Fine-grained parallelization of the Car–Parrinello *ab initio* molecular dynamics method on the IBM Blue Gene/L supercomputer

E. Bohm
A. Bhatele
L. V. Kalé
M. E. Tuckerman
S. Kumar
J. A. Gunnels
G. J. Martyna

*Important scientific problems can be treated via **ab initio**-based molecular modeling approaches, wherein atomic forces are derived from an energy function that explicitly considers the electrons. The Car–Parrinello **ab initio** molecular dynamics (CPAIMD) method is widely used to study small systems containing on the order of 10 to 10^3 atoms. However, the impact of CPAIMD has been limited until recently because of difficulties inherent to scaling the technique beyond processor numbers about equal to the number of electronic states. CPAIMD computations involve a large number of interdependent phases with high interprocessor communication overhead. These phases require the evaluation of various transforms and non-square matrix multiplications that require large interprocessor data movement when efficiently parallelized. Using the Charm++ parallel programming language and runtime system, the phases are discretized into a large number of virtual processors, which are, in turn, mapped flexibly onto physical processors, thereby allowing interleaving of work. Algorithmic and IBM Blue Gene/L™ system-specific optimizations are employed to scale the CPAIMD method to at least 30 times the number of electronic states in small systems consisting of 24 to 768 atoms (32 to 1,024 electronic states) in order to demonstrate fine-grained parallelism. The largest systems studied scaled well across the entire machine (20,480 nodes).*

Introduction

As the computational power of large parallel computers increases [1], the efficiency of modeling methods improves correspondingly, placing increasing demands on parallel programming techniques. Therefore, it is important to develop strategies capable of scaling the algorithmically complex, multiphase methods of important scientific applications to large numbers of processors. Here, we present a description of a fine-grained parallel implementation of the Car–Parrinello *ab initio* molecular dynamics (CPAIMD) algorithm [2–6] in a form suitable for nonexperts. This algorithm has been used to study key chemical and biological processes as well as to examine important problems in materials science, biophysics, nanotechnology, and solid-state physics [7–16]. (The term *fine grained* indicates that the implementation decomposes the computational work into small parts so that small systems can scale to numerous processors.) This study highlights the ability of synergistic research in parallel algorithm, hardware design, and methodological development to generate fast applications that allow new scientific insights to be garnered.

CPAIMD can be intuitively understood as numerically solving Newton's equations of motion using forces derived from electronic structure or *ab initio* calculations performed as the simulation proceeds, thereby permitting the examination of phenomena that require a model containing a representation of the electronic states to

0018-8646/08/$5.00 © 2008 IBM

describe the phenomena. Typically, CPAIMD implementations employ an (in principle) exact *ab initio* technique known as the *Kohn–Sham (KS) density functional theory* [17, 18], the practical implementation of which requires the use of an approximate functional [19–21] and a finite basis set, such as plane-waves [2–6]. However, it should be noted that the quality of both the basis sets [22–25] and the functionals [26] employed are continuously improving.

From a computational point of view, the CPAIMD method involves many phases, including multiple concurrent sparse three-dimensional (3D) fast Fourier transforms (FFTs), non-square matrix multiplications, and several concurrent dense 3D-FFT computations that possess nontrivial dependencies that are quite different from those found in a classical molecular dynamics computation [27]. The parallel 3D FFTs are themselves communication intensive because of all-to-all interprocessor communication patterns inherent in their computation. For example, all of the processors that have a portion of the dataset to be transformed must communicate. The efficient concurrent execution of hundreds of parallel 3D FFTs introduces another challenge. In order to switch between phases, movement of a large amount of data must be orchestrated between processors that generate relatively little computation. In general, parallelization of the phases of CPAIMD necessitates complex trade-offs between memory use, load balance of work across processors, and interprocessor communication costs. Basic MPI (Message Passing Interface)-based implementations of CPAIMD, such as the implementation developed by two of the authors several years ago [28], exhibit limited scalability, thus restricting the number of processors that can be effectively employed to roughly the same number of electronic states in the system. In contrast, recent efforts by us and others have overcome the states-equal-processors barrier [29–31].

In this paper, we show parallel scaling on processor numbers more than 30 times the number of electronic states in small systems consisting of 24 to 768 atoms (32 to 1,024 electronic states). The 768-atom and 1,024-state system scaled well on all 20,480 nodes of the IBM Blue Gene/L* (BG/L) torus network supercomputer located at the IBM T. J. Watson Research Center. (In other words, CPU times continued to diminish with increasing processor numbers.) Our implementation employs the Charm++ parallel programming language and runtime system [32]. Charm++ offers the benefits of overpartitioning via migratable objects, a concept whose implementation involves decomposing the problem into many more discretizations than available processors and mapping these parallel objects to processors at startup. Charm++ also allows the runtime system the freedom to adjust the location of objects with respect to processors as the system evolves. This approach allows us to obtain high scalability and low per-iteration CPU times on the small problem sizes of interest here. Performance is notably enhanced by exploiting the ability of Charm++ to enable the interleaving of computation and communication and to simplify the effective topologically aware mapping of work to processors, which is key to the efficient use of the BG/L torus network architecture. This case study illustrates the competing objectives that must be balanced in the parallelization of a complex multiphasic application and demonstrates a series of useful strategies and techniques that can be applied to optimize a wider class of problems on large parallel supercomputers, with or without a torus network.

This paper is organized as follows. We briefly review the basic CPAIMD algorithm [2–6] for completeness so that the reader can understand its parallelization. We highlight new approaches to basic elements [33, 34] of the CPAIMD technique that are based on the Euler exponential spline (EES). These approaches are key to our parallel scaling, allow improved scalar performance, and have not been described in previous reviews [2–6, 29]. We stress that the adoption of a CPAIMD method that is based on the work in Reference [35] permits the use of many highly efficient routines and, furthermore, allows our nonstandard, but effective, CPAIMD equations of motion to be derived from a simple Hamiltonian formalism. Next, we present the parallelization of the serial algorithm following the approach outlined in an earlier paper [29] that led to scaling on 1,024 processors of a 32-water-molecule system. We discuss significant extensions of our earlier work, including the use and parallelization of EES-based techniques and the improved parallelization of many of the other phases of the computation, in particular the 3D FFTs and matrix multiplications under Charm++. We then describe the architecture of the BG/L system, followed by a discussion of the CPAIMD optimizations motivated by the BG/L design. Next, we present the mapping of the parallel objects of CPAIMD to physical processors in the 3D torus network. Established Charm++ techniques that have been proven to be effective in scaling classical molecular dynamics on BG/L systems have been applied where appropriate [36]. Finally, we present scaling results for liquid water in system sizes consisting of 8 to 256 molecules under 3D periodic boundary conditions using standard parameters and a generalized gradient approximate (GGA) functional, specifically, the BLYP (Becke–Lee–Yang–Parr) functional [19, 20].

The computation

The ground-state electronic energy can be calculated by minimizing a functional of the electron density following

the principles of density functional theory (DFT), a formally exact theory [17, 18]. In the KS formulation of DFT used here, which is also formally exact, the electron density, $\rho(\mathbf{r}) = \rho(x,y,z)$ (i.e., a probability density in three spatial dimensions), is expressed as the sum of the modulus square of a set of single-particle KS electronic states (hereafter referred to simply as *states*). Typically, two electrons are considered to occupy each state. Unfortunately, the exact functional is not known, and an approximate functional must be employed. In this work, we provide for an arbitrary generalized gradient-corrected approximate functional of the type given in References [19–21]. Furthermore, a finite set of plane-waves is used to describe the single-particle states of the theory, as described in more detail below. Additionally, only the valence electrons of the atoms are treated. Thus, a single water molecule or H_2O—which contains 10 electrons of total charge −10e, an oxygen ion of charge +8e, and two hydrogen ions of charge +1e—is treated as an eight-electron system of total charge −8e. The value −8e arises because the oxygen moiety carries valence charge +6e (this is often referred to as an *ion core*), and the two hydrogen ions each carry a charge +1e. Thus, eight electrons are needed to neutralize the system. Rather than use the tedious nomenclature *ion core* throughout this paper, we will sometimes simply use the term *atom* (e.g., in subscripts), allowing the reader to determine the ion core (i.e., valence charge) by context.

Because KS DFT is a theory for noninteracting electrons in an effective self-consistent potential that yields the same ground state as the physical interacting electron system, the KS density functional contains several terms: the quantum mechanical *kinetic* energy of noninteracting electrons, the Coulomb repulsion or Hartree energy between negative charge clouds $\rho(\mathbf{r})$ and $\rho(\mathbf{r}')$ (or simply the Hartree term), the correction of the Hartree energy to account for the quantum nature of the electrons (or the *exchange-correlation* energy), and the interaction of the electrons with the atoms (ion cores) in the system (or the *external* energy). In the last term, the interaction of the valence electrons with the atoms (ion cores) is treated explicitly, and the core electrons are mathematically removed, resulting in the *electron–atom* (*ion-core*) *nonlocal* energy. This term should not be confused with nonlocal electron–electron interactions that are not of interest here. The form of the exchange-correlation energy is not known and must be approximated, as discussed above.

Once the functional is minimized, the forces acting on the atoms can be computed from the functional and the atom–atom (ion-core–ion-core) Coulomb interaction, and the positions and velocities of the atoms evolved in time according to a finite difference solution of Newton's equations of motion. The CPAIMD method is elegant because it avoids this discontinuous process and allows the two elements to occur simultaneously, enabled by a classical adiabatic principle [2, 37]. Using a variant of the nonorthogonal state dynamics in Reference [35], which is different from the standard CPAIMD technique, momenta are introduced conjugate to the plane-wave basis set coefficients and a fictitious classical kinetic energy introduced to form a fictitious classical Hamiltonian ($H_{\text{fict-tot}} = H_{\text{atom}} + H_{\text{fict-electron}}$) from which simple equations of motion naturally arise. Orthogonality constraints are absent because of the use of the nonorthogonal state-based approach [35]. As long as the plane-wave coefficients move rapidly compared to the atoms and exhibit small-amplitude oscillations around the minimum of the functional at the current (slowly evolving) atom positions, the method works well. The basic algorithm is summarized in its essential details in **Figure 1**. More-complete reviews with many details are given in References [2–6]. The new EES-based methods are described in more technical detail in References [33, 34].

A set of n_s electronic states is introduced that are represented as a set of (complex) Fourier coefficients, $\Psi(s,g_x,g_y,g_z)$, on a regular 3D g-space (or reciprocal Fourier space) of side N, where s is the state index and g_x,g_y,g_z index the Fourier coefficients. The array is not dense, but a sphere of radius g_{cut}, proportional to $N/4$, defines the n_g non-zero elements. The number of electronic states (n_s) and the number of points in g-space (n_g) both increase linearly with the number of atoms, N_{atom}, and the memory requirement is $\sim N_{\text{atom}}^2$.

In the course of the computation, each state switches between its g-space and its real-space representation via 3D FFTs of size $N \times N \times N$ (Phases I and VII; see Figure 1). As there are many states, this requires the computation of multiple simultaneous 3D FFTs. The real-space representation of the states, $\Psi(s,x,y,z)$, is a dense 3D array of real numbers of size $N \times N \times N$. (Recent improvements [38–40] that are based on localized states [41] are beyond the scope of this paper.) The real-space representation of the electron density, $\rho(x,y,z)$, is computed by squaring the real-space representation of each state and then summing over all states, $\rho(x,y,z) = \sum_s |\Psi(s,x,y,z)|^2$ (Phase II). The density in its real-space representation is, in any case, a dense $N \times N \times N$ array of real numbers.

The scalar work employed to generate the density scales as $N_{\text{atom}}^2 \log N_{\text{atom}}$, dominated by the computational cost of the 3D FFTs used to generate the real-space representation of the states. The (complex) Fourier coefficients of the density, $\rho(g_x,g_y,g_z)$, are obtained from a single 3D FFT of $\rho(x,y,z)$, which scales as $N_{\text{atom}} \log N_{\text{atom}}$. Note that this 3D FFT cannot be launched until the reduction is complete (Phase II). The g-space representation of the density is a single array in a g-space

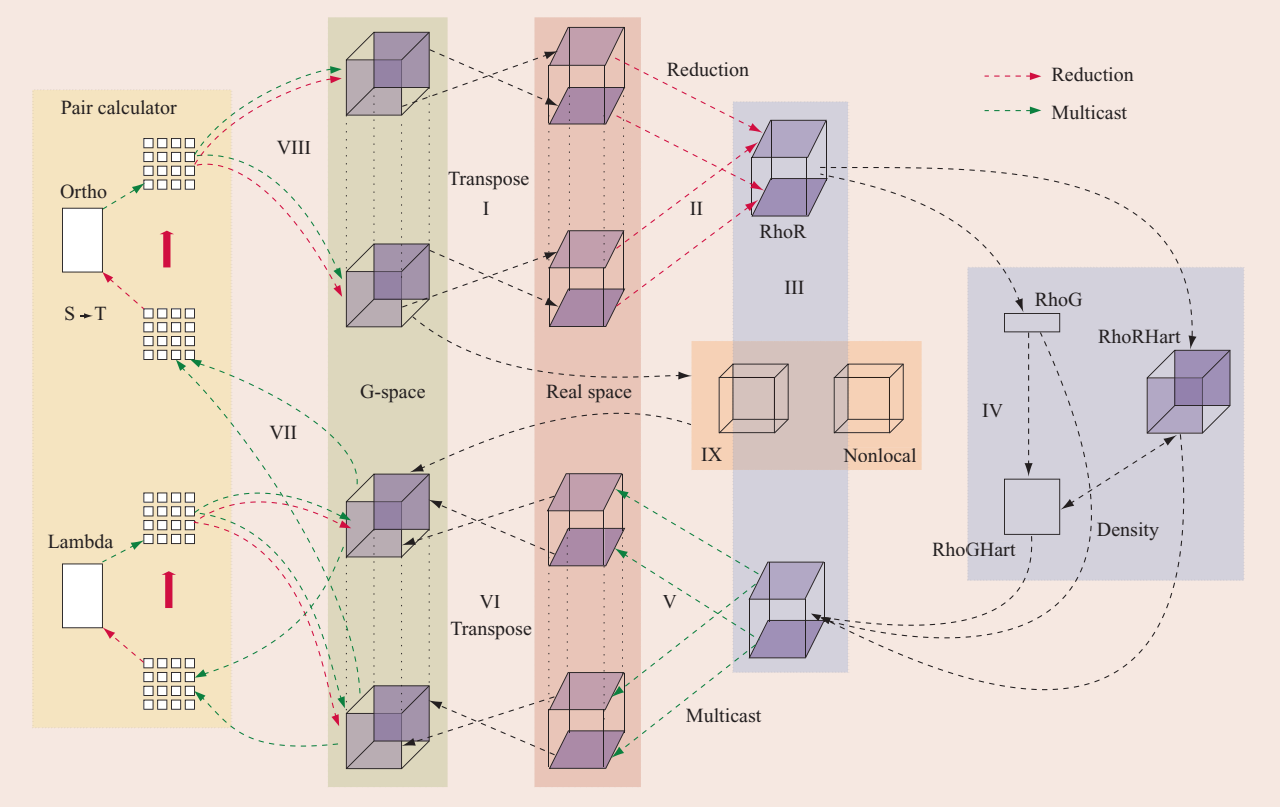

Figure 1

Structure of our implementation. Phases are in Roman numerals. RhoGHart refers to the chare array assigned to compute the g-space Hartree and external energy computations. RhoG is the chare array assigned to help compute gradients of the electron density. RhoRHart is the chare assigned to compute the real-space (r-space) Hartree and external energy computations. RhoR is the chare array assigned to compute the exchange correlation energy. Lambda and Ortho are the chare arrays assigned to handle post-processing of overlap matrices. Pair calculator is the four-dimensional chare array assigned to compute the overlap matrices. The small white squares emphasize that the overlap matrices are partitioned into blocks, here shown as a 4×4 array. The cubes represent the three-dimensional r-space and g-space datasets handled by the chare arrays.

of size $N \times N \times N$ that has non-zero values inside a sphere of radius $N/2$. The memory required to store the density in both real-space and g-space scales as N_{atom}.

Phases III and IV compute the Hartree, the external, and the exchange-correlation energies, as well as the KS potential, $v_{\text{ks}}(x,y,z)$. The approximate exchange-correlation functionals of interest here and their associated KS potential depend on both the density $\rho(x,y,z)$ and its spatial gradient $\nabla\rho(x,y,z)$, the latter requiring the computation of additional 3D FFTs. The external energy is computed via $2(N_{\text{atom-type}}+1)$ 3D FFTs within an EES formalism analogous to those in References [33, 34]. Here, $N_{\text{atom-type}}$ is the number of atom types present in the system (e.g., if only carbon, hydrogen, and nitrogen atoms are present, then there are three atom types). These 3D FFTs are 1.4 times larger on edge (i.e., $1.4N$) than those used to compute the exchange-correlation energy. Using the EES-based method, the computation of the external potential energy scales exactly as the computation of the exchange-correlation energy with system size N_{atom} log N_{atom}. The KS potential, a portion of which is related to each energy term, is computed via several more 3D FFTs and then sent (i.e., multicast) back to the states in the real space (Phase V). The derivative of the three energy terms with respect to the Fourier coefficients, $\Psi(s,g_x,g_y,g_z)$, is computed by multiplying each state in real space by the KS potential and performing an inverse FFT (Phase VI). The negative derivative of an energy term with respect to a Fourier coefficient of a given state is referred to as a *force*. The forces on the states are denoted $F_\Psi(s,g_x,g_y,g_z)$.

The electron–atom (ion-core) nonlocal energy and the kinetic energy of noninteracting electrons (Phase IX) can be computed independently from the above work, as depicted in Figure 1. The nonlocal energy and forces are computed using the EES technique described in

Reference [33]. Briefly, each state object performs $2N_{\text{atom-type}}$ 3D FFTs if, for example, only s-wave nonlocality is considered. The FFTs are smaller on edge ($\sim 0.7N$) than those employed to compute the density. The required scalar work scales as $N_{\text{atom}}^2 \log N_{\text{atom}}$, while the memory requirement scales as N_{atom}^2. The 3D FFTs are performed concurrently because the nonlocal energy does not couple the states together but couples each state individually to all the atoms. The kinetic energy of noninteracting electrons is expressed (in computer science parlance) as a *point-by-point multiply* and reduction operation whose computation scales as $\sim N_{\text{atom}}^2$ [e.g., $(\hbar^2/2m_e) \sum_{g_x,g_y,g_z \in |\mathbf{g}| < g_{\text{cut}}} \sum_s f_s g^2 |\Psi(s,g_x,g_y,g_z)|^2$ where $\hbar$ is Planck's constant, h, divided by 2π, and m_e is the mass of the electron].

The forces from the nonlocal energy and the kinetic energy of noninteracting electrons are added to those obtained from the Hartree, the external, and the exchange-correlation energies to complete the force computation. In order to proceed further, we note that the states satisfy an orthogonality condition expressed as a matrix multiplication

$$\sum_{g_x,g_y,g_z \in |\mathbf{g}| < g_{\text{cut}}} \Psi(s,g_x,g_y,g_z)\Psi^*(s',g_x,g_y,g_z) = f_s \delta_{ss'}, \quad (1)$$

where f_s is the occupation of each state and the indices are as above. δ is a Kronecker delta. Equation (1) fixes the normalization of the total wave function, and hence, the number of electrons, and ensures that the Pauli exclusion principle is satisfied. Typically, $f_s = 2$ and there are two electrons per state, one spin up and one spin down. The number of allowed points in g-space, n_g, is much larger than the number of states, and hence, the matrix multiplication is non-square. Enforcing the orthogonality constraints requires $\sim N_{\text{atom}}^3$ operations.

If the Fourier coefficients were naively updated using the coefficient forces alluded to above, the states would deviate wildly from the orthogonality constraint of Equation (1). Therefore, two orthonormalization steps are performed (Phases VII and VIII). The first step regularizes the forces that are then used to evolve the states. The second step corrects the small deviations of the new states from orthogonality and normalization constraints. Both steps scale as $\sim N_{\text{atom}}^3$. The order of operations given above is appropriate for functional minimization. Implementing the nonorthogonal state CPAIMD method employed in our software, which is based on Reference [35], simply requires a change in the order of operations and an additional matrix multiply, as described in more detail in Reference [35].

In summary, the memory requirement corresponds to $\sim N_{\text{atom}}^2$. The computational cost is $\sim N_{\text{atom}} \log N_{\text{atom}}$ for density-related work, $\sim N_{\text{atom}}^2 \log N_{\text{atom}}$ for state-related work, and $\sim \log N_{\text{atom}}^3$ for the orthogonality-related work. In small systems, the $\sim N_{\text{atom}}^2 \log N_{\text{atom}}$ work is dominant, while in large systems the $\sim N_{\text{atom}}^3$ work becomes the slow step.

Parallelization: Parallel operations and data decomposition

The processor virtualization approach [42] embodied in the Charm++ parallel programming system [32] forms the backbone of our CPAIMD application. Work is divided into a large number of objects or virtual processors (VPs) and the computation is initially expressed only in terms of VPs. Either the runtime system or the user can specify or change the mapping of VPs to physical processors at startup or during the execution of the application. The VPs are C++ objects (called *chares*), which communicate with each other via asynchronous method invocations or messages. Chares are organized into indexed collections, called *chare arrays*. In this way, Charm++ separates the issues of decomposition and mapping.

CPAIMD parallel operations and data decomposition are now described. The analysis is based on earlier work [29] that led to scaling on 1,024 processors of a 32-water-molecule system. Here, we highlight opportunities for interleaving communication and computation and reveal bottlenecks relevant to the BG/L system. The description of the data decomposition that follows shows how these opportunities can be exploited on a torus network architecture, such as the BG/L architecture, to generate scaling to a large number of processors. Nontrivial extensions of our earlier work include the use and parallelization of EES-based techniques, the improved parallelization of the orthogonality work (matrix multiplications), and the 3D-FFT work related to density-based computations.

Parallel operations

In this section, we review the phases from a perspective of parallelization and discuss certain novel features. As mentioned, in Phase I, the electronic states are transformed from their g-space representation to their real-space state representation. The parallel 3D FFTs employed to effect this change of representation are implemented using a transpose-based method [43]. Our parallel 3D FFT is performed in three stages. First, a set of one-dimensional (1D) FFTs are computed to yield $\Psi(s,g_x,g_y,g_z) \rightarrow \Psi(s,g_x,g_y,z)$. Second, a transpose is performed. Third, two sets of 1D FFTs are computed to generate $\Psi(s,g_x,g_y,z) \rightarrow \Psi(s,x,y,z)$ as a result of our planewise decomposition of $\Psi(s,x,y,z)$ (see the section "Data decomposition: Enabling parallelism and interleaving"). Because the states have a sparse spherical distribution within their 3D g-space grid but are dense in real space, this order of operations reduces the communication cost. The spherical cutoff and the real-to-

complex nature of the state 3D FFT reduce both computation and communication by approximately a factor of 4 compared to a full complex-to-complex 3D-FFT computation. On the BG/L system, the required serial 1D FFTs were performed using the ESSL (Engineering Scientific Subroutine Library). The ratio of computation to communication, defined as the ratio of the number of floating-point operations required to compute the quantity of interest in scalar mode to the number of bytes that must be communicated to compute the quantity of interest in parallel, is $\sim\log N_{\text{atom}}$.

The CPAIMD algorithm requires n_s, multiple, concurrent 3D FFTs to be performed. The computation phase of one 3D FFT is typically interleaved with the communication phase of several other 3D FFTs. In this way, the Charm++ programming model can effectively interleave computation and communication on the BG/L system.

The reduction to form the probability density in real space occurs in Phase II. As mentioned, once the states are available in real space, the electronic density can be computed via $\rho(x,y,z) = \sum_s |\Psi(s,x,y,z)|^2$. While the states in real space are decomposed into N-planes, the density in real space is decomposed into N_yN subplanes (see also the section "Data decomposition: Enabling parallelism and interleaving"). In other words, each of the N-planes is further subdivided into N_y parts. Therefore, the density construction requires spawning (N_yN) simultaneous reductions involving large datasets ($\sim 10^6$ real numbers). The ratio of computation to communication is roughly unity.

In Phases II–VI, the electron density is processed. The electron density in real space is immediately used to compute a portion of the exchange-correlation energy upon arrival. The Fourier components of the density are then created by subjecting a copy of the density in real space to a 3D FFT (Phase IV), which requires two transposes due to the subplane decomposition described in the section "Data decomposition: Enabling parallelism and interleaving." Earlier in this paper, we discussed how once in g-space, the Fourier coefficients of the density are used to compute the Hartree and external energies via $2(N_{\text{atom-type}} + 1)$ 3D FFTs, which can be performed independently. In addition, we note that the gradient of the electron density in real space is created by making three copies of the Fourier coefficients of the density, point by point multiplying each appropriately, and performing three 3D FFTs to real space (requiring six transposes). The electron density and its gradients are employed to compute a second gradient-dependent contribution to the exchange-correlation energy. The KS potential, $v_{\text{ks}}(x,y,z)$, is created by performing five more 3D FFTs (requiring ten transposes) on datasets generated during the energy computation. The ratio of computation to communication in this phase is $\log N_{\text{atom}}$.

The three energies are reduced across all processors involved, and N_y multicasts of degree n_s are performed in order to send each of the N_yN subplanes of the KS potential to the matching plane of each of the n_s states. This operation is a bottleneck and is, therefore, interleaved with the nonlocal energy computation.

After performing multiple 3D FFTs (Phase VI) on the product of the KS potential with each of the states, in a procedure exactly the reverse of that used to form the density, the forces $F_\Psi(s,g_x,g_y,g_z)$ are obtained in g-space. The ratio of computation to communication is again $\log N_{\text{atom}}$.

Phase IX requires the computation of the kinetic energy of the noninteracting electrons, which can be computed without interprocessor communication, and the computation of the nonlocal interaction between the electrons and the atoms via the new EES method [33]. The latter calculation involves $2N_{\text{atom-type}}$ 3D FFTs with a size of $\sim 0.7N$ on edge and is interleaved with Phases II–VI. The memory requirement and the ratio of communication to computation for this phase both scale as the 3D-FFT phase used to create the density (Phase I).

The force regularization and orthonormalization are performed in Phases VII–VIII. Once the forces have been computed, a series of matrix multiplications must be performed. For functional minimization, an $(n_s \times n_g) \times (n_g \times n_s)$ multiplication of the forces, $F_\Psi(s,g_x,g_y,g_z)$, by the states, $\Psi(s',g_x,g_y,g_z)$, to form the matrix $\Lambda(s,s')$ is followed by a modification of the forces, $F_\Psi(s,g_x,g_y,g_z) \leftarrow F_\Psi(x,g_x,g_y,g_z) - \sum_{s'} \Lambda(s,s')\Psi(s',g_x,g_y,g_z)$.

Under nonorthogonal state dynamics, the forces are further multiplied by the precomputed inverse square root matrix, which is available because of the change in the order of operations required by the dynamics method [35]. The modified forces are then employed to evolve the states in both cases.

Under functional optimization, the evolved states are slightly nonorthogonal and must be "reorthogonalized" by matrix transformation. This requires the computation of the overlap matrix, $S(s,s') = \sum_g \Psi(s,g_x,g_y,g_z)\Psi(s',g_x,g_y,g_z)$. Next, the inverse square root of the S-matrix $T(s,s') = S^{-1/2}(s,s')$ is computed and the orthonormal states $\Psi^{(\text{new})}(s,g_x,g_y,g_z) = \sum_s T(s,s')\Psi(s',g_x,g_y,g_z)$ are constructed. The matrix multiplications are non-square because the number of non-zero points in g-space, n_g, is much larger than the number of states, n_s. For the nonorthogonal state dynamics, the input nonorthogonal states are diagonalized using the same transformation, $\Psi^{(\text{ortho})}(s,g_x,g_y,g_z) = \sum_s T(s,s')\Psi^{(\text{non-ortho})}(s',g_x,g_y,g_z)$, and the inverse square matrix is stored for use later.

Our multiplication scheme has a novel feature, namely that the resultant matrix is used in a *backward path* in order to compute the necessary modification to the input data. The input data, $\Psi(s,g_x,g_y,g_z)$, is modified by the computed matrix $T(s,s')$ so that $\Psi^{(\text{new})}(s,g_x,g_y,g_z) = \sum_s T(s,s')\Psi(s',g_x,g_y,g_z)$, while the input data, $F_\Psi(s,g_x,g_y,g_z)$, is modified by the computed matrix $\Lambda(s,s')$ so that $F_\Psi(s,g_x,g_y,g_z) \leftarrow F_\Psi(s,g_x,g_y,g_z) - \sum_{s'} \Lambda(s,s')\Psi(s',g_x,g_y,g_z)$ for functional minimization. Hence, the Charm++ chare arrays assigned to perform the multiply do not destroy their input data. After summing contributions from all of the g-space to form S or Λ matrices and post-processing if necessary (e.g., to form the T-matrix), the result of the matrix multiplication is simply applied in place for each block. A reduction over rows, s', is then performed to generate the desired result. The procedure required by the new nonorthogonal state CPAIMD method that is based on Reference [35] is not significantly different.

In summary, under functional minimization, an iteration starts with a set of orthogonal input states that are used to compute forces. After force regularization, the states are evolved, and slightly nonorthogonal states are generated. These states are in turn orthogonalized and the entire procedure is repeated. Under nonorthogonal state dynamics, a set of input nonorthogonal states are first orthogonalized and the inverse square root matrix is stored. Forces are computed on the orthonormal states, and the forces on the orthonormal states are transformed to forces on the nonorthonormal states as described above. The nonorthonormal states are then evolved to the next step using the forces, and the procedure is repeated. This summary omits some of the details described elsewhere [35].

Note that the orthogonalization phase forms a barrier to further progress. The forces must be completed before orthogonalization starts, and new forces cannot be computed until the orthogonalization phase ends. Thus, the communication into and out of this phase is critical, as is its implementation within a data-driven model that allows the orthogonalization work to proceed (as far as possible) as data arrives and as data is sent on the outward path.

Data decomposition: Enabling parallelism and interleaving

The nine phases of CPAIMD are parallelized by decomposing the system into 14 indexed collections of objects, each implemented as a *chare array*. The g-space and r-space representations of the electronic states are decomposed into two 2D chare arrays of size (state times g-space collection) and (state times r-space collection), respectively. These are written as $G(s,p)[n_s \times N_g]$ and $R(s,p)[n_s \times N]$, respectively, where the dimension sizes are inside the square brackets. The g-space and r-space representations required for the external/Hartree energy work decomposed as the density but with an additional atom type of index, $G_{\text{HE}}(p,a)$, $[N_{g\text{HE}} \times n_{\text{atom-type}}]$ and $R_{\text{HE}}(p,p',a)[(1.4N/N_y) \times N_y \times n_{\text{atom-type}}]$ with $n_{\text{atom-type}} \leq N_{\text{atom-type}}$, and the g-space and real-space representations required for the nonlocal work decomposed as the states, $G_{\text{nl}}(s,p)[n_s \times N_g]$ and $R_{\text{nl}}(s,p)[n_s \times 0.7N]$. The orthogonality-related computation is decomposed into two 4D g-space-based arrays (called *pair calculators*) and a 2D auxiliary chare (called *ortho*). The computational work and memory are always well balanced within density and state chares. Orthogonality is inherently more expensive, with respect to computation and data movement, than all other elements in large systems.

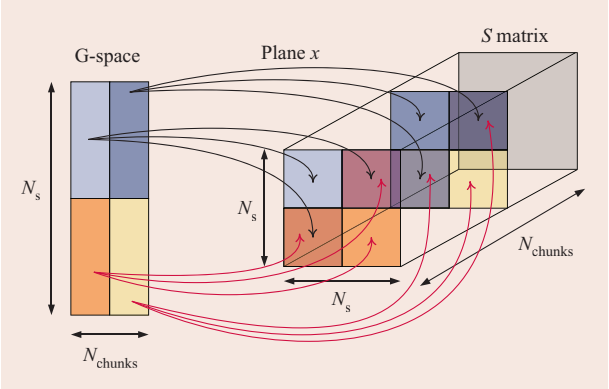

Figure 2

Computing the entries of the S matrix. Each chare-array element $P_c(s,s',p,p')$, $s \leq s'$ provides a contribution to an *sgrain* × *sgrain* tile of the S matrix ($sgrain = N_s/2$) for its section of g-space indexed by p,p' after receiving data from the state g-space chares. After computation, the g-space contributions are added together by summing over p,p' at fixed s,s' (a "section reduction") in order to produce the desired *sgrain* × *sgrain* tile of S. Here, $0 \leq p \leq 1$, $p' = 0$.

The orthogonality phase presented in **Figure 2** is performed by a set of four index pair calculator chare arrays, $P_c(s,s',p,p')$, of size $N_s \times N_s \times N_g \times N_g$. These arrays are designed to compute the matrix multiplications $(n_s \times n_g) \times (n_s \times n_g)$ that generate the overlap matrices of size $(n_s \times n_s)$. The s,s' indices control the decomposition of the $(n_s \times n_s)$ overlap matrices into chunks (i.e., portions) of size *sgrain* × *sgrain*, where $sgrain = (n_s/N_s)$. The p and p' indices control the decomposition of g-space, while the *sgrain* × *sgrain* portions of the overlap matrix are indexed by (s,s'). Therefore, an *sgrain* × *sgrain* portion of the overlap matrix assigned to pair calculator chare with indices s,s' is generated by performing a reduction over the (p,p') indices of $P_c(s,s',p,p')$. Here, N_g is exactly the same as above, and N_g allows for a finer decomposition of g-space, for example, splitting n_g into chunks of size $n_g/(N_g N_g)$.

Postprocessing of the overlap matrices, forming the T matrix from the S matrix, is performed by the ortho array, $O(s,s')$ $[N_{sO} \times N_{sO}]$, whose decomposition is generally finer than that of P_c, $N_{sO} \geq N_s$. Communication from the pair calculator chares to the ortho chares and back forms a bottleneck that requires careful attention. The time consumed in this bottleneck is minimized by the adoption of a data-driven model that enables interleaving, as described in the section "Interleaving communication and computation." The reduction to form the S and Λ matrices and multicasts of the T and Λ matrices back to the P_c chare arrays (from O) can create contention and imbalance within root and intermediate nodes in the spanning tree, particularly across N_g, unless these are balanced by shifting the roots for each N_g. Network contention is mitigated via the topology-aware mapping described in the section "Mapping challenges." In particular, the matrix multicasts can be further optimized using a rectangular multicast scheme enabled by topologically aware mapping described later in the section "Mapping challenges."

We note that the memory requirement of CPAIMD scales as $\sim N_{\text{atom}}^2$ in scalar mode. In the parallel implementation given above, the memory requirement scales like $\sim N_{\text{atom}}^{2/3}$ per processor because of the planewise decomposition. This assumes that we permit the number of processors used in the computation to increase as $\sim N_{\text{atom}}^{4/3}$. At a fixed problem size, the memory requirement per processor decreases until the number of processors, N_{proc} exceeds Nn_s ($N_{\text{proc}} > Nn_s$). This is also the point at which our decomposition would have to be further refined to continue scaling. For the largest system studied here, 256 water molecules, the total memory requirement in scalar mode is approximately 100 GB using the nonlocal EES method, a computation that barely fits onto 512 BG/L processors but is easily accommodated on $N_{\text{proc}} \geq 1{,}024$ in coprocessor mode.

In order to judge the scale of the computation, consider a 32-water-molecule system under a 70-Rydberg spherical g-space cutoff. The system contains $n_s = 128$ states, $N =$ 100-state r-space planes, and about $n_g = 64{,}000$ non-zero g-space points per state. The computation is decomposed into 12,800 VPs to represent the states in real-space, and about as many VPs to hold the non-zero points in g-space. In addition, there are 500 VPs representing density in real-space and reciprocal space (see Figure 1). Older MPI-based applications, such as those by two of the authors [28], scale this benchmark to 128 processors ($\sim n_s$).

The Blue Gene/L architecture

The BG/L system that we used herein to test performance [44] is a low-power massively parallel supercomputer with a 20,480-node installation at the IBM T. J. Watson Research Center. Each node has two PowerPC* 440 cores running at a low clock speed of 700 MHz. The performance of each BG/L core is enhanced through a second floating-point unit (FPU), called the *double hummer* [45], resulting in a peak performance of 2.8 Gflops per core. The second FPU is usable only with 16-byte aligned inputs. Linear algebra libraries such as Basic Linear Algebra Subprograms (BLAS) can use the double-hummer unit, and optimized sequential libraries are utilized as much as possible by our application.

The BG/L system has a 3D torus interconnection network [46] for messaging with a favorable ratio of network bytes to flops. The network has low messaging latency and good performance for short messages. Because the torus network has limited bisection bandwidth, localizing communication improves performance.

Message passing on the BG/L system is performed by the core itself. Packets are sent by writing application data to memory mapped to a FIFO (first-in, first-out) queue. Therefore, overlap of computation and communication is limited, although optimizations can be achieved by interleaving computation and communication given a data-driven programming model such as Charm++. Because the PowerPC 440 core can have three outstanding loads and three outstanding stores, packet throughput is limited, and each core can barely keep the entire network busy. Thus, each CPU is kept occupied during phases in which it sends messages to near neighbors. However, during phases in which messages are sent to far neighbors, the data rate is restricted by the torus or at the torus bisection, and communication can be overlapped with computation.

Parallelization: Techniques and trade-offs

In order to interleave communication and computation not only within the individual phases but also among phases that occur simultaneously and at the boundaries of phases that form natural barriers to progress, synergistic optimization of the following elements is required: 1) the 3D-FFT phase used to create the density by interleaving computation and communication of the many 3D FFTs, 2) the multicast or reduction of real-space state chares to the real-space density chares and the multicast from the density chares back to the state chares, 3) the interleaving of the nonlocal-related computation and communication with the density multicasts to and from the states, and intradensity chare communication and computational work, 4) the few 3D FFTs required to perform the density work, 5) the many simultaneous 3D FFTs required to perform the nonlocal computation, 6) the communication into and out of the pair-calculator chares to and from the g-space state chares in order to mitigate the natural barrier formed by the orthogonalization work, and 7) the communication into

and out of the pair-calculator chares to and from the ortho chares that post-process the overlap matrices. In order to accomplish these seven tasks, the following were all systematically improved: 1) the scalar performance, 2) the decomposition of work to chare array elements, 3) the topologically aware mapping of chare array elements to processors, and 4) interprocessor communication through use of the BG/L machine layer. In this process, we quite naturally borrowed any applicable general optimizations from previous work by the University of Illinois authors [47].

Scalar optimization

The scaling of the nonlocal energy computation with system size was reduced from N_{atom}^3 to $N_{\text{atom}}^2 \log N_{\text{atom}}$ via the new EES nonlocal energy method [33]. The scaling of the external local energy computation with system size was similarly reduced from N_{atom}^2 to $N_{\text{atom}} \log N_{\text{atom}}$ via an EES method. The use of EES techniques is unique to our CPAIMD implementation.

The orthogonalization scheme for functional optimization and nonorthogonal state dynamics [35] employed herein requires the computation of an inverse matrix square root. A second-order iterative method involving three ($n_s \times n_s$) matrix multiplications per cycle was implemented [48], which has a lower scalar computational work overhead than the standard technique and is easier to parallelize.

Decomposition optimization

Three important decomposition optimizations were developed. The first simultaneously balances the communication—from the g-space state chares, $G(s,p)$, to the pair-calculator orthogonality chares, $P_c(s,s',p,p')$—and the 3D-FFT work and communication. Briefly, the spherical truncation of g-space, which is key to obtaining high scalar efficiency and reducing communication in both the state 3D-FFT phase and the orthogonality phase, can lead to unbalanced chare array elements if a naive decomposition is employed. Creating sets of complete g_z-lines of state Fourier coefficients in order to balance the number of points and lines in each $G(s,p)$ chare array element leads to substantial performance improvement. It also permits the introduction of a free parameter, N_g, that can be used to tune the granularity of $G(s,p)$.

The second optimization increases the degree of parallelism of the orthogonality phase. Rather than keeping the collection of g-space points assigned to each $G(s,p)$ intact, and simply communicating them to appropriate $P_c(s,s',p)$ calculators, the collections were further split. This was implemented by simply adding another index to the pair-calculator chare array, $P_c(s,s',p,p')$. Increasing the degree of parallelism increases the number of messages sent, but each message is smaller. The total amount of data to be reduced in the formation of the overlap matrices is also increased. However, the trade-off results in efficiency gains.

The third optimization increases the degree of parallelism of the density work. The g-space representation of the density has about eight times the number of non-zero entries as the g-space representation of a state. Therefore, the computational cost to transform the density between g-space and real-space is about five times that required to transform a state. Further decomposing the density into subplanes, $R_\rho(s,p,p')$, resulted in higher efficiency despite the communication cost of an additional transpose. In addition, the g-space representation of density was decomposed into sets of complete g_z-lines of density Fourier coefficients in order to balance the number of points and lines in each chare array element, $G_\rho(p)$, allowing $N_{g\rho}$ to become an adjustable parameter in precisely the manner given above for the states.

Mapping challenges

The Charm++ system supports a default user-provided mapping of VPs to real processors. Because of the complexity of the CPAIMD application, there are opportunities for intelligent mapping to simultaneously optimize load balance and communication overhead. Under CPAIMD, the load on each VP is static, and therefore, the mapping can be defined at startup.

A simple map that allocates all planes of a state to the same processor would make all 3D-FFT transpose messages local, resulting in good performance. This map is not scalable, because we desire to run the application on processor configurations much larger than the number of states. We call this function the *state map*. In contrast, the planes of the same rank in all the states could be placed on the same processor. This mapping would optimize the KS potential multicast operation (see Figure 1, Phase V) from the density objects to real-space plane state objects. Of course, keeping planes together would make the FFT transpose messages highly nonlocal (Phases I and VI), thereby increasing the communication overhead of the application. This mapping function is referred to as the *plane map*.

A compromise between the state and plane mapping would allocate planes of a state partition on a processor partition. This would result in a smaller fanout for the KS potential multicast and keep FFT messages local or restrict them to nearby processors in the network. We refer to this map as the hybrid map. The mapping functions, state map and plane map, are two extremes of the hybrid map. The best scheme depends on the number of processors and the interconnect message latency and bandwidth.

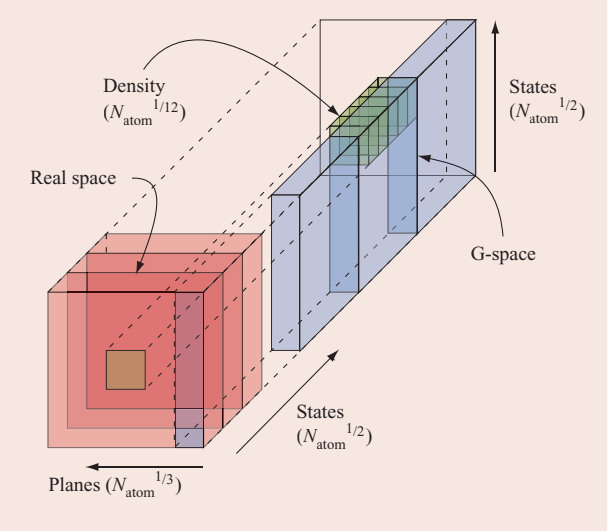

Figure 3

Placement of g-space, real-space, and density objects on the 3D torus.

Careful topology-aware mapping and relationship mapping are critical design issues for software that must run efficiently on a 3D network torus architecture. The concept of data locality must be extended to consider placement on neighboring processors. The cost of communication between processors is affected by their distance within the torus. The effective bandwidth consumed by a message is its size multiplied by the number of links that must be traversed from sender to destination in the torus, referred to as the *hop count*. Network contention resulting from link saturation can severely damage the performance of a communication-intensive application such as CPAIMD. The most efficient use of bandwidth and, therefore, the best performance for CPAIMD come from minimizing the distance between objects, which frequently communicate, while maintaining good load balance.

The key to defining an effective mapping of VPs to processors in the CPAIMD application lies in placement of the state g-space VPs, $G(s,p)$, where s is the state index and p is the plane index. Both the real-space VPs and the orthonormalization VPs depend on the $G(s,p)$ placement.

In order to localize communication, we used a box mapping for $G(s,p)$. VPs with the same plane index, p, were placed in rectangular prisms. These rectangular prisms were selected such that their long axis spanned one dimension of the torus and was oriented along whichever of the X, Y, or Z torus axes that allowed the torus to be completely tiled by the prisms. The number of prisms is equal to the number of planes and was always chosen at configuration time to be a multiple of at least two torus dimensions (choices involving power of 2 suffice on natural BG/L system partitions). Within each prism, the chares for $G(s,p)$ were allocated in increasing state order longitudinally along the long axis of the prism. **Figure 3** shows the map employed on the BG/L torus network. The matrix-multiplication/pair-calculator VPs, $P_c(s1, s2, p, p')$, could then be placed starting at the centroid of the 3D object formed by $G(s1,p) + \ ... \ G(s1 + sgrain, p) + G(s2, p) + ... + G(s2 + sgrain, p)$ within the prism for each plane p, thereby minimizing the hop count for the orthonormalization input and output. The ortho VPs, $O(s1 \ ... \ s1 + sgrain, s2 \ ... \ s2 + sgrain)$, can likewise be placed on the basis of the $P_c(s1 \ ... \ s1 + sgrain, s2 \ ... \ s2 + sgrain, *, *)$ processor list, but this led only to an ideal hop count for elements on the diagonal $s1 = s2$, motivating further communication optimizations described in the section "Interleaving communication and computation." In other words, the ortho chare array element $O(i,j)$ is assigned to the processor selected from the set of all processors mapped to P_c chare arrays that share the same two indices, or $P_c(i,j,*,*)$, which minimizes communication.

The real-space VPs $R(s,p)$ were placed with one state per flat prism formed by $G(s,*)$. The previously described longitudinal statewise placement within $G(s,p)$ resulted in flat bounding prisms for each state orthogonal to the long axis of each prism. The $R(s,p)$ chares were placed in increasing order of plane index along one axis of the flat bounding prism, thereby providing an orthogonal localization for the plane indices. The density-related chares $R_\rho(p,*,*)$ were placed on processors proximal to state real-space chares, $R(s,p)$, by starting with the centroid of each $R(*,p)$, then $G_\rho(p,*,*)$ is mapped near, but not on, the processors used by $R_\rho(p,*,*)$. The nonlocal g-space EES chares were pinned to the $G(s,p)$, while its r-space nonlocal EES chares $R_{EES}(s,*)$ were mapped on the basis of the placement of $G(s,*)$. If there were sufficient processors, the R_{EES} map excluded the processors used by the density objects to minimize interference during overlap.

A critical result of the mapping scheme is that it reduces the maximum hop count for each phase; that is, phases that might otherwise exhibit communication patterns spanning the entire torus in a naive mapping scheme are instead confined to communicate within prism-shaped subtori. This, in turn, balances the overall communication load throughout the available torus. Performance degradation due to network contention and areas of high network traffic is avoided, while each phase remains able to exploit the resources of the entire processor allocation as needed. In more detail, planewise communications in the orthonormalization chares were confined to each $G(*,p)$ prism. Statewise communications

$G(s,*) \leftrightarrow R(s,*)$ were confined to planes orthogonal to the long axis of the $G(s,p)$ prisms. Planewise communications $R(*,p) \leftrightarrow R_\rho(p,*,*)$ were likewise confined to a different set of prisms. Each of these prism objects spanned one or more torus dimensions, permitting the torus wraparound to reduce the maximum hop count within each prism to approximately half the length of the largest spanned dimension. The hop count could in many cases be reduced further via the centroid scheme described above.

Finally, we note that the memory cost of these maps grows linearly (three integers per VP) in proportion to the number of VPs, which were a few megabytes in the largest system studied. The runtime cost of creating the most complex of these maps is $pn^2\log(n)$, where n is the number of VPs and p is the number of processors; however, the constant time is sufficiently small for the largest runs to require less than a few minutes to map. Nevertheless, after being created, maps were stored once and reloaded in subsequent runs in order to minimize restart time. The offline creation of maps by using even more sophisticated techniques and by adapting these ideas to other topologies is an area of future work.

Charm++ native layer for the BG/L system

The MPI communication software stack on the BG/L system [49] has two messaging protocols, *eager* and *rendezvous*. Because MPI requires message ordering, eager protocol [49] packets are routed deterministically, but this protocol works well only for short messages. For long messages, the rendezvous protocol is used, in which a rendezvous packet is sent to the destination, where the MPI tags are matched and an acknowledgment is sent back. Upon receiving the acknowledgment, the source sends the packets to the destination with adaptive routing for higher network throughput. The performance of the rendezvous protocol is affected when the rendezvous packets are delayed by other packets. This approach prevents some sources from making progress on their message processing.

In contrast, Charm++ does not require message ordering. This allows different components of an application to make progress independently, thereby enabling the effective interleaving of computation and communication. Ordering and synchronization are handled at the application and runtime levels. All messages are unexpected and are not required to carry additional tag information. Handling of this messaging scheme within MPI requires frequent calls to *MPI_Test( )* and *MPI_Probe( )* in order to drive network buffer progress, which introduces undesirable overhead. The flexibility of Charm++ allows it to be built on top of a message layer that is lower than MPI; the Charm++ native layer is built on the BG/L system message layer [50], which reduces message overhead and permitted the development of the new, more efficient protocol described in the following section. A comparison of the performance of the MPI-based layer and the native layer is given in the section "Results."

Finally, we note that the per-node memory of the Charm++ runtime system on the BG/L system does not increase with processor number at fixed system size.

The adaptive eager communication protocol

Since Charm++ does not require message ordering and all messages are received as unexpected messages, it can take advantage of the adaptive eager protocol [36]. Here, messages are sent on the network as eager messages without handshakes and with adaptive routing. Hence, this protocol has low message overhead and good network throughput. Messages sent with adaptive-routing packets could arrive out of order, which would violate MPI semantics, but this is not a problem for Charm++. However, each packet must carry the size of the message in order to allocate a buffer for the entire message. The packet must also carry the source rank, offset in the message, and a sequence number to uniquely identify it at the receiver. Fortunately, the software header for a torus packet has space for up to 1-MB messages and a 4-bit sequence number. This allows each processor to send 16 messages to every other processor in the partition. After a processor receives 16 messages from a given source, it sends an acknowledgment message back to the source to send the next 16 messages. We have observed that this protocol significantly improves performance of Charm++ applications on the BG/L system.

Optimized multicasts

Several multicast operations exist in CPAIMD. In Figure 1, Phases V and VIII involve each-to-many communication operations with large messages. Two techniques are used to optimize this communication. For the *randomized direct multicast*, each of the NN_y subplanes of the density in Phase V of the computation sends n_s messages to the corresponding state real-space planes. These are sent as point-to-point messages using the adaptive eager protocol with adaptive routing. They are also randomized to saturate the links at the core bisection of the BG/L torus [46]. For the *rectangular multicast* on the BG/L system, the destination objects of a multicast can sometimes be mapped onto a rectangular prism, such as in the orthonormalization phases. Here, we exploit the deposit-bit broadcast capabilities of the network hardware, in which each packet can be sent to all the destinations on a line, thereby reducing message overhead.

Optimizing parallel 3D FFTs by message combining

The overhead of sending several short messages during 3D-FFT phases was optimized using a streaming

communication strategy. These messages are sorted into buckets on the basis of the destination processor rank. When the bucket for a particular destination reaches full capacity, the messages are combined and sent to the destination as one message, resulting in a lower per-message overhead. A bucket size of five messages was found to be optimal.

Interleaving communication and computation

The density (Phases III and IV) and nonlocal (Phase IX) computations can be interleaved because these phases involve 3D FFTs and, hence, all-to-all communication. Again, on the BG/L system, the processor itself must packetize messages, and overlap between computation and communication is not achieved easily. However, with all-to-all communication, the rate at which the data arrives at the processors is limited by the bisection bandwidth of the torus and gains from the overlap of computation and communication are possible. Effective overlap allows involved VPs (here, nonlocal and density chares) to share the same physical processor on smaller machines.

In order to enable interleaving, *network progress calls* are inserted into the core compute loop of an application every tens of thousands of processor cycles [47]. On the receiver side, progress calls ensure that arriving packets are processed while the lag time is used to perform computation. On the sender side, progress calls fill up the 24 network FIFOs, thereby allowing the application to compute while messages drain from the network. This approach was borrowed from other work by the University of Illinois authors [47].

The application of the T or Λ matrices in the backward path of the pair-calculator chares described previously provides further opportunity for overlap using a data-driven programming model. As the construction of tiles of T or Λ are completed and communicated back from the ortho chares, O, to the pair-calculator chares, P_c, these can be multiplied by the original data as they arrive. (The granularity in the state indices of O is typically finer than that of the P_c, $N_{sO} \geq N_s$.) Furthermore, the resulting components can be returned to appropriate G chare array elements as soon as possible, and the reduction performed in the G chare array as messages arrive. In this way, interleaving computation and communication is achieved while reducing peak bandwidth usage.

Results

The efficiency of the Charm++-based implementation of CPAIMD was investigated by using liquid water as a test case because of the importance of aqueous solutions in biophysics [51]. A single water molecule, H_2O, has three atoms per molecule and two atom types, and it possesses four doubly occupied valence electron states. All the computations described in this section were performed using the standard g-space spherical cutoff radius on the states of $|\mathbf{g}|^2_{\text{cut}} = 70$ Ry (i.e., Rydberg) at the Γ point (1 k-point), 3D periodic boundary conditions, the BLYP generalized gradient-corrected density functional [19, 20], and Martins–Troullier type of pseudopotentials [52]. [The Γ point refers to the condition of $k = (0,0,0)$.] More-complex systems with up to five atom types are currently running using our framework and will be discussed in a later publication. The framework is not limited to five atom types, but this is simply the maximum number attempted to date.

We note that water is an extremely common CPAIMD simulation target [10–12]. Other groups that are now developing fine-grained parallel CPAIMD software [30, 31] have not yet provided recent scaling data on water with standard parameters, and we would be hesitant to estimate timings for software other than our own. Here, we publish a set of unambiguous timings for a well-studied system, liquid water, using standard parameters and a well-known architecture so that other groups can compare their results to ours. We note that 2-year-old scaling data on 32 water molecules using the nonstandard cutoff, $|\mathbf{g}|^2_{\text{cut}} = 100$ Ry, on small processors numbers, $N_{\text{proc}} \leq 512$, has been presented elsewhere [30]. The one published data point for the 32-molecule water system (using standard parameters, $|\mathbf{g}|^2_{\text{cut}} = 70$ Ry) is 0.35 s/step on 512 processors [30]. This likely does not represent the current status of their project.

Specifically, for our study, we selected six liquid water systems consisting of 8, 16, 32, 64, 128, and 256 molecules and 32, 64, 128, 256, 512, and 1,024 doubly occupied electron states, respectively, in order to probe the performance of our CPAIMD application at many limits. In the small systems with 8 and 16 water molecules, the 3D-FFT work is dominant. In large systems with 128 and 256 water molecules, the orthogonalization work is more dominant. In the intermediate-size systems with 32 and 64 water molecules, the 3D-FFT work and orthogonalization have more-balanced workloads. In Reference [31], a 1,000-atom system of solid molybdenum under 3D periodic boundary conditions was studied using from 1 to 8 k-points and a cutoff of 44 Ry. Compared to our largest benchmark, 256 water molecules using 1 k-point (the Γ-point), the number of non-zero g-vectors per state in the previous work is essentially the same, but the molybdenum system has $n_s = 6{,}000$ doubly occupied states and is more strongly dominated by the orthogonality work than the 256-water-molecule system. The computational cost of the orthogonality computation in the 1,000-molybdenum atom system is 36 times that of the orthogonality computation in the 256-water-molecule system. There are significantly more nonlocal electron–atom (ion-core) interactions in molybdenum than in

Table 1 Parallel performance for liquid water. (CO: coprocessor; VN: virtual node; no topo: no topological or relational maps; no ESS: no Euler exponential spline; WMs: water molecules.)

CO-mode native layer with optimizations											
Nodes	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	20,480
Processors	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	20,480
Time (seconds/step)											
8 WMs	0.22	0.10	0.082	0.071	0.046	0.026	0.020				
16 WMs	0.73	0.40	0.23	0.15	0.106	0.061	0.041	0.035			
32 WMs	2.71	1.52	0.95	0.44	0.26	0.15	0.11	0.081	0.063		
64 WMs		6.62	3.75	1.88	0.87	0.51	0.31	0.21	0.15		
128 WMs					6.9	2.73	1.40	0.91	0.58	0.37	0.3
256 WMs						16.4	8.14	4.83	2.75	1.71	1.54
VN-mode native layer with optimizations											
Nodes	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	20,480
Processors	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,768	40,960
Time (seconds/step)											
8 WMs	0.13	0.11	0.08	0.06	0.028	0.021					
16 WMs	0.46	0.28	0.19	0.13	0.08	0.047	0.035				
32 WMs	1.99	1.40	0.81	0.43	0.174	0.13	0.082	0.067			
64 WMs		9.07	3.38	1.71	0.67	0.38	0.22	0.17	0.15		
128 WMs					3.0	1.48	0.90	0.65	0.48	0.40	0.3
256 WMs							5.10	3.48	2.41	1.47	1.2
Performance without selected optimizations for comparison to CO mode											
Nodes	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	20,480
Processors	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	40,960
Time (seconds/step)											
32 WMs (MPI)						0.33	0.22	0.17	0.12		
32 WMs (no topo)						0.21	0.23	0.16	0.18		
32 WMs (no EES)						0.22	0.14	0.098	0.082		
256 WMs (no topo)						28.8	23.0	13.4	6.83	3.40	

water (i.e., 250 times more interactions), and these were treated [31] using the standard N_{atom}^3 method (to the best of our knowledge). Thus, the molybdenum system has significantly more computational work to parallelize than the 256-water-molecule system. Scaling results starting at 1,024 nodes are given for the molybdenum system with 1 *k*-point in Reference [31].

We performed numerical tests to examine the extreme scaling limit of the CPAIMD application on our benchmark suite. **Table 1** (top) shows the present performance up to processor numbers that are many times higher than physical parameter values such as the number of electronic states.

Weak scaling is observed in all cases. (Weak scaling studies generally grow the problem size and the number of processors together in order to preserve the work per processor.) Thus, the timing of approximately 0.2 s/step is observed for 8 water molecules at 32 nodes, for 16 water molecules at 128 nodes, for 32 water molecules at 512 nodes, and for 64 water molecules at 4,096 nodes in coprocessor mode, in which the number of nodes equals the number of processors. For 128 water molecules on 20,480 nodes in coprocessor mode, a rather promising time of 0.3 s/step is achieved. This is promising in the sense that this is fairly close to the 0.2-s/step timing required for strong scaling on 32,768 nodes. However, in

virtual node mode for 128 water molecules using 16,384 nodes and 32,768 processors, we attain a slower timing than using 20,480 nodes in coprocessor mode. This motivates future work optimizing our CPAIMD application for large systems using large processor numbers. For example, more scaling tests on the full BG/L machine are required to analyze and eliminate the bottleneck for this case. Note that the other systems studied exhibited quite good performance in virtual node mode even at high processor numbers. The number of processors required to reach 0.2 s/step increases as N_{atom}^2 at small system size and as N_{atom}^3 at a large system size, as expected. Good strong scaling is also observed. (Experiments that are *strong scaling* refer to studies in which researchers fix the problem size and increase the number of processors.) The CPU time per step is reduced fairly monotonically as processor number increases at fixed system size until the scaling limit is reached. The scaling limit appears at processor numbers much greater than the number of states in the system ($N_{\text{proc}} \geq 30n_s$). We also note that our time per step for 32 water molecules is 0.26 s/step on 512 nodes (and processors) in coprocessor mode. In the next few paragraphs, we describe some of the more important elements that led to these results.

First, Charm++ can be compiled using MPI as its machine interface or using a BG/L system-customized machine interface [36] called the *native layer*. The scaling of the CPAIMD application on the MPI driver was limited (see Table 1) because MPI adds a level of overhead and imposes message ordering, which is not necessary or pertinent for this application.

Second, because the BG/L system employs a torus network architecture with a limited bisection bandwidth, efficiency gains can be achieved by implementing VP mapping optimizations that improve communication locality. Table 1 (see rows labeled "no topo" [no topology]) shows the performance degradation if topology-specific and relational maps are disabled in the application. In the simple mapping scheme, the work is spread over processors without regard for network locality. As can be seen in the no topo line for the 256-water-molecule system, the utility of topology mapping increases with the size of the torus, exceeding a factor-of-2 reduction in CPU time on most of the 256-water-molecule data at large processors. The no-topo runs reach the scaling limit before 8,192 processors for the 32-water-molecule test case.

Third, we observed that the nonlocal and density computations had limited parallelism when implemented using $N_{\text{atom}}^3$and N_{atom}^2 algorithms rather than the $N_{\text{atom}}^2 \log N_{\text{atom}}$ and $N_{\text{atom}} \log N_{\text{atom}}$ EES-based methods. In Table 1 (labeled "no EES"), results produced using the standard methods are given for comparison. The EES-based methods permit more parallelism using the decomposition described in the text, as can be seen by the increasing beneficial effect of EES on processor numbers higher than 1,000. The standard methods could be made to scale better if a finer decomposition of the states is implemented, as described elsewhere [31].

Conclusion

A fine-grained parallel implementation of the CPAIMD method using the concept of processor virtualization facilitated scaling of important systems to physical processor numbers greater than or equal to 30 times the number of electronic states on the IBM BG/L system. Virtualization and adaptive interleaving of communication (automatically engendered by the Charm++ runtime system), novel topologically aware mapping of objects to processors, and new scalar algorithms resulted in improvements both in the number of processors used and in absolute performance. The study demonstrates the ability of synergistic research in hardware, software, and science to generate efficient and useful applications.

Acknowledgments

We thank Dr. Fred Mintzer, Dr. Bob Walkup, and the Blue Gene/L team at IBM T. J. Watson Research Center, because this work would not have been possible without their help and access to the hardware. We thank Dr. Gheorghe Almasi for assistance with the BG/L message layer. We also thank Philip Heidelberger for technical support with respect to the BG/L torus interconnection network. We thank Professor Jason Crain and Mr. Raphael Troitzsch, University of Edinburgh, and the staff of the Edinburg Parallel Computing Center because the critical initial BG/L porting and scaling research relied on their support. This work was supported by the National Science Foundation (ITR-0121357 and ITR-0229959).

References

1. IBM Blue Gene team, "Blue Gene: A Vision for Protein Science Using a Petaflop Supercomputer," *IBM Syst. J.* **40**, No. 2, 310–327 (2001).
2. R. Car and M. Parrinello, "Unified Approach for Molecular Dynamics and Density-Functional Theory," *Phys. Rev. Lett.* **55**, No. 22, 2471–2474 (1985).
3. G. Galli and M. Parrinello, "*Ab Initio* Molecular Dynamics: Principles and Practical Implementation," *Computer Simulations in Materials Science, NATO Asi Series E: Applied Sciences*, Vol. 205, M. Meyer and V. Pontikis, Eds. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991, pp. 283–290.
4. M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopolous, "Iterative Minimization Techniques for *Ab-*

Initio Total-Energy Calculations: Molecular Dynamics and Conjugate Gradients," *Rev. Mod. Phys.* **64**, No. 4, 1045–1097 (1992).
5. M. E. Tuckerman, "*Ab Initio* Molecular Dynamics: Basic Concepts, Current Trends and Novel Applications," *J. Phys. Condensed Matter* **14**, No. 50, R1297–R1355 (2002).
6. D. Marx and J. Hutter, "*Ab Initio* Molecular Dynamics: Theory and Implementation," *Modern Methods and Algorithms of Quantum Chemistry, Proceedings*, Second Edition, J. Grotendorst, Ed., *NIC Series*, FZ Jülich, 2000, pp. 329–477.
7. A. De Vita, G. Galli, A. Canning, and R. Car, "A Microscopic Model for Surface-Induced Diamond-to-Graphite Transitions," *Nature*, **379**, 523–526 (February 1996).
8. G. Galli, R. M. Martin, R. Car, and M. Parrinello, "Melting of Diamond at High Pressure," *Science* **250**, No. 4987, 1547–1549 (1990).
9. D. Kruger, H. Fuchs, R. Rousseau, D. Marx, and M. Parrinello, "Pulling Monatomic Gold Wires with Single Molecules: An *Ab Initio* Simulation," *Phys. Rev. Lett.* **89**, No 18, 186402-1–186402-4 (2002).
10. M. Benoit, D. Marx, and M. Parrinello, "Tunneling and Zero Point Energy in High Pressure Ice," *Nature* **392**, No. 6673, 258–261 (1998).
11. D. Marx, M. E. Tuckerman, J. Hutter, and M. Parrinello, "The Nature of the Hydrated Excess Proton in Water," *Nature* **367**, No. 6720, 601–604 (1999).
12. M. E. Tuckerman, D. Marx, and M. Parrinello, "The Nature and Transport Mechanism of Hydrated Hydroxide Ions in Aqueous Solution," *Nature* **417**, No. 6892, 925–929 (2002).
13. D. Alfe, M. J. Gillan, and G. D. Price, "The Melting Curve of Iron at the Pressures of the Earth's Core from *Ab Initio* Calculations," *Nature* **401**, No. 6752, 462–464 (1999).
14. P. Minary and M. E. Tuckerman, "Reaction Mechanism of *cis*-1,3-Butadiene Addition to the Si(100)-2 × 1 Surface," *J. Am. Chem. Soc.* **127**, No. 4, 1110–1111 (2005).
15. R. Iftimie, P. Minary, and M. E. Tuckerman, "*Ab Initio* Molecular Dynamics: Concepts, Recent Developments, and Future Trends," *Proc. Natl. Acad. Sci. USA* **102**, No. 19, 6654–6659 (2005).
16. R. Martonák, D. Donadio, A. R. Oganov, and M. Parrinello, "Crystal Structure Transformations in SiO2 from Classical and *Ab Initio* Metadynamics," *Nat. Mater.* **5**, No. 8, 623–626 (2006).
17. W. Kohn and L. J. Sham, "Self-Consistent Equations Including Exchange and Correlation Effects," *Phys. Rev.* **140**, No. 4A, A1133–A1138 (1965).
18. R. G. Parr and W. Yang, *Density Functional Theory of Atoms and Molecules*, Oxford University Press, New York, 1989.
19. A. Becke, "Density-Functional Exchange-Energy Approximation with Correct Asymptotic Behavior," *Phys. Rev. A* **38**, No. 6, 3098–3100 (1988).
20. C. Lee, W. Yang, and R. Parr, "Development of the Calle–Salvetti Correlation Energy into a Functional of the Electron Density," *Phys. Rev. B* **37**, No. 2, 785–789 (1988).
21. J. P. Perdew, K. Burke, and M. Ernzerhof, "Generalized Gradient Approximation Made Simple," *Phys. Rev. Lett.* **77**, No. 18, 3865–3868 (1996).
22. Y. Liu, D. A. Yarne, and M. E. Tuckerman, "*Ab Initio* Molecular Dynamics Calculations with Simple, Localized, Orthonormal Real-Space Basis Sets," *Phys. Rev. B* **68**, 125110-1–125110-8 (2003).
23. H. S. Lee and M. E. Tuckerman, "*Ab Initio* Molecular Dynamics With Discrete Variable Representation Basis Sets: Techniques and Application to Liquid Water," *J. Phys. Chem. A* **110**, No. 16, 5549–5560 (2006).
24. H. S. Lee and M. E. Tuckerman, "Structure of Liquid Water at Ambient Temperature from *Ab Initio* Molecular Dynamics Performed in the Complete Basis Set Limit," *J. Chem. Phys.* **125**, No. 15, 154507-1–154507-14 (2006).
25. H. S. Lee and M. E. Tuckerman, "Dynamical Properties of Liquid Water from *Ab Initio* Molecular Dynamics Performed in the Complete Basis Set Limit," *J. Chem. Phys.* **126**, 164501-1–164501-16 (2007).
26. A. Cohen, P. Mori-Sanchez, and W. Yang, "Development of Exchange-Correlation Functionals with Minimal Many-Electron Self-Interaction Error," *J. Chem. Phys.* **126**, 191109-1–191109-5 (May 2007).
27. J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé, "NAMD: Biomolecular Simulation on Thousands of Processors," *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, MD, 2002, pp. 1–18.
28. M. Tuckerman, D. Yarne, S. Samuelson, A. Hughes, and G. Martyna, "Exploiting Multiple Levels of Parallelism in Molecular Dynamics Based Calculations Via Modern Techniques and Software Paradigms," *Comp. Phys. Comm.* **128**, No. 1/2, 333–376 (2000).
29. R. V. Vadali, Y. Shi, S. Kumar, L. V. Kale, M. E. Tuckerman, and G. J. Martyna, "Scalable Fine-Grained Parallelization of Plane-Wave-Based *Ab Initio* Molecular Dynamics for Large Supercomputers," *J. Comp. Chem.* **25**, No. 16, 2006–2022 (2004).
30. J. Hutter and A. Curioni, "Car–Parrinello Molecular Dynamics on Massively Parallel Supercomputers," *Chem. Phys. Chem.* **6**, No. 9, 1788–1793 (2005).
31. F. Gygi, "Large-Scale First-Principles Molecular Dynamics: Moving from Terascale to Petascale Computing," *J. Phys. Conf. Ser.* **46**, 268–277 (2006).
32. L. V. Kale and S. Krishnan, "Charm++: Parallel Programming with Message-Driven Objects," *Parallel Programming Using C++*, G. V. Wilson and P. Lu, Eds., MIT Press, 1996, pp. 175–213.
33. H. Lee, M. Tuckerman, and G. Martyna, "Efficient Valuation of Nonlocal Pseudopotentials via Euler Exponential Spline Interpolation," *Chem. Phys. Chem.* **6**, No. 9, 1827–1835 (2005).
34. D. Yarne, M. E. Tuckerman, and G. J. Martyna, "A Dual Length Scale Method for Plane-Wave-Based, Simulation Studies of Chemical Systems Modeled Using Mixed *Ab Initio*/Empirical Force Field Descriptions," *J. Chem. Phys.* **115**, No. 8, 3531–3539 (2001).
35. J. Hutter, M. Tuckerman, and M. Parrinello, "Integrating the Car–Parrinello Equations. III. Techniques for Ultrasoft Pseudopotentials," *J. Chem. Phys.* **102**, No. 2, 859–871 (1995).
36. S. Kumar, C. Huang, G. Zheng, E. Bohm, A. Bhatele, J. C. Phillips, H. Yu, and L. V. Kalé, "Scalable Molecular Dynamics with NAMD on the IBM Blue Gene/L System," *IBM J. Res. & Dev.* **52**, No. 1/2, 177–188 (2007, this issue).
37. M. Born and K. Huang, *Dynamical Theory of Crystal Lattices*, Oxford University Press, New York, 1954.
38. M. Sharma, Y. Wu, and R. Car, "*Ab Initio* MD with Maximally Localized Wannier Functions," *Intl. J. Quant. Chem.* **95**, No. 6, 821–829 (2003).
39. J. W. Thomas, R. Iftimie, and M. E. Tuckerman, "Field Theoretic Approach to Dynamical Orbital Localization in *Ab Initio* Molecular Dynamics," *Phys. Rev. B* **69**, No. 12, 125105-1–125105-17 (2004).
40. R. Iftimie, J. W. Thomas, and M. E. Tuckerman, "On-the-Fly Localization of Electronic Orbitals in Car–Parrinello Molecular Dynamics," *J. Chem. Phys.* **120**, No. 5, 2169–2181 (2004).
41. G. H. Wannier, "The Structure of Electronic Excitation Levels in Insulating Crystals," *Phys. Rev.* **52**, No. 3, 191–197 (1937).
42. L. V. Kalé, "The Virtualization Model of Parallel Programming: Runtime Optimizations and the State of Art," *Proceedings of the LACSI 2002*, Albuquerque, 2002.
43. C. E. Cramer and J. A. Board, "The Development and Integration of a Distributed 3D FFT for a Cluster of Workstations," *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000, pp. 121–128.
44. A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, et al., "Overview of the Blue Gene/L System Architecture," *IBM J. Res. & Dev.* **49**, No. 2/3, 195–212 (2005).

45. S. Chatterjee, L. R. Bachega, P. Bergner, K. A. Dockser, J. A. Gunnels, M. Gupta, F. G. Gustavson, et al., "Design and Exploitation of a High-Performance SIMD Floating-Point Unit for Blue Gene/L," *IBM J. Res. & Dev.* **49**, No. 2/3, 377–391 (2005).
46. N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, et al., "Blue Gene/L Torus Interconnection Network," *IBM J. Res. & Dev.* **49**, No. 2/3, 265–276 (2005).
47. S. Kumar, C. Huang, G. Almasi, and L. V. Kalé, "Achieving Strong Scaling with NAMD on Blue Gene/L," *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, 2006.
48. N. Higham, "Stable Iterations for Matrix Square Roots," *Numerical Algorithms* **15**, No. 2, 227–242 (1997).
49. G. Almasi, C. Archer, J. G. Castaños, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, et al., "Design and Implementation of Message-Passing Services for the Blue Gene/L Supercomputer," *IBM J. Res. & Dev.* **49**, No. 2/3, 393–406 (2005).
50. M. Blocksome, C. Archer, T. Inglett, P. McCarthy, M. Mundy, J. Ratterman, A. Sidelnik, et al., "Design and Implementation of One-Sided Communication for the IBM eServer Blue Gene Supercomputer," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Article No. 120, Tampa, FL, 2006.
51. J. Thar, W. Reckien, and B. Kirchner, "Car–Parrinello Molecular Dynamics Simulations and Biological Systems," *Topics Curr. Chem.* **268**, 133–171 (2007).
52. N. Troullier and J. Martins, "Efficient Pseudopotentials for Planewave Calculations," *Phys. Rev. B* **43**, No. 3, 1993–2006 (1991).

Received March 15, 2007; accepted for publication April 12, 2007; Internet publication January 17, 2008

Eric Bohm *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (ebohm@uiuc.edu)*. Mr. Bohm received a B.S. degree in computer science from the State University of New York at Buffalo in 1992. He worked as Director of Software Development for Latpon Corporation from 1992 to 1995, and next as Director of National Software Development from 1995 to 1996. He worked as Enterprise Application Architect at MEDE America from 1996 to 1999 and as an Application Architect at WebMD from 1999 to 2001. Following a career shift toward academia, he joined the Parallel Programming Lab at University of Illinois at Urbana-Champaign in 2003. His current focus as a Research Programmer is on optimizing molecular dynamics codes for tens of thousands of processors.

Abhinav Bhatele *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (bhatele2@uiuc.edu)*. Mr. Bhatele received a B.Tech. degree in computer science and engineering from Indian Institute of Technology, Kanpur, India, in 2005. He is a Ph.D. student at the Parallel Programming Lab at the University of Illinois. His research is centered on topology-aware mapping and load balancing for parallel applications. He is a co-developer of the molecular dynamics applications NAMD and LeanCP, developed at the Parallel Programming Lab.

Laxmikant V. Kalé *Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801 (kale@uiuc.edu)*. Professor Kalé has been working on various aspects of parallel computing, with a focus on enhancing performance and productivity via adaptive runtime systems. His collaborations involve the widely used biomolecular simulation program NAMD, which won the Gordon Bell Prize at the Supercomputing Conference in 2002. Other collaborative work focuses on computational cosmology, quantum chemistry, rocket simulation, space-time meshes, and other unstructured mesh applications. His group successfully distributes and supports software embodying his research ideas involving Charm++, adaptive MPI, and the ParFUM framework. Professor Kalé received the B.Tech. degree in electronics engineering from Benares Hindu University, Varanasi, India, in 1977, and an M.E. degree in computer science from Indian Institute of Science in Bangalore, India, in 1979. He received a Ph.D. degree in computer science from the State University of New York, Stony Brook, in 1985. He worked as a scientist at the Tata Institute of Fundamental Research from 1979 to 1981. He joined the faculty of the University of Illinois at Urbana-Champaign as an Assistant Professor in 1985, where he is currently employed as a professor.

Mark E. Tuckerman *Department of Chemistry and Courant Institute of Mathematical Sciences, New York University, New York, New York 10003 (mark.tuckerman@nyu.edu)*. Professor Tuckerman obtained his Ph.D. degree in physics from Columbia University in 1993. From 1993 to 1994, he held an IBM postdoctoral fellowship at the IBM Forschungslaboratorium in Rüschlikon, Switzerland, and from 1995 to 1996, he held an NSF postdoctoral fellowship in advanced scientific computing at the University of Pennsylvania in Philadelphia. He is currently an Associate Professor of Chemistry and Mathematics at New York University. His research interests include reactions in solution, organic reactions on semiconductor surfaces, dynamics of molecular crystals, development of the methodology of molecular dynamics, including novel techniques for enhancing conformational sampling and prediction of free energies in biological systems, and the development of new approaches to

electronic structure and *ab initio* molecular dynamics calculations. In 2005, he received the Wilhelm Friedrich Bessel Research Prize from the Alexander von Humboldt Foundation.

Sameer Kumar *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sameerk@us.ibm.com)*. Dr. Kumar received a B.Tech. degree in computer science from Indian Institute of Technology, Madras, India, in 1999, and M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign. His Ph.D. thesis focused on optimizing communication for massively parallel processing. He is currently a Research Staff Member at the IBM T. J. Watson Research Center and is working on the Blue Gene* Project. His research interests include scaling parallel applications to massively parallel machines and next-generation interconnection network design. He coauthored a paper on scaling the molecular dynamics program NAMD, and it was one of the winners of the Gordon Bell Prize at the 2002 Supercomputing Conference.

John A. Gunnels *IBM Research Division, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (gunnels@us.ibm.com)*. Dr. Gunnels received his Ph.D. degree in computer science from the University of Texas at Austin. He joined the Mathematical Sciences Department of the IBM T. J. Watson Research Center in 2001. His research interests include high-performance mathematical routines, parallel algorithms, library construction, compiler construction and verification, and graphics processors. Dr. Gunnels has coauthored several journal papers and conference papers on these topics and has been the recipient of three Outstanding Technical Achievement Awards, two Gordon Bell Prizes, and the Gerstner Award for Client Excellence.

Glenn J. Martyna, *Physical Sciences Division, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (martyna@us.ibm.com)*. Dr. Martyna received his Ph.D. degree in chemical physics from Columbia University and then became an NSF Postdoctoral Fellow in computational science and engineering at the University of Pennsylvania. He was a tenured faculty member at Indiana University, Bloomington, before joining IBM. Dr. Martyna's research has focused on atomistic modeling of chemical, biological, and materials processes. He is well known for developing novel techniques that markedly increase the speed and efficiency of computer simulations, as well as for applying the methods to investigate important phenomena.