Quantum Monte Carlo algorithms: making most of large-scale multi/many-core clusters

Jeongnim Kim[1], Kenneth P. Esler[1], Jeremy McMinis[2], and David M. Ceperley[1,2]

National Center for Supercomputing Applications[1] and Department of Physics[2]
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract. With advances in algorithms and the changing landscape of high performance computers (HPC), the quantum Monte Carlo method has become a leading contender for high accuracy calculations for the electronic structure of realistic systems. QMC, being statistical, is naturally scalable to a large number of processors. We discuss QMC implementations to overcome the important efficiency and scalability bottlenecks encountered with the HPC systems based on the multi/many-core architecture of today and present state-of-art QMC calculations of solid-state and molecular systems using tens or hundred thousand cores on the petascale computers. Also presented are the solutions for QMC to adapt to the future HPC architectures and to harness ever-increasing computing powers to tackle outstanding materials and chemical problems.

1. Introduction

Continuum quantum Monte Carlo (QMC) methods employ explicitly correlated wavefunctions to describe the many-body effects and symmetries in an efficient and compact manner and solve the Schrödinger equation by a stochastic process [1]. QMC is one of the most accurate *ab initio* methods to describe the behavior and properties of quantum many-particle systems. It is general and applicable to a wide range of physical and chemical systems in any dimension, boundary conditions etc. The favorable scaling, 3-4 power of the problem size, and ample opportunities of parallelizations, e.g., over multiple Bloch **k**-vectors, make QMC methods uniquely powerful tools to study the electronic structure of realistic systems on large-scale parallel computers.

Until recently, QMC methods have been able to exploit the steady increase in clock rates and enhancement of performance enabled by advances in CMOS VLSI technology for faster and bigger simulations [2]. However, we have seen some significant changes in the mode of increase of computational power in this decade. The performance gain has been largely driven by increasing parallelism in a shared-memory processor (SMP) unit in the form of multi-core processors and GPUs (Graphics Processing Units). Emerging architectures will have an order of magnitude higher concurrency and will force applications to deal with the heterogeneity and hierarchy in the memory, processing and communication subsystems [3]. This presents challenges as well as opportunities for QMC algorithms.

In this article, we discuss the design and implementation details of QMCPACK to take advantage of clusters of multi-core processors and GPUs. QMCPACK, an open-source QMC simulation code written in C++, implements advanced QMC algorithms for large-scale parallel computers [4]. It utilizes hybrid (OpenMP,CUDA)/MPI approaches on multi/many-core

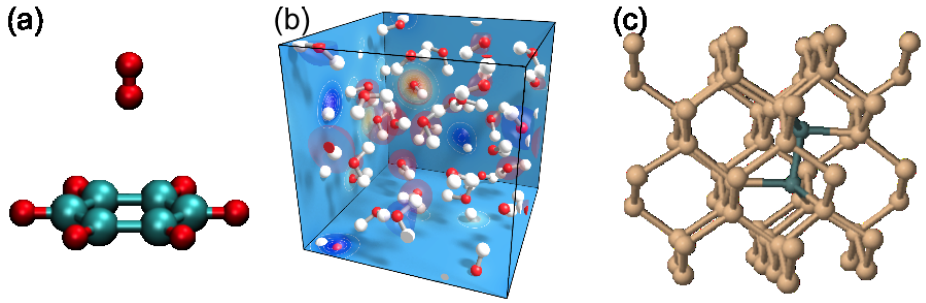

Figure 1. (a) Binding energy of H_2 to a benzene, (b) free-energy of liquid water and (c) energetics of native defects in metals and semiconductors.

architectures to achieve the high computational efficiency, while minimizing communication overhead. We present an overview of applications enabled by QMCPACK and conclude with the outlook of the QMC algorithms for the current and next generations of HPC systems and our plans for emerging computing architectures.

QMC Methods

Various quantum Monte Carlo methods solve the time independent Schrödinger equation whose solution, a many-body wavefunction $\Psi(\mathbf{R})$ in a 3N configuration space $\mathbf{R}$, satisfies an eigenvalue equation of the form,

$$\hat{\mathcal{H}}\Psi = E\Psi. \tag{1}$$

Here E is the ground-state energy for the system and $\hat{\mathcal{H}}$ is the *Hamiltonian*, or total energy operator. Since the exact solution Ψ is known only for simple few-body systems, QMC methods employ a *trial* wave function Ψ_T and find the ground-state solution either by minimizing the energy by varying parameters in a Ψ_T as in variational Monte Carlo (VMC) method or by projecting out the ground state by repeatedly applying an imaginary time Green's function or *propagator* $\exp(-\tau\mathcal{H})$ on Ψ_T as in diffusion Monte Carlo (DMC) method [1].

The vast majority of the computational time in an accurate QMC simulation is spent in DMC and therefore minimizing the wall-clock time to obtain a DMC solution within a target error is desired. The optimized computational kernels, physical abstractions and parallelization schemes, however, are also applicable to all other QMC algorithms. We will use DMC to introduce the key concepts and components as implemented in QMCPACK and to guide readers throughout this paper.

In the DMC algorithm, an ensemble of walkers (*population*) is propagated stochastically from generation to generation, where each walker is represented by $\mathbf{R}$. In each propagation step, the walkers are moved through position space by a *drift-diffusion process*. At the end of each step, each walker may reproduce itself, be killed, or remain unchanged (*branching process*), depending on the value of the local energy at that point in space, $E_L(\mathbf{R}) = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$. The walkers in an ensemble are tightly coupled through a trial energy E_T and a feedback mechanism to keep the average population at the target 10^3-10^5 walkers. This leads to a fluctuating population and necessitates shuffling of walkers among parallel processing units to maintain good load balance in an appropriate interval.

QMC Applications

Figure. 1 highlights recent QMC calculations using the petascale systems at National Center for Computational Sciences (NCCS) and Lincoln GPU cluster at NCSA. Highly scalable and

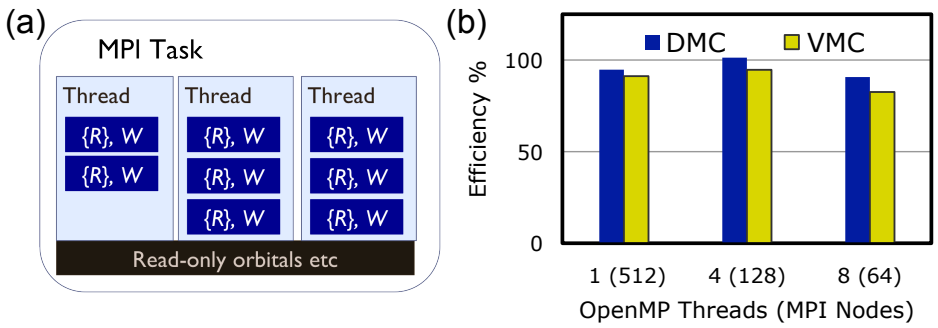

Figure 2. (a) Schematic view of `Walker` allocation per MPI task. Each `Walker` object, shown as a box, contains **R**, weight and other data associated with it. (b) Parallel efficiency of OpenMP/MPI hybrid runs on a Cray XT4 (dual quad-core AMD Opteron) with respect to a run using a single core among 8 available cores per node.

efficient implementations in QMCPACK have allowed QMC studies of the electronic structure of realistic systems with great technological impacts, e.g., on energy, biological and microprocessor development.

2. Hybrid QMC algorithms

Among many ways to parallelize DMC algorithm, the multiplicity of the walkers in an ensemble provides the most natural units for data and task parallelizations. The internal state of each walker, its configuration **R** and other data to reduce recomputing, is encapsulated in `Walker` class. The operations to propagate each walker during the drift-diffusion process are expressed as a parallel loop over the `Walker`s. Once a generation has evolved, the properties of all the walkers in an ensemble are collected to determine E_T and N_w, the number of walkers of the next generation, which employs global reduction operations among MPI tasks. The redistribution of `Walker`s during the load-balance step can be efficiently done by exchanging a serialized `Walker` object in a large message between paired MPI tasks.

We fully exploit the language features of C++ and standard libraries, e.g., allocators, to maximize cache reuse, eliminate false sharing among threads, reduce the overhead in managing threads and optimize MPI communications. There are several different and independent ways to parallelize a QMC calculation beyond the walker level, e.g., over parameter and configuration spaces and over Bloch **k**-vectors. Such high-level parallelism can be easily managed by mapping a DMC ensemble on a MPI group and is not the subject of this article.

QMC on clusters of multi-core SMPs

Figure 2(a) illustrates the distribution of `Walker`s among OpenMP threads and MPI tasks. This hybrid OpenMP/MPI scheme has several advantages over the standard MPI-only scheme.

- Memory optimized: large read-only data to store one-body orbitals and other shared properties to represent the trial wave function and many-body Hamiltonian can be shared among threads, which reduces the memory footprint of a large-scale problem.
- Cache optimized: the data associated with an active `Walker` are in cache during the compute-intensive drift-diffusion process and the operations on an `Walker` are optimized for cache reuse. Thread-local objects are used to ensure the data affinity to a thread.
- Load balanced: `Walker`s in an ensemble are evenly distributed among threads and MPI tasks. The two-level parallelism reduces the population imbalance among MPI tasks and

reduces the number of point-to-point communications of large messages (serialized objects) for the `Walker` exchange.

- Communication optimized: the communication overhead, especially for the collective operations necessary to determine E_T and measure the properties of an ensemble, is significantly lowered by using less MPI tasks.

Effectiveness of the hybrid scheme is evident in Fig 2(b). The parallel efficiency remains high for any combination of OpenMP threads and MPI tasks even at this modest scale of 512 cores. In fact, DMC scales nearly perfectly with respect to the number of threads: the additional parallelism from multithreading allows more walkers per MPI task and improves the overall parallel efficiency and load balance among SMP nodes. The VMC efficiency reflects the memory bandwidth-limited nature of QMC algorithms when all the cores are used. However, the performance reduction due to resource sharing on a node, e.g., bandwidth, is insignificant and the added parallelism with more cores improves the efficiency of a QMC simulation by increasing the rate of MC sample generation and, therefore, reducing the time to solution to reach the target error bars of the quantities of interest. The parallel efficiency of hybrid runs is affected by the memory architecture and is subject to the quality of compilers and MPI implementations. In general, the optimal performance is obtained when a MPI task is mapped over a NUMA node as shown in Fig. 2(b) with 4 threads on a quad-core processor.

QMC on clusters of GPUs
Recently, a path to significant jump in computational performance has become available through the use of GPUs for general-purpose computation. These processors combine many floating point units and relatively simple execution units with a very wide memory bus to allow dramatically higher peak throughput than conventional CPUs. In order to make use of these capabilities, QMC algorithms are reformulated to increase parallelism in the computational kernels: instead of operating on one `Walker` at a time, which is optimal on cache-based CPUs, we reorder the loops so that each kernel works on all the `Walker`s in parallel and parallelize the additional loops over the particles or orbitals. Accordingly, the data structures are redesigned to make good use of the memory hierarchy of a GPU and capabilities are added to efficiently manage data transfer between host and GPU. Using mixed precision on GT200 GPUs and MPI for intercommunication and load balancing, we observe typical full-application speedups of 10x to 15x relative to quad-core Xeon CPUs alone, while reproducing the double-precision CPU results within statistical error [7].

Performance of hybrid QMC
In Figure 3, we show the calculated speedup for a DMC calculation of a defect in a 64 atom supercell (260 electrons). The parallel efficiency of DMC, which compares the number of MC samples generated in a given wall-clock time, shows near 95% of the ideal speedup up to 216K cores. The scalability of CUDA/MPI is equally high, because the communication overhead can be effectively compensated by increasing the computational load, i.e., by allocating more `Walker`s on a GPU than a CPU. On average, QMCPACK achieves 25% of the peak performance on x86 systems, which amounts to 0.5 PF sustained performance on jaguar-pf for the 216K-core runs. Our scaling studies expose the performance issues with blocking collectives and parallel I/O at large scales. However, several practical solutions exist including increased intervals between the synchronizations, asynchronous I/O and use of non-blocking collectives [6].

3. Conclusions

The quantum Monte Carlo method has exploited the steady increase of computational power over past decades. Increasing parallelism in the form of multi-core SMPs or GPUs is not

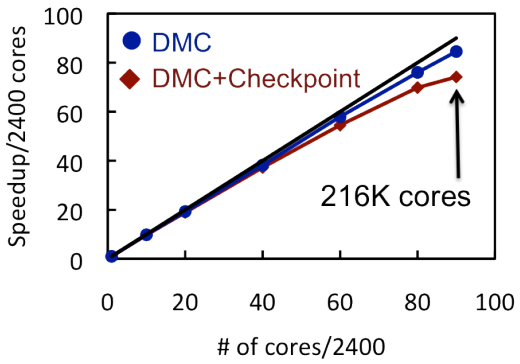

Figure 3. (blue) DMC speedup normalized to 2400 cores. Shown in red is the DMC performance including periodic checkpointing using parallel HDF5 library [5]. The runs use 6 OpenMP threads per MPI task on Cray XT5 (jaguar-pf) systems at NCCS.

an exception. We demonstrated that QMC implementations using (OpenMP,CUDA)/MPI hybrid programming model can achieve high scalability and computational efficiency on the current generation of HPC systems. The barriers for sustained petaflop QMC simulations are successfully removed by the hybrid methods which minimize the memory use and communication overhead, while maximizing the use of available parallel processing units on each platform. New developments are in progress to enhance QMC performance by employing light-weight threads, one-sided communications and mixed precision, which will be increasingly critical on the emerging architectures characterized by limited memory per processing unit, heterogeneity, and high concurrency.

Acknowledgments

This work was supported by the U.S. Department of Energy (DOE) under contract No. DOE-DE-FG05-08OR23336 and the U.S. National Science Foundation (NSF) under contract No. 0904572. We used the resources of the National Center for Computational Sciences and the Center for Nanophase Materials Sciences, which are sponsored by the respective facilities divisions of the offices of Advanced Scientific Computing Research and Basic Energy Sciences of DOE under Contract No. DE-AC05-00OR22725. Also utilized are the resources at NCSA and National Institute for Computational Sciences through the NSF Teragrid program under TG-MCA93S030 and TG-MCA07S016.

References

[1] Foulkes W M C, Mitas L, Needs R J and Rajagopal G 2001 *Rev. Mod. Phys.* **73** 33–83
[2] Esler K P, Kim J, Ceperley D M, Purwanto W, Walter E J, Krakauer H, Zhang S, Kent P R C, Hennig R G, Umrigar C, Bajdich M, Kolorenc J, Mitas L and Srinivasan A 2008 *Journal of Physics: Conference Series* **125** 012057 (15pp)
[3] Dongarra J, Beckman P, Aerts P, Cappello F, Lippert T, Matsuoka S, Messina P, Moore T, Stevens R, Trefethen A and Valero M 2009 *Int. J. High Perform. Comput. Appl.* **23** 309–322 ISSN 1094-3420
[4] Jeongnim Kim, K Esler, J McMinis, B Clark, J Gergely, S Chiesa, K Delaney, J Vincent and D Ceperley QMCPACK simulation suite URL `http://qmcpack.cmscc.org`
[5] Esler K, Kim J, Shulenburger L and Ceperley D M 2010 Fully accelerating quantum monte carlo simulations of real materials on gpu clusters submitted to Computing in science and engineering
[6] Hdf (hierarchical data format) URL `http://hdf.ncsa.uiuc,edu/HDF5`
[7] Hoefler T, Lumsdaine A and Rehm W 2007 *Proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07* (IEEE Computer Society/ACM)