This article has been accepted for publication in Computing in Science and Engineering but has not yet been fully edited.
Some content may change prior to final publication.

Fully accelerating quantum Monte Carlo simulations of real materials on GPU clusters

K. P. Esler,[1] Jeongnim Kim,[1] L. Shulenburger,[2] and D.M. Ceperley[3]

[1] *University of Illinois at Urbana-Champaign, NCSA**
[2] *Geophysical Laboratory, Carnegie Institution of Washington*
[3] *University of Illinois at Urbana-Champaign, NCSA and Dept. of Physics*

(Dated: September 27, 2010)

Continuum quantum Monte Carlo (QMC) has proved to be an invaluable tool for predicting the properties of matter from fundamental principles. By solving the many-body Schrödinger equation through a stochastic projection, it achieves greater accuracy than mean-field methods and much better scalability than quantum chemical methods, enabling scientific discovery across a broad spectrum of disciplines. The multiple forms of parallelism afforded by QMC algorithms make them ideal candidates for acceleration in the many-core paradigm. We present the results of our effort to port the QMCPACK simulation code to the NVIDIA CUDA GPU platform. We restructure the CPU algorithms to express additional parallelism, minimize GPU-CPU communication, and efficiently utilize the GPU memory hierarchy. Using mixed precision on GT200 GPUs and MPI for intercommunication and load balancing, we observe typical full-application speedups of approximately 10x to 15x relative to quad-core Xeon CPUs alone, while reproducing the double-precision CPU results within statistical error.

Introduction

In 1929, Paul Dirac, wittingly or unwittingly, issued a grand challenge that continues to engage the scientific community even today:

> "**The underlying physical laws necessary for a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact applications of these laws lead to equations much too complicated to be soluble.**"

This challenge reflects the understanding that an exact solution of the Dirac equation (or its non-relativistic counterpart, the Schrödinger equation), would allow us the predict all the properties and behavior of matter, at least at terrestrial energy scales. For a system containing N electrons, the Schrödinger equation, which governs the quantum mechanical wave function, Ψ, is a partial differential equation in $3N$-dimensions. Because of this high dimensionality, exact solutions can be found only in extremely simple cases, such in an isolated hydrogen or helium atom. As such, Dirac's challenge has yet to admit defeat. Nevertheless, more than eight decades of development in theoretical physics and chemistry have produced means of finding ever more accurate, albeit approximate, solutions to these equations. This methodological development, coupled with an exponential increase in computing power, has enabled progress from qualitatively correct models based on empirically determined parameters to truly first-principles calculations with significant predictive capability. Such *ab initio* methods take as input only the atomic numbers and positions of the atoms in the molecule or material. These methods complement experiment and analytic theory and have helped drive discovery and advance understanding across a very broad range of disciplines.

Ab initio methods can be categorized in roughly three broad areas. Mean-field methods, including Hartree-Fock (HF), and density-functional theory (DFT), make an approximate mapping of the $3N$-dimensional equation onto a coupled set of N three-dimensional equations. While these methods often yield great insight and reasonable accuracy, there are many outstanding cases in which a mean-field description is quantitatively, or even qualitatively, incorrect. A second class of methods, utilized primarily in quantum chemistry, expand the wave function in a linear basis which is truncated intelligently to capture the important physics. While these can be very accurate, the computation time scales as N^4 or higher, which limits the most accurate calculations to small systems.

The final class, continuum quantum Monte Carlo (QMC) methods, addresses the high dimensionality of the Schrödinger equation by casting it as an integral which can be performed through stochastic sampling. By treating electron correlation in a natural and robust way, QMC has been demonstrated to give results significantly more reliable than those provided by mean-field methods [1]. The computational cost of QMC grows as N^3, the same as DFT, although the prefactor is typically two to three orders of magnitude greater in QMC. Nonetheless, simulations including tens of atoms and hundreds of electrons are now routine, and landmark simulations beyond a thousand electrons have been performed. The need for greater accuracy has led to a quickly increasing adoption of QMC in areas of chemistry, physics, and materials science [2].

During the 80s and 90s, computational methods enjoyed a "free lunch" from advances in silicon process technology. Serial codes could expect a doubling of clock speeds about every eighteen months. Taking advantage of larger installations required parallelizing the code with a message-passing layer such as MPI. While this process was often nontrivial, once this "lunch pass" was purchased, one had access to any number of publicly allocated clusters, which have grown over the years to exceed one petaFLOPS of performance. Since CMOS technology hit the power wall in the last decade, clock speeds have plateaued, constraining the growth in computational power to come almost exclusively through parallelism. In particular, multicore processors have enabled a rapid increase in the core-count of clusters without a concomitant increase in cost.

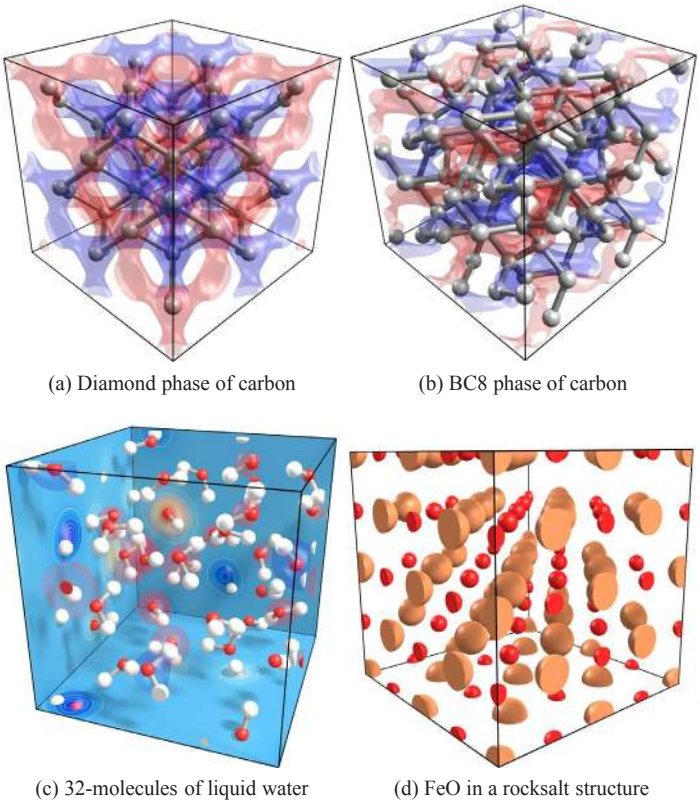
*Electronic address: esler@uiuc.edu

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

This article has been accepted for publication in Computing in Science and Engineering but has not yet been fully edited.
Some content may change prior to final publication.

In recent years, a new computational meal ticket has hit the market, in the form of computing with graphics processing units (GPUs). These processors push beyond multicore to the extreme of many-core, by combining numerous floating point units and relatively simple execution units with a very wide memory bus to allow dramatically higher peak throughput than conventional CPUs. Single GPU performance exceeding 20 CPU cores is not uncommon, and some applications have enjoyed much higher degrees of acceleration. In order to make use of these capabilities, algorithms and data structures must be reorganized to expose parallelism and make good use of the explicit memory hierarchy of a given GPU platform. Work has to be divided between the CPU and GPU, and the transfer of data between the two explicitly managed. As was the case with moving from workstations to clusters, this investment is nontrivial.

The many levels of parallelism afforded by QMC algorithms make QMC intrinsically scalable and ideally suited to take advantage of this many-core architecture. Early work which partially accelerated QMC algorithms on GPUs showed promise [3]. Other electronic structure methods, including HF and DFT, have also benefited from GPU acceleration [4]. As such, we have elected to make the investment, and have ported a large fraction of our QMC code to the NVIDIA's CUDA GPU platform. No one can yet be certain whether the long-term payoff for moving to this new computing paradigm will be as large as it was with distributed memory clusters, but our hopes remain high.



(a) Diamond phase of carbon

(b) BC8 phase of carbon

(c) 32-molecules of liquid water

(d) FeO in a rocksalt structure

FIG. 1: Visualizations of atomic structures and orbitals from systems under study with QMCPACK. Red and blue transparent surfaces are positive and negative isosurfaces of example orbitals.

QMC Methods

Many methods fall under the broad heading of quantum Monte Carlo. We consider the two continuum methods most widely used: variational Monte Carlo (VMC) and diffusion Monte Carlo (DMC). These methods are stochastic means to solve the time-independent Schrödinger equation, an eigenvalue equation of the form, $\hat{\mathcal{H}}\Psi = E\Psi$, where $\hat{\mathcal{H}}$ is the *Hamiltonian*, or total energy operator. The Hamiltonian depends on the type of system studied and describes the interactions in the system. In this work, we consider a system of electrons and ionized atomic cores, so that

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m}\nabla^2 + \sum_{i<j\in\text{elecs}} \frac{e^2}{r_{ij}} + \sum_{I\in\text{ions}} \hat{V}_I^{\text{NL}}, \qquad (1)$$

where the first term is the kinetic energy, the second is the Coulomb repulsion between electrons, and $\hat{V}_I^{\text{NL}}$ is a nonlocal *pseudopotential* operator[1]. For a system containing N electrons, we use the notation that $\mathbf{R}$ is a $3N$-dimensional vector representing the positions of the electrons.

In variational Monte Carlo, an approximation for the lowest-energy eigenstate is found by 1) parametrizing a trial wave function, $\Psi_T(\mathbf{R})$; 2) generating electron positions, $\mathbf{R_n}$, sampled stochastically from the probability distribution $|\Psi_T(\mathbf{R})|^2$; 3) finding the parameters that minimize the expectation value of the energy or variance over the set of samples. This yields the best wavefunction consistent with the parametrization. Diffusion Monte Carlo begins with an optimized trial function, Ψ_T, and projects out the ground state by repeated application of an imaginary time Green's function or *propagator*. This propagator is applied stochastically by a combination of a *drift-diffusion* process followed by a *branching process*. The drift-diffusion is akin to a force-bias Monte Carlo for classical simulations.

Example Applications

In the initial development of our GPU implementation, we have focused on the simulation of real materials and molecules comprised of electrons and ions[5]. This allows an extremely wide class of systems to be studied. We have depicted some example applications in Figure 1. Figures 1(a) and 1(b) show two structural phases of carbon. Diamond is metastable at low pressure, and is believed to be stable until approximately ten million atmospheres at room temperature. We are presently using our GPU code to determine the precise transition pressure, which may be of great practical importance to laser-induced fusion experiments.

Figure 1(c) shows a configuration of 32 molecules of liquid water. Several hundred such configurations were recently simulated using QMCPACK on a Cray-XT5 system, consuming over 30 million CPU hours. We believe our GPU version will allow us to further this research at greatly reduced computational cost. Figure 1(d) shows the mineral wüstite, FeO, in its rocksalt structure. We are using our GPU version of QMCPACK to understand pressure-induced changes in its electronic structure. Understanding such transition metal oxides is closely linked to a broad

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

This article has been accepted for publication in Computing in Science and Engineering but has not yet been fully edited.
Some content may change prior to final publication.

range of phenomena ranging from superconductivity to metal-to-insulator transitions.

Variational Monte Carlo

Variational Monte Carlo (VMC) computes the expectation value of a quantum-mechanical operator, $\hat{\mathcal{O}}$, defined as

$$\left\langle \hat{\mathcal{O}} \right\rangle = \int d^{3N}\mathbf{R}\ |\Psi_T(\mathbf{R})|^2 \underbrace{\frac{\hat{\mathcal{O}}\Psi_T(\mathbf{R})}{\Psi_T(\mathbf{R})}}_{\mathcal{O}_L(\mathbf{R})}, \tag{2}$$

where $\mathcal{O}_L(\mathbf{R})$ is termed the *local value* of the operator. When the operator is the Hamiltonian, $\hat{\mathcal{H}}$, the average of the *local energy*, $E_L(R)$, gives the expected value of the total energy. Other observables commonly computed are the potential energy, pair correlation functions, the electronic charge density, and, more recently, the forces on the atoms[6, 7]. To compute this very high-dimensional integral, VMC utilizes the standard Metropolis Monte Carlo method to generate samples from the probability distribution $P(\mathbf{R}) \propto |\Psi_T(\mathbf{R})|^2$.

To sample this distribution, we generate an ensemble of points in $3N$-dimensional space, $\{\mathbf{R}_i\}$, which we call *walkers*. Each walker is propagated by attempting a random move to $\mathbf{R}_i'$, which is accepted or rejected based on the ratio $|\Psi_T(\mathbf{R_i'})|^2 \,/\, |\Psi_T(\mathbf{R_i})|^2$. In practice, highest efficiency is usually achieved when only a single electron is moved at a time. After the walkers have reached their equilibrium distribution, the expected value of observable properties can be computed as an average of the local values $\mathcal{O}_L(\mathbf{R}_i^n)$ taken over all walkers, i, and all Monte Carlo steps, n.

We write the trial wave function in an analytic form with adjustable parameters, p_j, so that $\Psi_T(\mathbf{R}) = \Psi_T(\mathbf{R};\{p_j\})$. It satisfies a *variational principle*, i.e., for any set of parameters $\{p_j\}$, the average energy of $\Psi_T(\mathbf{R};\{p_j\})$ must be greater than or equal to the true ground state energy, E_0. Thus, the parameters may be optimized to minimize the energy $E(\{p_j\})$. Alternatively, we may minimize the variance of the *local energy*, $E_L(\mathbf{R}_i;\{p_j\})$ over the ensemble of samples.

Diffusion Monte Carlo

Diffusion Monte Carlo modifies the stochastic procedure in order to repeatedly apply a Green's function operator that projects out the ground state from the trial wave function. Since smaller steps must be taken, DMC is somewhat more expensive than VMC, but provides much more accurate results.

In practice, the stochastic projection is effected through a *branching* process. After a single-particle move is attempted for each electron in turn, $E_L(\mathbf{R}_i)$ is computed for each walker. Each walker carries a weight, w_n, which is updated as

$$w_i^{n+1} = w_i^n \exp\left\{\tau\left[E_T - \frac{E_L(\mathbf{R}_i^{n+1}) + E_L(\mathbf{R}_i^n)}{2}\right]\right\}, \tag{3}$$

where the superscripts label Monte Carlo generations, and τ is an adjustable time step which must be chosen small enough to avoid error. After the weights are assigned, walkers are stochastically replicated or destroyed so that the final number of copies of each is proportional to its weight. The *trial energy*, E_T, is adjusted through a slow feedback process to keep the population size centered around a target.

Algorithm 1 Pseudocode for diffusion Monte Carlo, with loop ordering optimized for CPUs.

```
for generation = 1 ··· N_MC do
  for walker = 1 ··· N_w do
    let R = {r_1 ... r_N}
    for particle i = 1 ··· N do
      set r_i' = r_i + δ
      let R' = {r_1 ... r_i' ... r_N}
      ratio ρ = Ψ_T(R')/Ψ_T(R)
      if r → r' is accepted then
        update inverse matrix, distance tables, etc.
      end if
    end for{particle}
    Compute local energy, E_L = ĤΨ_T(R)/Ψ_T(R)
      Kinetic energy = −(1/2)∇²Ψ_T(R)/Ψ_T(R)
      Electron-electron energy (Coulomb)
      Pseudopotential energy
    Reweight and branch walkers
    Update E_T
    if generation > N_eq then
      Collect properties
    end if
  end for{walker}
end for{generation}
```

For bosons (e.g. ^{4}He atoms), the DMC method is exact (within statistical errors) if the time step, τ, is chosen sufficiently small. However, for fermions, including electrons, the *fixed-node approximation* is introduced to alleviate the *fermion sign problem*. It yields exact results if the nodes of Ψ_T (i.e. the $(3N-1)$-dimensional surface defined by $\Psi_T(\mathbf{R}) = 0$) are exact. With approximate nodes, the error in the energy is quadratic in the error in the nodes. In practice, the error is usually very small if Ψ_T is reasonable. For example, using a trial wave function derived from density functional theory, the accuracy of DMC usually greatly surpasses that of the DFT calculation.

The Trial Wave Function

The most commonly used trial wave function for solid state systems takes the Slater-Jastrow type,

$$\Psi_T = \det\left[\phi_n^{\uparrow}(\mathbf{r}_j^{\uparrow})\right] \det\left[\phi_n^{\downarrow}(\mathbf{r}_j^{\downarrow})\right] e^{J_1(\mathbf{R};\mathbf{I})} e^{J_2(\mathbf{R})}, \tag{4}$$

where ϕ_n are single-particle orbitals, and J_1 and J_2 are one-body and two-body Jastrow correlation factors. Here, $\mathbf{I} = \{\mathbf{i}_1 \ldots \mathbf{i}_M\}$ is the $3M$-dimensional vector giving the coordinates of the nuclei or ions. In the absence of magnetic fields, electrons are assigned a definite spin and the configuration is given as

$$\mathbf{R} = \{\mathbf{r}_1^{\uparrow} \ldots \mathbf{r}_{N^{\uparrow}}^{\uparrow}, \mathbf{r}_1^{\downarrow} \ldots \mathbf{r}_{N^{\downarrow}}^{\downarrow}\}. \tag{5}$$

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

For simplicity, we will omit the spin label for the remainder of our discussion.

In general, it is possible to optimize both the single-particle orbitals and Jastrow correlation functions. In practice, however, only a small number of parameters (typically ~ 20) are needed to describe the Jastrow factors, while very many may be required to describe the orbitals. At present, then, we parametrize and optimize only the Jastrow functions.

Standard CPU implementation

We first describe our reference QMC implementation in QMCPACK [8] and discuss the performance issues on common architectures based on multicore CPUs. QMCPACK, an open-source QMC simulation code written in C++, implements advanced QMC algorithms for large-scale parallel computers. Designed with the modularity afforded by object-oriented architecture, it makes extensive use of template metaprogramming to achieve high computational efficiency through inlined specializations. It utilizes a hybrid OpenMP/MPI approach to parallelization to take advantage of the growing number of cores per SMP node. Finally, it utilizes standard file formats for input and output, including XML and HDF5, to facilitate data exchange.

The main design goal of the computation kernels and physical abstractions of QMCPACK is to minimize the time to complete a generation, a step in the stochastic projection process in DMC as summarized in Algorithm 1. The propagation of the walkers within the walker loop can be carried out independently by distributing N_w walkers over parallel processing units, *i.e.*, MPI nodes and OpenMP threads in QMCPACK. Once a generation has evolved, the properties of all the walkers in an ensemble are collected to determine E_T and N_w of the next generation. This requires a global communication among the parallel units and redistribution of the walkers to keep the load balanced. To initialize the DMC simulation, uncorrelated walker positions are generated with VMC. The ensemble of walkers must then be propagated for many generations (typically 500-1000) to reach the steady-state distribution. The properties collected during this equilibration period are discarded for the final results.

GPU implementation

Several differences between CPU and GPU architecture mandate changes in how algorithms are best implemented in each case. Present multicore CPUs have relatively low memory bandwidth relative to GPUs, but this deficit is partly compensated by the inclusion of a hierarchy of large caches. The working set of data required for a single walker is usually small enough to reside entirely within the L2 or L3 cache of a typical workstation CPU. In this case, it is most efficient to make many moves on a single walker before moving onto the next in order to reuse data in the cache.

In contrast, on GPUs, a very large number of in-flight threads is required to fully utilize all the functional units on the processor and to hide latency. For this reason, it is greatly advantageous to propagate all walkers in parallel, i.e. to interchange the walker and particle loops in Algorithm 1. Since the working data set for a single walker is far too large to fit in the shared memory in present GPUs, the advantage of working on a single walker at a time is also lost. In practice, this loop reordering requires restructuring all the computational kernels.

In typical simulations, only 64 to 256 walkers are assigned to each GPU. If only walker-level parallelism were exploited, an insufficient number of threads would be available to hide latency. Furthermore, the GPUs we employ load and store data to DRAM in 64-byte chunks. Hence, the highest bandwidth utilization is achieved when memory accesses have unit stride. This memory operation *coalescing* would be difficult to achieve if we assigned adjacent threads to distinct walkers. Thus, in each kernel, we exploit additional parallelism, typically loops over electrons, atomic cores, or orbitals. In order for the kernels to operate efficiently, the data structures employed required reorganization to allow coalesced memory access, and to allow efficient use of shared memory. In some cases, functions were broken into two or more kernels to allow for use of less shared memory, different block sizes, and synchronization between blocks.

To the major computational classes in QMCPACK, we have added routines which act on an ensemble of walkers rather than a single walker. These C++ member functions transfer any necessary data to the GPU, call C functions which launch CUDA kernels, and copy back results required by the host CPU. By incorporating the GPU functionality into an existing code, we have been able to perform cross-checks for correctness and accuracy at each stage, essentially providing a facility for built-in unit tests. The stochastic nature of the method, combined with the generality and complexity of QMCPACK, has made this method of debugging and validation invaluable. Identifying one of dozens of kernels responsible for a statistically erroneous result in a stochastic code may otherwise have proven intractable.

Data Structures

For each walker, temporary data associated with each component of the wave function must be stored to avoid recomputation, e.g. the inverse matrices associated with determinant evaluation. To avoid spending excessive time allocating, deallocating, and transferring data in small chunks, we aggregate all data associated with a walker into a single anonymous buffer. In a *pre*-allocation stage, each object which requires per-walker storage makes a storage request for each data member. Each preallocation request returns an offset into the buffer. Requests are padded to 64-byte boundaries to assure coalesced access to global memory. After the preallocation requests, a single buffer is allocated for each walker. This scheme has particular benefits during DMC simulation, in which walkers are dynamically created, destroyed, and migrated between nodes.

Multi-GPU Parallelization

In both the CPU and GPU versions of QMCPACK, MPI is used for parallelization between nodes. In particular, MPI is used to: 1) migrate walker data sets between nodes for load balancing in

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

This article has been accepted for publication in Computing in Science and Engineering but has not yet been fully edited.
Some content may change prior to final publication.

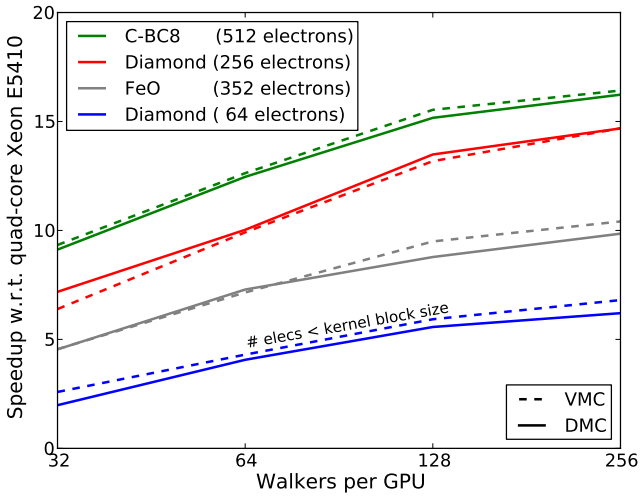


FIG. 2: GPU speedup versus quad-core Xeon E5410 with four threads.

DMC; 2) accumulate statistical averages; 3) update the trial energy, E_T, as DMC progresses; 4) broadcast large input data sets. The dynamic duplication and annihilation of walkers during DMC simulation can result in a load imbalance between nodes. To correct this, we redistribute walkers between nodes with point-to-point transfers after each branching step. To effect this transfer, the GPU-resident walker data must first be copied to host memory. By storing the entire data set in an anonymous buffer, this copy can be accomplished in a single transfer, greatly mitigating overhead. In practice, we have observed very little overhead from the transfers due to load balancing. All other communication (i.e. 2-4) represents either one-time or very small transfers, and do not impose practical limitations on MPI scaling.

Overall Performance and Accuracy

Because the NVIDIA GT200 GPU has a significant gap between single and double-precision peak FLOP rate, we elected to utilize single-precision where possible. After computing properties for each walker in single-precision, results are transferred to CPU memory and ensemble averages are accumulated in double precision. We found that only one GPU kernel, namely the inversion of determinant matrices, required double-precision to retain very high accuracy. At present, however, we have not yet modified the CPU version to take advantage of this property. Thus, in our performance comparison, we compare double precision performance on the CPU to a mixed-precision implementation on the GPU.

Figure 2 gives a plot of the GPU-to-CPU speedup, measured by the ratio of the average time to execute a block of Monte Carlo moves. The vertical axis gives the ratio of execution rates when comparing GT200 GPUs (in Tesla S1070s) to quad-core Xeon E5410 CPUs in dual-socket compute blades. The speedup is truly one GPU to one CPU, with four threads running on each CPU, in contrast to the commonly used comparison to single-core performance.

The relative speedup increases with both the number of electrons and with the number of walkers running on each GPU. This reflects the fact that many in-flight threads are required to fully hide memory access latency. As we discuss below, some further restructuring of our kernels may lower the number of walkers needed to saturate GPU performance. Nonetheless, for large systems in our present implementation, one GT200 can exceed the performance of 15 quad-core Xeon E5410 processors. We note further that our CPU code has been highly optimized, with some critical routines hand-optimized with SSE intrinsics, so that this is not a "straw man" comparison.

If the mostly single-precision GPU implementation does not yield sufficiently accurate results, it is useless. We have conducted numerous tests comparing the results of the CPU and GPU implementations. So far, we have not found any statistically significant discrepancies. Despite the use of single precision, the GPU code appears to be as accurate as the CPU version, at least to the level of statistical error we have achieved thus far.

Main Computational Kernels

We have implemented approximately 100 CUDA kernels (including a number of specializations), which perform essentially all the computation needed to simulate real materials in periodic boundary conditions and molecules in open boundary conditions. Most can be divided into kernels that evaluate wave function ratios and derivatives, and those that evaluate potential energy.

To determine the acceptance probability for single-particle moves, we need to compute the ratio of the trial wave function values at the new and old positions, i.e. $\Psi_T(\mathbf{R}')/\Psi_T(\mathbf{R})$, where $\mathbf{R}$ and $\mathbf{R}'$ differ by the position of one electron. In order to compute the kinetic energy, we also require the quantities $\nabla_i\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$ and $\nabla_i^2\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$. Since the trial wave function is the product of several components, we have expressed the derivatives in terms of logarithms, making the complicated application of the product rule at run time unnecessary.

Single-Particle Orbital Evaluation

To evaluate the determinant part of the wave function, we must first evaluate the single-particle orbitals, a set of functions defined on the 3D domain of our simulation cell. Other *ab initio* methods for solids typically expand the orbitals in plane-waves, $e^{i\mathbf{G}\cdot\mathbf{r}}$, allowing the use of FFTs to efficiently evaluate the orbitals at all points on a regular grid simultaneously. In real-space, however, the number of basis function evaluations required to compute the value of an orbital at a single location grows with system size, and many thousands are needed even for a modest number of atoms.

In QMC, it is more efficient to use a localized basis with compact support. 3D tricubic B-splines provide a basis in which only 64 elements are nonzero at any given point in space[9], allowing the rapid evaluation of each orbital in constant time. Furthermore, the basis can be improved systematically simply by decreasing the spacing parameter. On the downside, the 3D B-splines require a comparatively large amount of memory.

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

This article has been accepted for publication in Computing in Science and Engineering but has not yet been fully edited.
Some content may change prior to final publication.

B-splines are also very amenable to parallel evaluation on GPUs. When we move an electron in QMC, we must compute the values of all the orbitals at the new position. By arranging the B-spline coefficients so that the orbital index runs fastest in memory, we can ensure coallesced reads, minimizing wasted memory bandwidth. The gradient and Laplacian of the orbitals, needed for the kinetic energy, can be evaluated simply by taking derivatives of the basis functions. We have released a separate CPU and GPU library for the construction and evaluation of cubic B-splines in 1D, 2D, and 3D (http://einspline.sf.net).

Determinant Ratios

Consider the Monte Carlo move of the k^{th} electron from $\mathbf{r}_k$ to $\mathbf{r}'_k$. Defining $A_{in} = \phi_n(\mathbf{r}_i)$ and $A'_{in} = \phi_n(\mathbf{r'}_i)$, we need to compute the ratio, $\det(A')/\det(A)$. Since we have moved only electron k, A and A' differ only by row k. By expanding in cofactors, the determinant ratio may be written in terms of the inverse of A as,

$$\frac{\det\left[A'\right]}{\det\left[A\right]} = \sum_n \phi_n(\mathbf{r'}_k)\left[A^{-1}\right]_{nk}. \tag{6}$$

The logarithmic gradient and Laplacian with respect to $\mathbf{r}_k$ of $\det A$ can be written similarly.

Inverses and Updates

Using the ratio formulae given above requires that we store the inverse of A for each walker. To compute the inverse initially, we use a relatively simple in-place Gauss-Jordan inversion with partial pivoting. Since this process involves numerous subtractions, truncation error would accumulate for large systems in single precision, resulting in a bias in the simulation. For this reason, we perform this inversion in double precision.

Since the moves we employ change only one row of the A matrices at a time, we employ a rank-1 update of A^{-1} using the Sherman-Morrison formula. This allows the inverse to be updated in $\mathcal{O}(N^2)$ time rather than $\mathcal{O}(N^3)$ time for a full inversion. In particular, if we replace row k of matrix A with a vector $\mathbf{v}$, then A^{-1} is updated as

$$\left[A + e_k \otimes \delta^T\right]^{-1} = A^{-1} - \frac{\left(A^{-1}e_k\right) \otimes \left(\delta^T A^{-1}\right)}{1 + \delta^T A^{-1} e_k}, \tag{7}$$

where δ is the change in row k of A, and e_k is the vector with a 1 as the k^{th} element and zeros for all others. To implement this update, we divide the operations into two kernels. In the first, the product $u \equiv \delta_T A^{-1}$ is computed. While computing this product, we store the k^{th} column of A^{-1}, i.e. $v \equiv A^{-1}e_k$, contiguously in global memory. This avoids using uncoalesced reads in the second kernel, which adds on the outer product of u and v, scaled by the constant $-1/(1+u_k)$. Since repeated updates in single precision can eventually result in an inaccurate inverse, we occasionally reinvert from scratch using our double-precision Gauss-Jordan routine.

Jastrow Evaluation

Jastrow factors allow the wave function to have explicit correlation between pairs of electrons and between electrons and ions.

$$J_2(\mathbf{R}) = \sum_{i<j} u_{ij}\left(\left|\mathbf{r}_i - \mathbf{r}_j\right|_{\text{min}}\right). \tag{8}$$

The function $u_{ij}(r)$ depends on whether i and j have the same or opposite spin label, and is constructed to vanish for distances beyond a cutoff radius, r_c. Since we perform the simulation in periodic boundary conditions, we utilize the minimum image convention when evaluating all distances. Kernel specializations are used when r_c is less than the inscribed radius of the simulation cell. This permits a much faster determination of the minimum-image distance.

The one-body, electron-ion term is similarly given by

$$J_1(\mathbf{R}) = \sum_{i,j} \chi_j\left(\left|\mathbf{r}_i - \mathbf{I}_j\right|\right), \tag{9}$$

where χ_j depends on the atomic number of ion j. Both u and χ functions are represented with cubic B-splines, whose coefficients have been constrained to give the right behavior at the origin and at r_c. The gradients and Laplacians of J_1 and J_2 are computed analytically in separate kernels.

Only a few B-spline coefficients are required to represent the function, so these may be read into shared memory at the beginning of kernel execution. Since the distances are stochastically generated, however, the index of the coefficient read by each thread is random. If more than one thread attempts to read the same coefficient simultaneously, serialization due to bank conflicts may result if more than one broadcast is required. NVIDIA's next generation architecture, Fermi, will mitigate this by supporting multiple simultaneous broadcasts from shared memory.

Coulomb Interaction

To compute the energy arising from Coulomb interaction between electrons, we must sum over all pairs of electrons. In periodic boundary conditions, the sum must include not only the particles in the simulation cell, but all their periodic images. A naive summation over images does not converge, but a rapidly converging method was given by Ewald, which divides the sum into a real-space part and Fourier-space part. In practice, we use a more efficient breakup which converges even more quickly[10], resulting in two functions, $v_s(r)$ and $v_l(r)$, the latter of which is Fourier transformed to reciprocal space, yielding v_G. $v_s(r)$ is tabulated on a fine grid and used to create a 1D "texture object" which allows linear interpolation with the GPU image-sampling hardware. The long-range part is summed in Fourier-space taking advantage of the fast hardware transcendental instructions. In both cases, we use the fast on-die shared memory to minimize loads from GPU DRAM.

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

This article has been accepted for publication in Computing in Science and Engineering but has not yet been fully edited.
Some content may change prior to final publication.

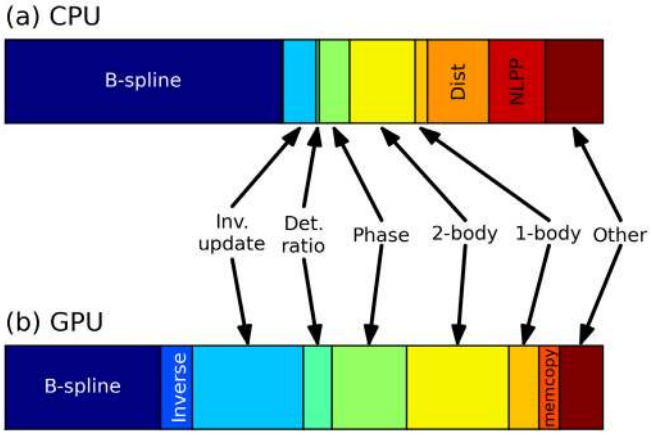


FIG. 3: Percentage of run time consumed by the main kernel categories for QMCPACK running on CPUs and GPUs.

Nonlocal Pseudopotential Evaluation

In many first-principles methods, including DFT, quantum chemistry, and QMC, the core electrons are eliminated and their effect on the valence electrons replaced by a nonlocal potential operator. This operator can be applied by approximating an angular integral by a quadrature on a spherical shell surrounding the atom, as

$$\frac{\left[\hat{V}_{\rm NL}\Psi_T(\mathbf{R})\right]}{\Psi_T(\mathbf{R})} = V_{\rm loc}(r) + \sum_{\ell=0}^{\ell_{\rm max}} \frac{2\ell+1}{4\pi}\left[V_\ell(r) - V_{\rm loc}(r)\right] \times \sum_{\mathbf{r}_i'} P_\ell\left[\cos\left(\theta_i'\right)\right] \frac{\Psi_T(r_1,\ldots,\mathbf{r}_i',\ldots,r_N)}{\Psi_T(r_1,\ldots,\mathbf{r}_i,\ldots,r_N)}. \quad (10)$$

The vast majority of the computation work involved in evaluating this expression is in the wave function ratios in the last term. Beyond a cutoff radius, r_c, $[V_\ell(r) - V_{\rm loc}(r)] = 0$, by construction. The first kernel for this routine simply determines which electrons are within r_c of each ion, and constructs a list of job objects, specifying the ratios which must be computed in the second phase. These jobs are concatenated into a contiguous list on the CPU, since this cannot be done easily in parallel. Finally, the ratios are computed on the GPU and passed back to the CPU, which then performs the summation.

Performance Analysis

It is instructive to consider the execution time of each class of kernels relative to the total execution time, and compare these percentages between the CPU and GPU implementations. Figure 3 shows bar charts representing the proportion of execution time spent in the main categories of kernels when executed on the same physical system. Note that these are percentages, rather than absolute times. Since the GPU speedup is over 10×, the absolute times are considerably smaller on the GPU for all kernels. Significant differences between the breakdowns can be readily identified. Most of these differences can be understood by comparing the difference in memory hierarchy between the GT200 GPUs and Xeon CPUs.

The Harpertown CPUs in this comparison have relatively low memory bandwidth of approximately 10 GB/s per socket in a dual-socket configuration, but feature a large 12 MB L2 cache which greatly mitigates this disadvantage in many cases. In contrast, the Tesla 10 series offers global memory bandwidth exceeding 100 GB/s, but the user-controlled cache (i.e. shared memory) is insufficient to store some larger data sets used in our code. We can identify three classes of kernels, based on the size of their data sets: 1) those with working sets too large for both CPU and GPU cache; 2) those with working sets that fit in the CPU cache, but not the GPU cache; 3) those with small data sets that fit in both CPU and GPU cache.

The most expensive kernels are those that perform the B-spline evaluation of the orbitals. Because this involves reading a series of N coefficients from random locations in a 3D table typically over 1 GB in size, cache is of very little benefit. With only two FLOPs per load, this kernel is strongly limited by bandwidth, and our implementation achieves 75% of the peak bandwidth available on the Tesla 10. As such, B-spline evaluations constitute a smaller percentage of total run time on the GPU.

In the CPU version, matrix inverses are computed fully only once at the beginning of a run and are then updated with each move, so that the inverse contributes a negligible amount to run time. On GPUs, they constitute a small time, though not completely negligible. On CPUs, the inverse matrices fit comfortably in L2 cache on Xeons, so that the updates are relatively fast. In contrast, on GPUs these matrices are much too large for shared memory, so that the kernel is again bandwidth limited and takes a larger percentage of time. A similar observation may be made about the determinant ratios.

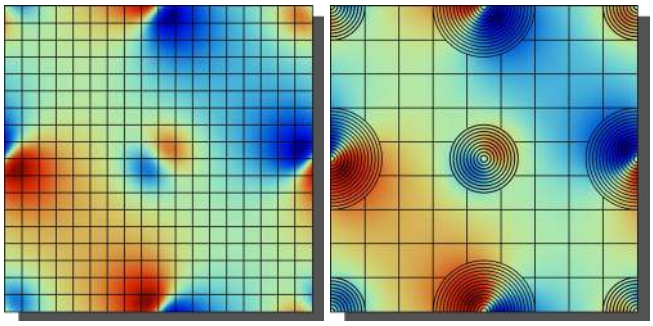
The Jastrow evaluations require only particle positions and B-spline coefficients, both of which can reside in L1 on CPUs or shared memory on GPUs, so that the kernels are compute-limited. In the CPU implementation, distances are computed and tabulated outside the Jastrow evaluation, while they are done inline in the GPU version. Thus, if we combine the times for J1, J2, and Dist., we find that the total percentage of runtime used for Jastrow and distance evaluation is quite similar on the CPU and GPU.

Considering these kernels, we can then make the following general observations about relative run times. When operating on data sets which cannot be contained in cache on either CPU or GPU, the GPU's superior bandwidth leads to the largest observed speedup. On data sets which fit in CPU cache, but not in GPU shared memory, we observed the least speedup. On very small data sets which fit in both CPU cache and GPU shared memory, the speedup is between these two extremes.

Dealing with limited memory

At present, our GPU implementation enables us to perform our typical production simulations with much less hardware. In the broader picture, however, we also desire to use GPUs to extend the range of what is computationally feasible. The application of QMC to real materials has until recently been mostly restricted

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE



(a)Uniform B-spline (b)Mixed basis

FIG. 4: Schematic of uniform B-spline and mixed-basis representation for orbitals. With a uniform B-spline representation, a dense Cartesian mesh is required. In a mixed-basis representation, a much coarser mesh is sufficient between the atoms.

to the bulk properties of perfect crystals. Addressing defects and disordered materials requires the ability to simulate larger systems and achieve higher statistical accuracy. At present, a number of factors limit the size of the systems to which our GPU implementation can be applied.

While computationally efficient, the 3D B-spline basis for the orbitals requires a large amounts of memory, which grows quadratically with system size. Currently, the largest GPU memory buffer available is 4 GB on the Tesla C1060 card, which limits the size of the system we can address. For larger systems, a number of methods can be used to reduce the required storage. In a perfect crystal, symmetry can be used to allow the memory usage to grow only linearly with the number of atoms, allowing simulations with over 500 electrons. In disordered systems, such as liquid water, this type of reduction is not possible and system size is more limited.

To go further, we implement a mixed-basis approach. The mesh spacing required to accurately represent the orbitals is determined by their smallest feature size. The shortest wavelength features are concentrated around the atomic cores, while in the area between the atoms, the functions are smooth. For this reason, we divide space into spherical regions called *muffin tins* surrounding the atoms, and an interstitial region between them. Inside the muffin tins, the orbitals are atomic-like, and can thus be represented accurately and compactly by spherical harmonics, as

$$\phi_{\mathrm{MT}}^{n,j}(\mathbf{r}) = \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} u_{\ell m}^{nj}(|\mathbf{r}-\mathbf{I}_j|) Y_m^{\ell}\left(\frac{\mathbf{r}-\mathbf{I}_j}{|\mathbf{r}-\mathbf{I}_j|}\right). \qquad (11)$$

Since the high-frequency components in the muffin tins have been removed, we can represent the orbitals in the interstitial region by 3D B-splines on a much coarser grid. This is represented schematically in Figure 4. The radial functions, $u_{\ell m}^{nj}(r)$ are represented as 1D Hermite splines. We did not use B-splines as elsewhere, since in single precision, truncation error leads to a poor estimate for the second derivative. Utilizing this dual basis in place of 3D B-splines alone typically allows the same accuracy to be achieved with $5\times$ to $10\times$ less memory, with about the same performance.

Final thoughts

Quantum Monte Carlo simulations have long been, and continue to be, a major consumer of supercomputing power, from vector machines such as the Cray XMP[11], to superclusters such as the Cray XT-5. The accuracy they afford in predicting the properties of a diverse set of real materials justifies this consumption. Nonetheless, we have seen that through the use of GPUs, the same scientific results may be achieved at considerably less expense. Furthermore, as the processors develop, the software is refined, and larger GPU clusters are built, this technology will allow us to progress from perfectly periodic systems to those systems with defects, disordered systems, etc. The speedups afforded by the GPUs may also allow us to proceed from simulations with the atoms frozen in a single configuration to those which couple the dynamics of the electrons with those of the atomic cores[12]. With further progress, perhaps in a few years QMC simulations will be as ubiquitous as DFT calculations are at present.

Acknowledgements

This work was supported by the U.S. Department of Energy (DOE) under Contract No. DOE-DE-FG05-08OR23336 and by the National Science Foundation under No. 0904572 . This research used 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. This research was supported by an allocation of advanced computing resources provided by the National Science Foundation, under TG-MCA93S030 and TG-MCA07S016, and utilized the Abe and Lincoln cluster at the National Center for Supercomputing Applications, as well as Kraken at the National Institute for Computational Sciences.

[1] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal, "Quantum monte carlo simulations of solids," *Rev. Mod. Phys.*, vol. 73, pp. 33–83, Jan 2001.

[2] K. P. Esler, J. Kim, D. M. Ceperley, W. Purwanto, E. J. Walter, H. Krakauer, S. Zhang, P. R. C. Kent, R. G. Hennig, C. Umrigar, M. Bajdich, J. Kolorenc, L. Mitas, and A. Srinivasan, "Quantum monte carlo algorithms for electronic structure at the petascale; the endstation project," *Journal of Physics: Conference Series*, vol. 125, p. 012057 (15pp), 2008.

[3] Amos G. Anderson, William A. Goddard, and Peter Schröder, "Quantum Monte Carlo on graphical processing units," *Computer Physics Communications*, vol. 177, pp. 298–306, 2007.

[4] Ivan S. Ufimtsev and Todd J. Martinez, "Quantum Chemistry on Graphical Processing Units. 3. Analytical Energy Gradients, Geometry Optimization, and First Principles Molecular Dynamics," *J. Chem. Theo. Comp.*, vol. 5, p. 2619, 2009.

[5] K. P. Esler, R. E. Cohen, B. Militzer, J. Kim, R. J. Needs, and M. D. Towler, "Fundamental High-Pressure Calibration from All-Electron

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE

Quantum Monte Carlo Calculations," *Phys. Rev. Lett.*, vol. 104, p. 185702, 2010.

[6] S. Chiesa, D. M. Ceperley, and S. Zhang, "Accurate, efficient, and simple forces computed with quantum monte carlo methods," *Phys. Rev. Lett.*, vol. 94, p. 036404, Jan 2005.

[7] A. Badinski and R. J. Needs, "Total forces in the diffusion monte carlo method with nonlocal pseudopotentials," *Phys. Rev. B*, vol. 78, p. 035134, Jul 2008.

[8] Jeongnim Kim, K. Esler, J. McMinis, B. Clark, J. Gergely, S. Chiesa, K. Delaney, J. Vincent, and D. Ceperley, "QMCPACK simulation suite, http://qmcpack.cmscc.org.".

[9] D. Alfè and M.J. Gillan, "Efficient localized basis set for quantum Monte Carlo calculations on condensed matter," *Phys. Rev. B.*, vol. 70, p. 1661101(R), 11 Oct. 2004.

[10] V. Natoli and D. M. Ceperley, "An Optmized Method for Treating Long-Range Potentials," *J. Comp. Phys.*, vol. 117, pp. 171–178, 1995.

[11] D. M. Ceperley and B. J. Alder, "Ground state of solid hydrogen at high pressures," *Phys. Rev. B*, vol. 36, pp. 2092–2106, Aug 1987.

[12] M. Dewing and D.M. Ceperley, *Recent Advances in Quantum Monte Carlo Methods*, ch. Methods for Coupled Electronic-Ionic Monte Carlo. Singapore: World Scientific, 2002.

Ken Esler recently began as a senior physicist at Stone Ridge Technology, after completing his postdoctoral work at the University of Illinois. His research interests include the quantum simulation of materials and high-performance computing, most recently on heterogeneous platforms. Esler holds a Ph.D. in physics from the University of Illinois at Urbana-Champaign. Contact him at kesler@stoneridgetechnology.com.

Jeongnim Kim is a senior research scientist at NCSA at University of Illinois. Her interests include electronic structure methods and algorithms, high-performance computing and software engineering. She has a PhD in physics from the Ohio State University and is a member of American Physical Society. Contact her at jnkim@illinois.edu.

Luke Shulenburger is a postdoctoral appointee at Sandia National Laboratories. His research interests include ab initio simulation of materials under extreme conditions. He holds a Ph.D. in physics from the University of Illinois at Urbana-Champaign. Contact him at lshulen@sandia.gov.

David Ceperley is a professor in the Physics Department at the University of Illinois and a staff scientist at NCSA. He has a PhD in Physics from Cornell University and is a member of the National Academy of Sciences. His research concerns using high performance computers to solve problems for quantum many-body systems, such as systems at low temperature or at high pressure. Contact him at ceperley@illinois.edu.

Digital Object Indentifier 10.1109/MCSE.2010.122 1070-9924/$26.00 2010 IEEE