Acceleration of Quantum Monte Carlo simulation using GPU

Tomoharu Terashima (0810040)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 9, 2010

Keywords: QMC, FMO, GPGPU, CUDA.

It is an interesting challenge to treat biomolecules by ab initio electronic structure calculations. Whole system is, however, too large to be dealt with by full quantum mechanical manner. Fragment Molecular Orbital (FMO) method enables us to handle such problems in practical calculational costs: The whole system is devided into several dense sub-systems, called as fragments, only in which electrons are treated fully by quautum mechanics. Interactions from other fragments are replaced into classical electrostatic fields formed by charge densities of fragments. Size of the simulation gets much reduced to that of subsystems, requiring memory and file capacities in reasonable extent. While molecular Orbital (MO) methods and Density Functional Theory (DFT) are commonly used to evaluate quantum dynamics, the application of Quantum Monte Carlo (QMC) method, instead, is expected to be powerful to get more reliable estimation of the electronic correlation which is believed to play important roles in biomolecules. In such a framework, QMC combined with FMO (FMO-QMC), the additional task to evaluate electrostatic fields at each Monte Carlo step causes considerable speed-down by around 50 times larger CPU time than that of normal QMC.

GPU (Graphical Processing Unit) has attracting performance to accelerate computation in reasonable price. Furthermore, the recent appearance of CUDA (Compute Unified Device Architecture) enables us to develop the

code with more portability. In this study, the FMO-QMC code is modified and combined with CUDA part running on GPU, applied to the FMO calculation of glycine trimer. The performance in CPU time and accuracy is evaluated and compared with those obtained by normal CPU calculations.

We developed CUDA code overriding the original subroutine to evaluate the electrostatic field (Hartree field), which is dominating the CPU time. In the code, registers to store quantities are carefully chosen to optimize the performance. Our implementation is proved to achieve 9.6 (18.3) times faster acceleration in double (single) precision calculation than that without GPU on single core basis. Even with four-fold multicore CPU calculation, it is compared to give 2.46 (4.66) times faster performance. The energy difference caused by GPU is found to be within $10^{-12}(10^{-5})$ at most. The achieved performance well coincides with the theoretical limit of the acceleration, the ratio of 84.2 GFlops for GPU (GeForce GTX275/double precision) to 44.8 GFlops for CPU (Intel Core i7 920).

Getting the performance near to the theoretical limit we expect our way to implement to be more accelerated by GPU with higher specs appearing near future. Idling CPU processors would be another resource for more acceleration: in the present case one node mounts only one GPU while it contains four processor cores (at most three GPUs can be mounted on a node by the current technical status), giving several idling cores during the GPU is working. There are many other potions of the code left which can be processed independently from those calculated by GPU, using parallelization such as Open-MP. Futher improved coding taking such ideas into account would achieve more efficiency.