte Carlo (QMC) Application

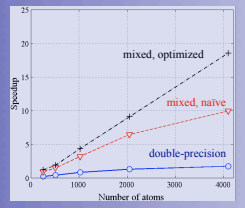
Baseline CPU:

- Eight core Intel Xeon Clovertown 2.66 GHz
- C++ software implementation (double-precision,), Intel MKL provided marginal speedup

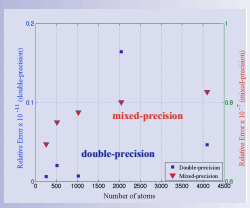
GPU Implementation:

- NVIDIA Tesla C1060 GPU (240 processing cores)
- CUDA: Naïve and Optimized implementations (64 threads in each block)
- Single-precision has the best speedup performance with least accuracy. Mixed-precision implementation has the advantage of higher speedup performance and improved accuracy

Speedup for the GPU implementations (compared to double-precision on a multi-core CPU) for QMC



Error Performance (compared to double-precision on the CPU) for QMC



Classical Monte Carlo Application

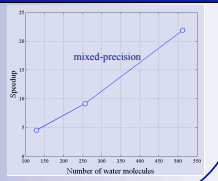
Baseline CPU:

- One core of the Intel Xeon X5570 2.93 GHz processor
- C software implementation (double-precision)

GPU Implementation:

- NVIDIA Tesla C1060 GPU
- CUDA: Mixed-precision (16, threads in each block, single-precision for function (Lennard Jones and electrostatic forces) evaluations and double-precision for accumulations) using TIP4P model

Speedup of the GPU implementation (compared to double-precision on the CPU) for classical Monte Carlo using TIP4P



Conclusions and Future Work

This work explores NVIDIA GPUs platforms to accelerate the kernels of quantum and classical Monte Carlo applications.

Future work includes the following:

- Optimize the GPU implementation of the kernels of the classical Monte Carlo algorithm to study water
- Experiment with the numerical precisions and choose the algorithm that provides the best performance on the GPU (both speedup and accuracy)
- Extending the GPU implementation from the simple TIP4P model to the DPP many-body force field

This work was partially supported by the University of Tennessee Computational Science Initiative and the National Science Foundation grant CHE – 0625598 .

References

[1] A. Defusco, D. P. Schofield and K. D. Jordan, "Comparison of models with distributed polarizable sites for describing water clusters," *Molecular Physics*, vol. 105, pp. 2681-2696, 2007.

[2] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, "Comparison of simple potential functions for simulating liquid water," *Journal of Chemical Physics*, vol. 79, pp. 926-935, 1983.