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

Eric Bohm, Abhinav Bhatele, and Laxmikant V. Kale
Department of Computer Science
University of Illinois, 201 N. Goodwin Avenue
Urbana, IL 61801-2302

Mark E. Tuckerman
Department of Chemistry and Courant Institute of Mathematical Sciences
New York University, 100 Washington Square East
New York, NY 10003

Sameer Kumar, John A. Gunnels and Glenn J. Martyna
IBM Thomas J. Watson Research Center, PO Box 218
Yorktown Heights, NY 10598

Abstract

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 method *(CPAIMD) which typically embodies a theoretically exact* ab initio *technique called Kohn-Sham Density Functional Theory, evaluated using an approximate functional and a finite basis set of plane-waves, is widely used to study small systems containing on the order of 10-10^3 atoms. However, CPAIMD's impact has been limited until recent efforts by ourselves and others due to difficulties inherent in 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 multiple concurrent sparse 3D-Fast-Fourier Transforms (3D-FFTs), non-square matrix multiplications and few concurrent dense 3D-FFTs all of which are known to require large interprocessor data movement when efficiently parallelized. Using Charm++, a parallel programming language and runtime system and the processor virtualization techniques it enables, 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. Interleaving is enhanced through both architecturally independent methods and network topology aware mapping techniques. Algorithmic and Blue Gene/L 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 1024 electronic states) in order to demonstrate fine grained parallelism. The largest systems studied scaled well across the whole machine (20480 nodes.)*

Introduction

As the computational power of large parallel computers has increased [1], the efficiency of the modeling methods has improved correspondingly, placing increasing demands on parallel programming techniques. Therefore, the development of strategies capable of scaling the algorithmically complex, multiphase methods of today's scientifically important applications to large numbers of processors, is relevant. Herein, we present, in form appropriate for the non-expert, a description of a fine grained parallel implementation of the Car-Parrinello *ab initio* molecular dynamics (CPAIMD) algorithm [2, 3, 4, 5, 6], which has been used to address key chemical and biological processes as well as to examine important problems in materials science, biophysics, nanotechnology, and solid-state physics [7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. The term fine grained indicates that the implementation decomposes the computational work into small parts such that small systems can scale to large numbers of processors. The 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 solving Newton's equations of motion numerically 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 describe them. Typically CPAIMD implementations employ an (in principle) exact *ab initio* technique known as Kohn-Sham Density Functional Theory[17, 18], the practical implementation of which requires the use of an approximate functional[19, 20, 21] and a finite basis set, e.g., plane-waves[2, 3, 4, 5, 6]. It should be noted, however, that the quality of both the basis sets[22, 23, 24, 25] and the functionals[26] employed are continually improving.

From a computational point of view, the CPAIMD method involves many phases, including multiple concurrent sparse 3D-Fast-Fourier Transforms (3D-FFTs), non-square matrix multiplications and several concurrent dense 3D-FFT computations that possess non-trivial dependencies quite different from those found in a classical molecular dynamics computation [27]. The parallel 3D-FFTs are, themselves, communication intensive due to all-to-all interprocessor communication patterns inherent in their computation (e.g. all processors which have a portion of the data set to be transformed must communicate). The efficient concurrent execution of hundreds of parallel 3D-FFTs introduces a further challenge. In order to switch between phases, large data movement between processors that generates relatively little computation must be orchestrated. In general, parallelization of CPAIMD's phases necessitates complex trade-offs between memory, load balance of work across processors, and interprocessor communication costs. Basic MPI-based implementations of CPAIMD, such as that 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 be roughly equal to the number of electronic states in the system. In contrast, recent efforts by ourselves and others, have overcome the states equal processors barrier [29, 30, 31].

In this paper, we present an overview for the non-expert of fine grained parallelization strategies for the CPAIMD method. Parallel scaling on processor numbers greater than 30 times the number of electronic states in small systems consisting of 24 to 768 atoms (32 to 1024 electronic states) is achieved. The 768 atom/1024 state system scaled well on all 20480 nodes of the Blue Gene/L torus network supercomputer at the IBM T. J. Watson Research Center. Our implementation employs the Charm++ parallel programming language and runtime system [32]. Charm++ offers the benefits of over-partitioning via migratable objects a concept whose implementation involves decomposing the problem into many more discretizations than available processors and placing these parallel objects on available processors at startup and allowing the runtime system the freedom to adjust their position 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 Charm++'s ability to enable the interleaving

of computation and communication and to simplify the effective, topologically cognizant mapping of work to processors which is key to the efficient use of Blue Gene/L's 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 whether with or without a torus network.

This short paper is organized as follows: The basic CPAIMD algorithm [2, 3, 4, 5, 6] is briefly reviewed for completeness so that its parallelization can be understood. We highlight new Euler Exponential Spline (EES) based approaches to basic elements [33, 34] of the CPAIMD technique that are key to our parallel scaling, allow improved scalar performance, and have not been discussed in previous reviews [2, 3, 4, 5, 6, 29]; one of the EES based approaches has not been described previously in the literature. It is stressed that the adoption of a CPAIMD method based on Ref. [35] permits the use of many highly efficient routines and, furthermore, allows our non-standard but effective CPAIMD equations of motion to be derived from a simple Hamiltonian formalism. Next, the parallelization of the serial algorithm is given following the approach outlined in an earlier paper [29] which led to scaling on 1k processors of a 32 water molecule system. 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++, are discussed. The architecture of Blue Gene/L is then described followed by a discussion of the CPAIMD optimizations motivated by Blue Gene/L's design. The mapping of the CPAIMD's parallel objects to physical processors in the 3D torus network is then given. Established Charm++ techniques proven effective in scaling classical molecular dynamics on Blue Gene/L have been applied where appropriate [36]. Last, scaling results are presented for liquid water in system sizes consisting of from 8 to 256 molecules under 3D-periodic boundary using standard parameters and a generalized gradient approximate (GGA) functional, specifically, the BLYP 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 Kohn-Sham (KS) formulation of DFT used here which is also formally exact, the electron density, $\rho(\mathbf{r}) = \rho(x, y, z)$, 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 said to occupy each state. Unfortunately, the exact functional is not known and an approximate functional must be employed. In this work, support is provided for an arbitrary generalized gradient corrected approximate functional of the type given in the following references [19, 20, 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. Last, 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 a two hydrogen ions of charge +1e, is treated as an 8 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, and, thus, 8 electrons are needed to neutralize the system. Rather than carry forward the tedious nomenclature "ion-core", the term 'atom' will be used, allowing the reader to determine the "ion-core" or valence charge by context.

Since KS DFT is a theory for non-interacting 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 non-interacting electrons, the Coulomb

repulsion or *Hartree* energy between negative charge clouds $\rho(\mathbf{r})$ and $\rho(\mathbf{r}')$ or the *Hartree*, 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) non-local* energy. This term should not be confused with non-local 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 beauty of the CPAIMD method is that it avoids this discontinuous process and allows the two elements to occur simultaneously, enabled by a classical *adiabatic principle* [37, 2]. Using a variant of the non-orthogonal state dynamics of Ref. [35] 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 ($H_{fict-tot} = H_{atom} + H_{fict-electron}$) from which simple equations of motion naturally arise. Orthogonality constraints are absent due to the use of the non-orthogonal state based approach (see below). 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 Fig. 1 and below. More complete reviews with many details are given in Refs. [2, 3, 4, 5, 6]. The new EES based methods are described in more technical detail in Refs. [33, 34].

A set of n_s, electronic *states* is introduced which are represented as a set of (complex) Fourier coefficients, $\Psi(s, g_x, g_y, g_z)$, on a regular 3-dimensional "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 rather, a sphere of radius 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 $\sim N_{atom}^2$.

In the course of the computation, each state switches between its g-space and real-space representation via 3D-FFTs of size $N \times N \times N$ (Phases I and VI). 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, 39, 40] based on localized states [41] are beyond the current discussion.) 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 $N_{atom}^2 \log N_{atom}$, dominated by the 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 $N_{atom} \log N_{atom}$. Note, 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 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 and g-space scales like N_{atom}.

Phases III and IV compute the Hartree, external and exchange-correlation energies, as well as the Kohn-Sham (KS) potential, $v_{ks}(x, y, z)$. 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 additional 3D-FFTs to compute. The external energy is computed via $2(N_{atom-type} + 1)$ 3D-FFTs within an Euler Exponential Spline (EES) formalism analogous to that of

Refs. [33, 34]. Here, $N_{atom-type}$ is the number of atom types present in the system (e.g. if only carbon, hydrogen, and nitrogen atoms are present, then there are 3 atom types.). These 3D-FFTs are 1.4x larger on edge ($1.4N$) 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 (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) non-local energy and the kinetic energy of non-interacting electrons, Phase IX, can be computed independently from the above work as depicted in Fig. 1. The non-local energy and forces are computed using the EES technique described in Ref. [33]. Briefly, each state object performs $2N_{atom-type}$ 3D-FFTs if, for example, only s-wave non-locality is considered. The FFTs are smaller on edge ($\approx 0.7N$) than those employed to compute the density. The required scalar work scales as $N_{atom}^2 \log N_{atom}$ while the memory requirement scales as N_{atom}^2. The 3D-FFTs are performed concurrently as the non-local energy does not couple the states together but rather couples each state, individually, to all the atoms. The kinetic energy of non-interacting electrons is expressed (in the computer science parlance) as a "point by point multiply" and reduction operation whose computation scales as $\sim N_{atom}^2$ (e.g. $(\hbar^2/2m_e) \sum_{g_x,g_y,g_z \epsilon |\mathbf{g}|<g_{cut}} \sum_s f_s g^2 |\Psi(s, g_x, g_y, g_z)|^2$ where $\hbar$ is Planck's constant, h divided by 2π and m_e is the mass of the electron).

The forces from the non-local energy and the kinetic energy of non-interacting electrons are added to those obtained from the Hartree, external and 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

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

where f_s is the occupation of each state and the indices are as above. Eq. (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 $\sim N_{atom}^3$ 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 Eq. (1). Therefore, two orthonormalization steps are performed (Phase VII and VIII). The first step regularizes the forces which are then used to evolve the states. The second step corrects the small deviations of the new states from orthogonality/normalization constraints. Both steps scale as $\sim N_{atom}^3$. 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 Ref [35] simply requires a change in the order of operations and an additional matrix multiply as discussed in more detail in Ref [35] and below.

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

Parallelization: parallel operations and data decomposition

The *processor virtualization* approach [42] embodied in the Charm++ parallel programming system [43] 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 in terms of virtual processors, only. Either the runtime system or the user can specify/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 issue of decomposition and mapping.

CPAIMD parallel operations and data decomposition are described below. The analysis is based on earlier work [29] which led to scaling on 1k processors of a 32 water molecule system. Here, we highlight opportunities for interleaving communication and computation and reveal bottlenecks relevant to Blue Gene/L. The description of the data decomposition that follows, shows how these opportunities can be exploited on a torus network architecture like Blue Gene/L to generate high scaling. Non-trivial extensions of our earlier work include the use and parallelization of Euler Exponential Spline based techniques and the improved parallelization of the orthogonality work (matrix multiplications) and the 3D-FFT work related to density based computations.

Parallel operations

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 [44]. Our parallel 3D-FFT is performed in 3 stages. First, a set of 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 and third, two sets of 1D-FFTs are computed to generate $\Psi(s, g_x, g_y, z) \rightarrow \Psi(s, x, y, z)$ due to our planewise decomposition of $\Psi(s, x, y, z)$ (see below). Since 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 ≈ 4x compared to a full complex-to-complex 3D-FFT computation. On Blue Gene/L, the required serial 1D-FFTs were performed using the ESSL library The computation/communication ratio, defined as the ratio of the number of floating point operations required to compute the quantity of interest in scalar to the number of bytes that must be communicated to compute the quantity of interest in parallel, is $\sim \log N_{atom}$.

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 Blue Gene/L.

The reduction to form the probability density in real space occurs in phase II. 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 $N_y N$ sub-planes (see also below). That is each of the N planes is further subdivided into N_y parts. Therefore, the density construction requires spawning ($N_y N$) simultaneous reductions involving large data sets ($\approx 10^6$ real numbers). The computation to communication ratio 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 sub-plane decomposition described below. Once in g-space, the Fourier coefficients of the density are used to compute the "Hartree and external energies" via $2(N_{atom-type}+1)$ 3D-FFTs, which can be performed independently. In addition, 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 6 transposes). The electron density and its gradients are employed to compute a second, gradient-dependent contribution to the exchange-correlation energy. The Kohn-Sham potential, $v_{ks}(x,y,z)$, is created by performing 5 more 3D-FFTs (requiring 10 transposes) on data sets generated during the energy computation. The computation to communication ratio in this phase is $\log N_{atom}$.

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 non-local energy computation.

After performing multiple 3D-FFTs (phase VI) on the product of the Kohn-Sham 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 computation to communication ratio is again $\log N_{atom}$.

Phase IX requires the computation of the kinetic energy of the non-interacting electrons which can be computed without interprocessor communication and the computation of the non-local interaction between the electrons and the atoms via the new EES method [33]. The latter calculation involves $2N_{atom-type}$ 3D-FFTs of size of $\approx 0.7N$ on edge and is interleaved with Phases II-VI. The memory requirement and the communication/computation ratio 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, a $(n_s \times n_g) \times (n_g \times n_s)$ 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 non-orthogonagal state dynamics the forces are further multiplied by the precompted inverse square root matrix which is available due to the change in the order of operations required by the dynamics method as described below. The modified forces are then employed to evolve the states in both cases.

Under functional optimization, the evolved states are slightly non-orthogonal 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 the orthonormal states, $\Psi^{(\mathrm{new})}(s,g_x,g_y,g_z) = \sum_s T(s,s')\Psi(s',g_x,g_y,g_z)$ 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 non-orthogonal state dynamics, the input non-orthogonal states are diagonalized in exactly the same way and the inverse square matrix stored for use later.

A novel feature of our multiplication scheme is that the resultant matrix is used in a "backward path" to compute the necessary modification to the input data, $\Psi^{(\mathrm{new})}(s,g_x,g_y,g_z) = \sum_s T(s,s')\Psi(s',g_x,g_y,g_z)$ or $F_\Psi(s,g_x,g_y,g_z) \leftarrow F_\Psi(s,g_x,g_y,g_z) - \sum_{s'} \Lambda(s,s')\Psi(s',g_x,g_y,g_z)$ for functional minimization . Hence, the Charm++ chare arrays assigned to perform the multiply do not destroy their input data (see below). The result of the matrix multiplication, after summing contributions from all of g-space to form the S or

Λ matrices (see the decomposition below) and post-processing if necessary (e.g. to form the T-matrix), 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 based on Ref. [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 non-orthogonal states generated. These states are in turn orthogonalized and the entire procedure repeated. Under non-orthogonal state dynamics, a set of input non-orthogonal states is first orthogonalized and the inverse square root matrix stored. Forces are computed on the orthonormal states and the forces on the orthonormal states are transformed to forces on the non-orthonormal states as described above. The non-orthonormal states are then evolved to the next step using the forces and the procedure repeated. This summary omits some of details described elsewhere [35].

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

Data decomposition : enabling parallelism and interleaving

CPAIMD's nine phases are parallelized by decomposing the system into 14 indexed collections of objects, each implemented as a *chare array*. These include the real-space and g-space representations of the electronic states decomposed into the 2D state $\times$ r/g-space collection arrays, $G(s,p)$ $[n_s \times N_g]$ and $R(s,p)$ $[n_s \times N]$, respectively, the real-space and g-space representation of the electronic density decomposed into 1D and 2D chare arrays, $G_\rho(p)$ $[N_{g\rho}]$ and $R_\rho(p,p')$ $[(N/N_y) \times N_y]$, the real-space and g-space representations required for the external/Hartree energy work decomposed as the density but with an additional atom type index, $G_{HE}(p,a)$, $[N_{gHE} \times n_{atom-type}]$ and $R_{HE}(p,p',a)$ $[(1.4N/N_y) \times N_y \times n_{atom-type}]$ with $n_{atom-type} \leq N_{atom-type}$, the real-space and g-space representations required for the non-local work decomposed as the states, $G_{nl}(s,p)$ $[n_s \times N_g]$ and $R_{nl}(s,p)$ $[n_s \times 0.7N]$. The orthogonality related computation is decomposed into two 4D g-space based arrays (called pair calculators) and a 2D auxiliary chare (called ortho) described below. The computational work and memory are always well balanced within density and state chares. Orthogonality is inherently more expensive than all other elements in large systems.

The orthogonality phase presented in Fig. 2, is performed by a set of 4 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 array are designed to compute the matrix multiplications, $(n_s \times n_g) \times (n_s \times n_g)$, which 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 of size $sgrain \times sgrain$ where $sgrain = (n_s/N_s)$. The p and p' indices control the decomposition of g-space and the $sgrain \times sgrain$ portions overlap matrix are generated by performing a reduction over the pair calculator chare indices (p,p') at fixed s, s'. Here, N_g is exactly the same as above and N'_g allows for a finer decomposition of g-space .e.g. 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 later. The reduction to form the S and Λ matrices and multicasts of the T and Λ matrices back to the P_c chare arrays (form 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 mapping section. The matrix multicasts, in particular, can be further optimized using a rectangular multicast scheme enabled by topologically aware mapping described later in the text.

We note the memory requirement of CPAIMD scales like $\sim N_{atom}^2$ in scalar. In the parallel implementation given above, the memory requirement scales like $\sim N_{atom}^{2/3}$ per processor due to the planewise decomposition. This assumes that we permit the number of processors used in the computation to increase like $\sim N_{atom}^{4/3}$. At fixed problem size, the memory requirement per processor decreases until the number of processors, N_{proc} exceeds Nn_s ($N_{proc} > Nn_s$). 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 is $\approx$ 100 Gigabytes using the non-local EES method, a computation which "squeezes" onto 512 Blue Gene/L processors but is easily accommodated on $N_{proc} \geq 1024$ in co-processor 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 virtual processors to represent the states in real-space, and about as many virtual processors to hold the non-zero points in g-space. In addition, there are 500 virtual processors representing density in real-space and reciprocal space, respectively (see Fig. 1). Older MPI based applications, such as that by two of the authors [28], scale this benchmark to 128 processors ($\approx n_s$).

The Blue Gene/L architecture

The Blue Gene/L machine [45] is a low power massively parallel supercomputer with 20480 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 Blue Gene/L core is enhanced through a second floating point unit, called the double hummer [46], resulting in a peak performance of 2.8 GF/s per core. The second floating point unit is only usable with 16 byte aligned inputs. Linear algebra libraries like BLAS can make use of the double hummer unit and optimized sequential libraries are utilized as much as possible by our application.

Blue Gene/L has a 3D torus interconnection network [47] for messaging with a favorable network bytes/flop ratio. The network has low messaging latency and good performance for short messages. As the torus network has limited bisection bandwidth, localizing communication improves performance.

Message passing on Blue Gene/L is performed by the core itself. Packets are sent by writing application data to memory mapped to a First In First Out queue (FIFO). 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++. Since the PowerPC 440 core can have 3 loads and 3 stores outstanding, 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. During phases in which messages are sent to far neighbors, the data rate is restricted by/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 (1) the 3D-FFT phase used to create the density by interleaving computation and communication of the many 3D-FFTs, (2) the multicast/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 non-local related computation/communication with the density multicasts to/from the states, and intra-density 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/out of the pair-calculator chares to/from the g-space state chares so as mitigate the natural barrier formed by the orthogonalization work, and (7) the communication into/out of the pair-calculator chares to/from the ortho chares which post-process the overlap matrices, is required. In order to accomplish these 7 tasks, (1) the scalar performance, (2) the decomposition of work to chare array elements, (3) the topologically cognizant mapping of chare array elements to processors, (4) interprocessor communication through use of the Blue Gene/L machine layer, were all systematically improved. In this process, we quite naturally borrowed any applicable general optimizations from previous work by the CS authors [48].

Scalar optimization:

The scaling of the non-local energy computation with system size was reduced from N_{atom}^3 to $N_{atom}^2 \log N_{atom}$ 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 $N_{atom} \log N_{atom}$ via an EES method. The use of EES techniques is unique to our CPAIMD implementation.

The orthogonalization scheme for functional optimization and non-orthogonal state dynamics [35] employed herein requires the computation of an inverse matrix square root. A 2nd order iterative method involving three $(n_s \times n_s)$ matrix multiplications per cycle was implemented [49] which has lower 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 so as 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 8 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 5 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 so as 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 virtual processors to real processors. Due to 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 virtual processor 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, as it is desired 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 (Fig. 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 non-local (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 to nearby processors in the network. We will 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.

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 which must be traversed from sender to destination in the torus or the 'hop count'. Network contention resulting from link saturation can severely damage the performance of a communication intensive application like CPAIMD. The most efficient use of bandwidth, and therefore the best performance for CPAIMD, comes 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 allowed the torus to be completely tiled by the prisms. The number of prisms is the number of planes, and was always chosen at configuration time to be a multiple of at least two torus dimensions (power of 2 choices suffice on natural BG/L 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 Blue Gene/L torus network. The matrix-multiplication/pair calculator virtual processors, $P_c(s1,s2,p,p')$, could then be placed starting at the centroid of the 3D object formed by $G(s1,p) + \ldots G(s1+sgrain,p) + G(s2,p) + \ldots + G(s2+sgrain,p)$ within the prism for each plane p, thereby, minimizing the hop-count for the orthonormalization input and output. The ortho VPs, $O(s1 \ldots s1+sgrain, s2 \ldots s2+sgrain)$, can likewise be placed based on the $P_c(s1 \ldots s1+sgrain, s2 \ldots s2+sgrain, *, *)$ processor list, but this only led to ideal hop-count for elements on the diagonal $s1 = s2$, motivating further communication optimizations described in the "Interleaving communication and computation" section.

The real space VPs $R(s,p)$ were placed with one state per flat prism formed by $G(s,*)$. The previously described longitudinal state-wise 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 "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 non-local EES chares $R_{EES}(s,*)$ were mapped based on the 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 which might otherwise exhibit communication patterns spanning the entire torus in a naive mapping scheme, were instead confined to communicate within prism shaped sub-tori. This, in turn, balanced the overall communication load throughout the available torus. Performance degradation due to network contention and "hot-spots" was avoided, while each phase remained 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 wrap around to reduce the maximum hop-count within each prism to approximately half the length of the largest spanned dimension, and 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 (3 integers per VP) in proportion to the number of VPs, which was a few megabytes in the largest system studied. Runtime cost of creating the most complex of these maps is $pn^2log(n)$ where n is the number of VPs and p 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, once created, maps were stored once and reloaded in subsequent runs to minimize restart time. Offline creation of maps using even more sophisticated techniques and adapting these ideas to other topologies is an area of future work.

The Charm++ Native Layer for Blue Gene/L :

The MPI communication software stack on Blue Gene/L [50] has two messaging protocols, *eager* and *rendezvous*. As MPI requires message ordering, eager protocol [50] packets are routed deterministically, a protocol which works well only for short messages. For long messages, the rendezvous protocol is used; a rendezvous packet is sent to the destination where the MPI tags are matched and an acknowledgement is sent back. Upon receiving the acknowledgement, the source sends the packets to the destination with adaptive routing for higher network throughput. The performance of the rendezvous protocol is effected when the rendezvous packets are delayed by other packets. This approach prevents some sources from making progress on their messages.

Charm++, in contrast, 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 (see Results section for a comparison). Furthermore, the flexibility of Charm++ allows it to be built on top of a lower level message layer than MPI; the Charm++ native layer is built on the Blue Gene Message Layer [51], which reduced message

overhead and permitted the development of the new more efficient protocol described in the following section.

Last, we note that the per node memory of the Charm++ runtime system on Blue Gene/L 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 is not a problem for Charm++. However, each packet must carry the size of the message to allocate a buffer for the entire message. The packet most 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 1MB 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 back to the source to send the next 16 messages. We have observed that this protocol significantly improves performance of Charm++ applications on Blue Gene/L.

Optimized multicasts:

There are several multicast operations in CPAIMD. In Fig. 1, phases V and VIII involve each-to-many communication operations with large messages. Two techniques are used to optimize this communication: *(1) 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 Blue Gene/L torus [47]. *(2) Rectangular Multicast on Blue Gene/L*: The destination objects of a multicast can sometimes be mapped onto a rectangular prism, such as in the orthonormalization phases. Here, the deposit-bit broadcast capabilities of the network hardware, where each packet can be sent to all the destinations on a line, is exploited, 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 based on the destination processor rank. When the bucket for a particular destination fills up, the messages are combined and sent to the destination as one message, resulting in a lower per message overhead. A bucket size of 5 was found to be optimal.

Interleaving communication and computation:

The density (phases III and IV) and non-local (phase IX) computations can be interleaved because these phases involve 3D-FFTs and, hence, all-to-all communication. Again, on Blue Gene/L, the processor, itself, has to 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 VPs involved, here, non-local and density chares, to share the same physical processor on smaller machines.

In order to enable interleaving, "network progress calls" are inserted into application's core compute loop every tens of thousands of processor cycles [48]. 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 the network drains. This approach was borrowed from other work by the CS authors [48].

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 against the original data as they arrive (The granularity in the state indices of O is typically finer than 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 described in the text on Blue Gene/L was investigated using liquid water as a test case due to the importance of aqueous solutions in biophysics[52]. A single water molecule, H_2O, consists of three atoms per molecule, 2 atom types and possesses 4 doubly occupied valence electron states. All the computations to be discussed below were performed using the standard g-space spherical cutoff radius on the states of $|\mathbf{g}|^2_{cut} = 70$Ry at the Γ point (1 k-point), three dimensional periodic boundary conditions, the BLYP generalized gradient corrected density functional[19, 20] and Martins-Troullier type pseudopotentials[53]. More complex systems with up to 5 atom types are currently running using our framework and will be discussed in a later publication; the framework is not limited to 5 atoms types but that is simply the maximum number attempted to date.

We note that water is an extremely common CPAIMD simulation target[10, 11, 12]. Other groups developing fine grained parallel CPAIMD software[30, 31], have not as 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 provide a set of unambiguous timings for a well studied system, liquid water, using standard parameters, on a well known architecture in a prominent publication to which other groups can compare their results in the future. (Two year old scaling data on 32 water molecules using the non-standard cutoff, $|\mathbf{g}|^2_{cut} = 100$Ry, on small processors numbers, $N_{proc} \leq 512$, has been presented by others[30]. The one published data point for the 32 molecule water system using standard parameters,$|\mathbf{g}|^2_{cut} = 70$Ry, is 0.35s/step on 512 processors[30]. This likely does not represent the current status of their project.).

Specifically, six liquid water systems consisting of 8, 16, 32, 64, 128, and 256 molecules and 32, 64, 128, 256, 512 and 1024 doubly occupied electron states, respectively, were selected for study in order to probe the performance of our CPAIMD application in 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 waters, the 3D-FFT work and orthogonalization have more balanced workloads. In Ref. [31], a 1000 atom system of solid molybdenum under 3D periodic boundary conditions was studied using from 1 to 8 k-points and a cutoff of 44Ry. Compared to our largest benchmark, 256 waters using 1 k-point (the Γ-point), the number of non-zero g-vectors per state is essentially the same but the molybdenum system has $n_s = 6000$ doubly occupied states and is more strongly dominated by orthogonality work than the 256 water molecule system (e.g. $36\times$). There are significantly more non-local electron-atom (ion core) interactions in molybdenum than in water ($250\times$) and these were treated 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 1024 nodes are given for the molybdenum system with 1 k-point in Ref. [31].

Numerical tests were performed 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 many times larger than physical parameter values such as the number of electronic states.

Strong scaling is observed in all cases. Thus, the timing ≈ 0.2s/step is observed for 8 waters at 32 nodes, for 16 waters at 128 nodes, for 32 waters at 512 nodes, and for 64 waters at 4096 nodes in co-processor mode (number of nodes= number of processors). For 128 waters, on 20,000 nodes in co-processor mode a rather promising time of 0.3s/step is achieved. However, in virtual-node-mode for 128 waters using 16,000 nodes and 32,0000 processors we attain a slower timing than using 20,000 nodes in co-processor mode. This motivates future work optimizing our CPAIMD application for large systems on large processor numbers (e.g. more scaling tests on the full Blue Gene/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 0.2s/step increases as N_{atom}^2 at small system size and as N_{atom}^3 at large system size as expected. Good weak scaling is also observed. 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 $N_{proc} \geq 30n_s$. We further point out our time per step for 32 waters is 0.26s/step on 512 node=processors in co-processor mode. Below, some of the more important elements that lead to these results are described.

First, Charm++ can be compiled using MPI as its machine interface or using a Blue Gene/L customized machine interface[36] called the Native Layer. The scaling of the CPAIMD application on the MPI driver was limited (see Table 1 (bottom labeled "MPI")) because MPI adds an additional level of overhead and imposes message ordering which is not necessary or pertinent for this application.

Second, as Blue Gene/L is 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 (bottom labeled "No Topo")) shows the performance degradation if topology specific and relational maps are disabled in the application. In the simple mapping scheme, the work is spread out 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 two reduction in CPU time on most of the 256 water molecule data at large processors; the "No Topo" runs hit the scaling limit before $N_{proc} = 8k$ for the 32 water molecule test case.

Third, we observed that the non-local and density computations had limited parallelism when implemented using N_{atom}^3 and N_{atom}^2 algorithms rather than the $N_{atom}^2 \log N_{atom}$ and $N_{atom} \log N_{atom}$ EES based methods. In Table 1 (bottom 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 greater than 1000. The standard methods could be made 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 was able to obtain scaling of important systems to physical processor numbers equal to 30 times the number of electronic states on IBM's Blue Gene/L. 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 an 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 extend our thanks to Drs. Fred Mintzer, Bob Walkup and the Blue Gene/L team at IBM TJ Watson, as this work would not have been possible without their help and access to the hardware. We would like to thank Dr. Gheorghe Almasi for assistance with the Blue Gene/L message layer. We would also like to thank Philip Heidelberger for technical support on the Blue Gene/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 as the critical initial Bluegene/L porting and scaling research relied upon their support. This work was supported by the National Science Foundation (ITR-0121357 and ITR-0229959).

Laxmikant V Kale Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, IL 61801. (kale@uiuc.edu). Professor Laxmikant Kale has been working on various aspects of parallel computing, with a focus on enhancing performance and productivity via adaptive runtime systems, and with the belief that only interdisciplinary research involving multiple CSE and other applications can bring back well-honed abstractions into Computer Science that will have a long-term impact on the state-of-art. His collaborations include the widely used Gordon-Bell award winning (SC'2002) biomolecular simulation program NAMD, and other collaborations on computational cosmology, quantum chemistry, rocket simulation, space-time meshes, and other unstructured mesh applications. He takes pride in his group's success in distributing and supporting software embodying his research ideas, including Charm++, Adaptive MPI and the ParFUM framework.

L. V. Kale received the B.Tech degree in Electronics Engineering from Benares Hindu University, Varanasi, India in 1977, and a M.E. degree in Computer Science from Indian Institute of Science in Bangalore, India, in 1979. He received a Ph.D. in computer science in from 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.

Glenn J. Martyna, *Physical Sciences Division, IBM T. J. Watson Laboratory, Yorktown Heights, NY (martyna@us.ibm.com)*. Dr. Martyna received his Ph.D. in Chemical Physics from the Columbia University and then became a 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 and applying the methods to investigate important phenomena.

Mark E. Tuckerman *Department of Chemistry and Courant Institute of Mathematical Sciences, New York University, New York, NY 10003*. Professor Tuckerman obtained his Ph.D. in physics from Columbia University in 1993. From 1993–1994, he held an IBM postdoctoral fellowship at the IBM Forschungslaboratorium in Rüschlikon, Switzerland, and from 1995–1996, 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 semi-conductor 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.

Abhinav Bhatele, Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, IL 61820. (bhatele2@uiuc.edu) Abhinav received a B. Tech. degree in Computer Science and Engineering from I.I.T. Kanpur (INDIA) in May 2005. He is a Ph.D. student at the Parallel Programming Lab at the University of Illinois. His research is centered around topology aware mapping and load balancing for parallel applications.

Eric Bohm, Department of Computer Science, Thomas M. Siebel Center, University of Illinois at Urbana-Champaign, Urbana, IL 61820. (ebohm@uiuc.edu). Eric 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 Corp from 1992 to 1995, then as Director of National Software Development from 1995 to 1996. He worked as Enterprise Application Architect at MEDE America from 1996 to 1999. He completed his time in industry as Application Architect at WebMD from 1999 to 2001. Following a career shift towards 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.

References

[1] F. Allen et al. "Blue Gene: A vision for protein science using a petaflop supercomputer." *IBM System Journal*, **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**, 2471–2474, (1985).

[3] G. Galli and M. Parrinello. "Ab initio molecular dynamics : principles and and practical implementation." *Computer Simulation in Materials Science : interatomic potentials, simulation techniques and applications*, **3**, 283, (1991).

[4] M. Payne, M. Teter, D. Allan, T. Aria, and J. Joannopolous. "Iterative minimization techniques for ab initio total energy calculations : molecular dynamics and conjugate gradients." *Rev Mod Phys*, **64**, 1045–1097, (1992).

[5] M. E. Tuckerman. "Ab initio molecular dynamics: Basic concepts, current trends and novel applications." *J Phys Condensed Matter*, **14**, R1297–R1395, (2002).

[6] D. Marx and J. Hutter. "Ab initio molecular dynamics: Theory and implementation." in *Modern methods and algorithms of quantum chemistry* (J Grotendorst (Ed), Forschungszentrum, Juelich, NIC Series), **1**, 301–449, (2000).

[7] A. D. Vita, G. Galli, A. Canning, and R. Car. "A microscopic model for surface-induced diamond-to-graphite transitions." *Nature*, **379**, 523–526, (1996).

[8] G. Galli, R. M. Martin, R. Car, and M. Parrinello. "Melting of diamond at high pressure." *Science*, **250**, 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**, 186402, (2002).

[10] M. Benoit, D. Marx, and M. Parrinello. "Tunneling and zero point enregy 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 x 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*, **102**, no. 19, 6654–6659, (2005).

[16] R. Martonak, D. Donadio, A. R. Oganov, and M. Parrinello. "Crystal structure transformations in SiO2 from classical and ab initio metadynamics." *Nature Materials*, **5**, no. 8, 623–626, (2006).

[17] W. Kohn and L. J. Sham. "Self-Consistent equations including exchange and correlation effects." *Phys Rev*, **140**, A1133, (1965).

[18] R. G. Parr and W. Yang. *Density Functional Theory of atoms and molecules*. Oxford University Press, Oxford, (1989).

[19] A. Becke. "Density-Functional exchange-energy approximation with correct assymptotic 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, (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**, 154507, (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, (2007).

[26] A. Cohen, P. Mori-Sanchez, and W. Yang. "Development of exchange-correlation functionals with minimal many-electron self-interaction." *J Chem Phys*, **126**, 191109, (2007).

[27] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé. "NAMD: Biomolecular Simulation on Thousands of Processors." In *Proceedings of SC 2002*. Baltimore, MD, (2002).

[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**, 333–376, (2000).

[29] R. Vadali, Y. Shi, S. Kumar, L. Kale, M. Tuckerman, and G. 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." In G. V. Wilson and P. Lu, eds., *Parallel Programming using C++*, 175–213. MIT Press, (1996).

[33] H. Lee, M. Tuckerman, and G. Martyna. "Efficient evaluation of nonlocal pseudopotentials via Euler exponential spline interpolation." *Chem Phys Chem*, **6**, no. 9, 1827–1835, (2005).

[34] D. Yarne, M. Tuckerman, and G. 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 .3. 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 Blue Gene/L." *IBM Journal of Research and Development: Applications of Massively Parallel Systems (to appear)*, (2007).

[37] M. Born and K. Huang. *Dynamical Theory of Crystal Lattices*. Oxford University Press, New York, 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**, 125105, (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**, 191, (1937).

[42] L. V. Kalé. "The Virtualization Model of Parallel Programming : Runtime Optimizations and the State of Art." In *LACSI 2002*. Albuquerque, (2002).

[43] L. V. Kale and S. Krishnan. "Charm++: Parallel Programming with Message-Driven Objects." In G. V. Wilson and P. Lu, eds., *Parallel Programming using C++*, 175–213. MIT Press, (1996).

[44] C. Cramer and J. Board. "The Development and Integration of a Distributed 3D FFT for a Cluster of Workstations." *4th Annual Linux Showcase and Conference*, 121–128, (2000).

[45] A. Gara et al. "Overview of the Blue Gene/L System Architecture." *IBM J Res Dev*, **49**, 195–212, (2005).

[46] S. Chatterjee et al. "Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L." *IBM J Res Dev*, **49**, 377–391, (2005).

[47] N. R. Adiga et al. "Blue Gene/L torus interconnection network." *IBM J Res Dev*, **49**, 265–276, (2005).

[48] S. Kumar, C. Huang, G. Almasi, and L. V. Kalé. "Achieving Strong Scaling with NAMD on Blue Gene/L." In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*. (2006).

[49] N. Higham. "Stable iterations for matrix square roots." *Numerical Algorithms*, **15**, 227–242, (1997).

[50] G. Almasi et al. "Design and implementation of message-passing services for the Blue Gene/L supercomputer." *IBM J Res Dev*, **49**, 393–406, (2005).

[51] M. B. et al. "Design and Implementation of One-Sided Communication for the IBM eServer Blue Gene Supercomputer." In *Proceedings of Supercomputing*. (2006).

[52] J. Thar, W. Reckien, and B. Kirchner. "Car-Parrinello molecular dynamics simulations and biological systems." *Topics Curr Chem*, **268**, 133–171, (2007).

[53] N. Troullier and J. Martins. "Efficient pseudopotentials for plane wave calculations." *Phys Rev B*, **43**, no. 3, 1993–2006, (1991).

Tables

Table 1. Parallel performance : Liquid water

CO Mode Native Layer with Optimizations											
Nodes	32	64	128	256	512	1024	2048	4096	8192	16384	20480
Processors	32	64	128	256	512	1024	2048	4096	8192	16384	20480
W8 Time s/step	0.22	0.10	0.082	0.071	0.046	0.026	0.020				
W16 Time s/step	0.73	0.40	0.23	0.15	0.106	0.061	0.041	0.035			
W32 Time s/step	2.71	1.52	0.95	0.44	0.26	0.15	0.11	0.081	0.063		
W64 Time s/step		6.62	3.75	1.88	0.87	0.51	0.31	0.21	0.15		
W128 Time s/step					6.9	2.73	1.40	0.91	0.58	0.37	0.3
W256 Time s/step						16.4	8.14	4.83	2.75	1.71	1.54
VN Mode Native Layer with Optimizations											
Nodes	32	64	128	256	512	1024	2048	4096	8192	16384	20480
Processors	64	128	256	512	1024	2048	4096	8192	16384	32768	40960
W8 Time s/step	0.13	0.11	0.08	0.06	0.028	0.021					
W16 Time s/step	0.46	0.28	0.19	0.13	0.08	0.047	0.035				
W32 Time s/step	1.99	1.40	0.81	0.43	0.174	0.13	0.082	0.067			
W64 Time s/step		9.07	3.38	1.71	0.67	0.38	0.22	0.17	0.15		
W128 Time s/step					3.0	1.48	0.90	0.65	0.48	0.40	0.3
W256 Time s/step							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	1024	2048	4096	8192	16384	20480
Processors	32	64	128	256	512	1024	2048	4096	8192	16384	40960
W32 (MPI) Time s/step						0.33	0.22	0.17	0.12		
W32 (No Topo) Time s/step						0.21	0.23	0.16	0.18		
W32 (No EES) Time s/step						0.22	0.14	0.098	0.082		
W256 (No Topo) Time s/step						28.8	23.0	13.4	6.83	3.40	

Figure captions

Fig. 1 Structure of our implementation. Phases are in Roman numerals.

Fig. 2 Computing the entries of the S matrix. Each chare array element $P_c(s, s', p, p'), s \leq s'$ computes a contribution to an $sgrain \times sgrain$ tile of the S matrix ($sgrain = N_s/2$ above) 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") to produce the desired $sgrain \times sgrain$ tile of S. Here, $0 \leq p \leq 1, p' = 0$.

Fig. 3 Placement of GSpace, RealSpace and density objects on the 3D torus.

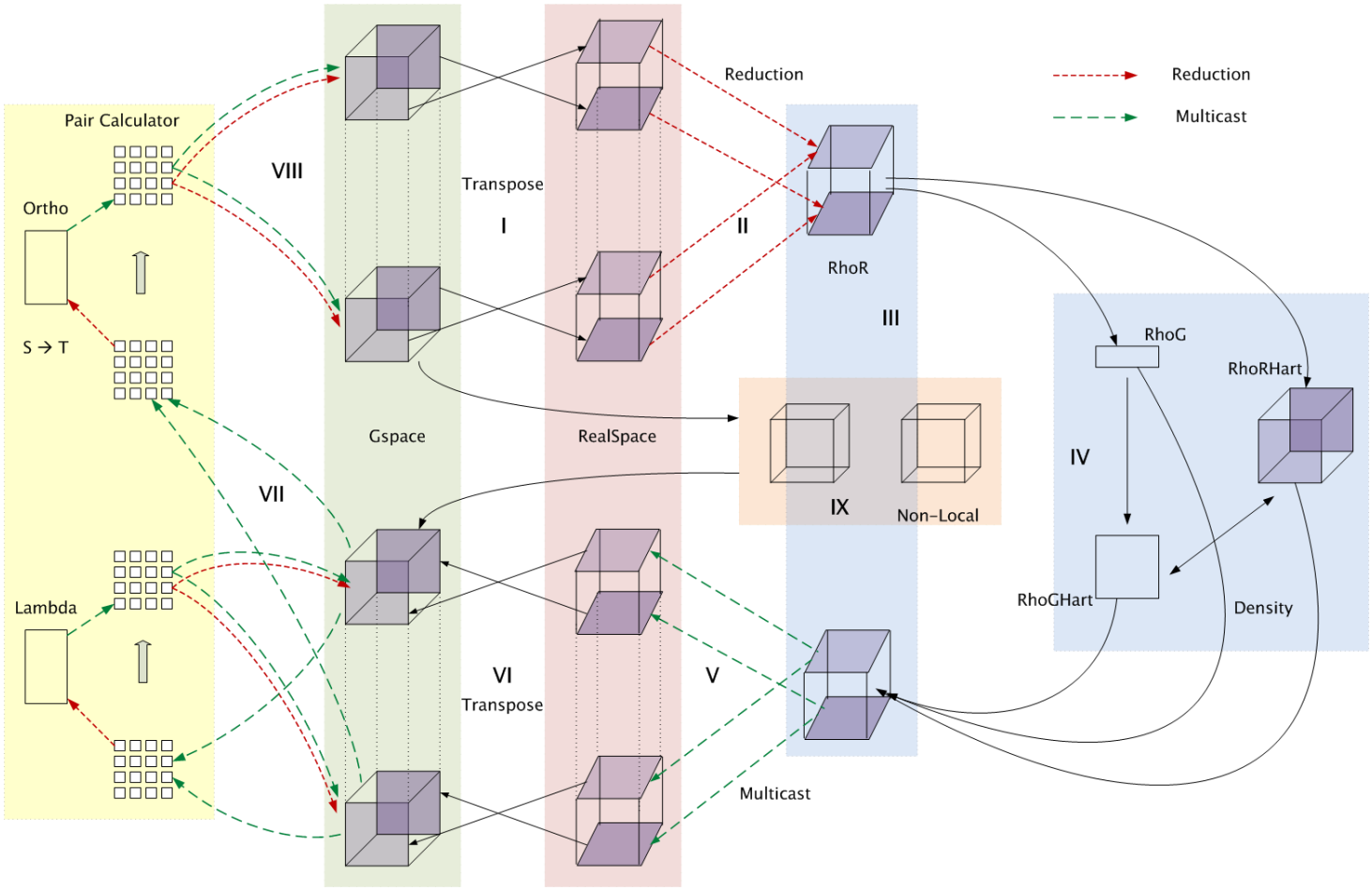

Figure 1. Structure of our implementation. Phases are in Roman numerals.

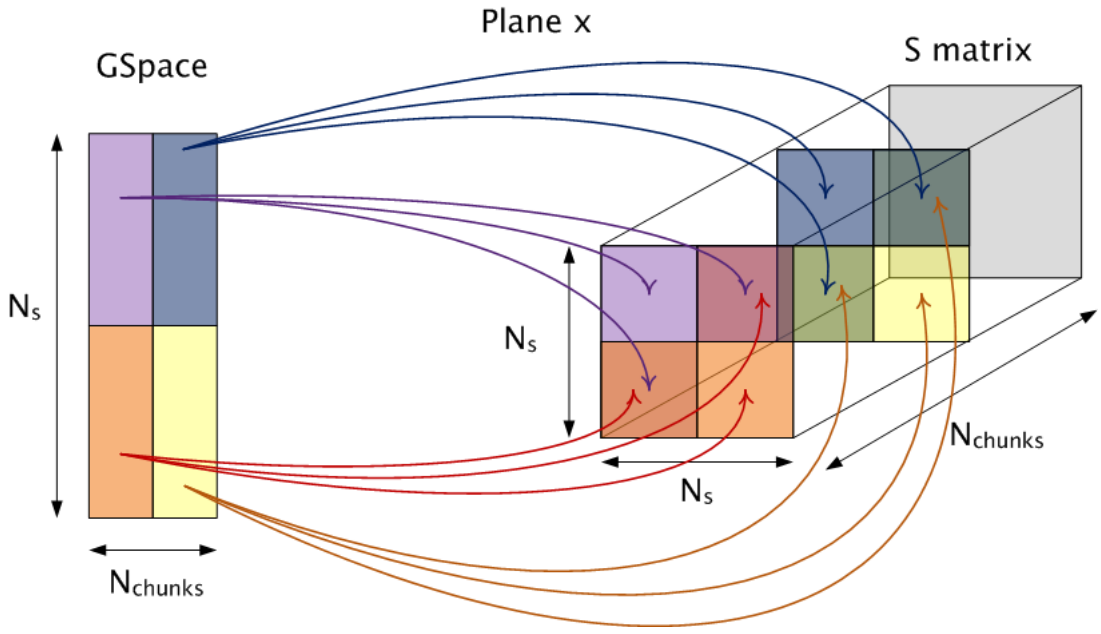

Figure 2. Computing the entries of the S matrix. Each chare array element $P_c(s, s', p, p'), s \leq s'$ computes a contribution to an $sgrain \times sgrain$ tile of the S matrix ($sgrain = N_s/2$ above) for its section of g-space indexed by p, p' after receiving data from the state g-space chares. After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

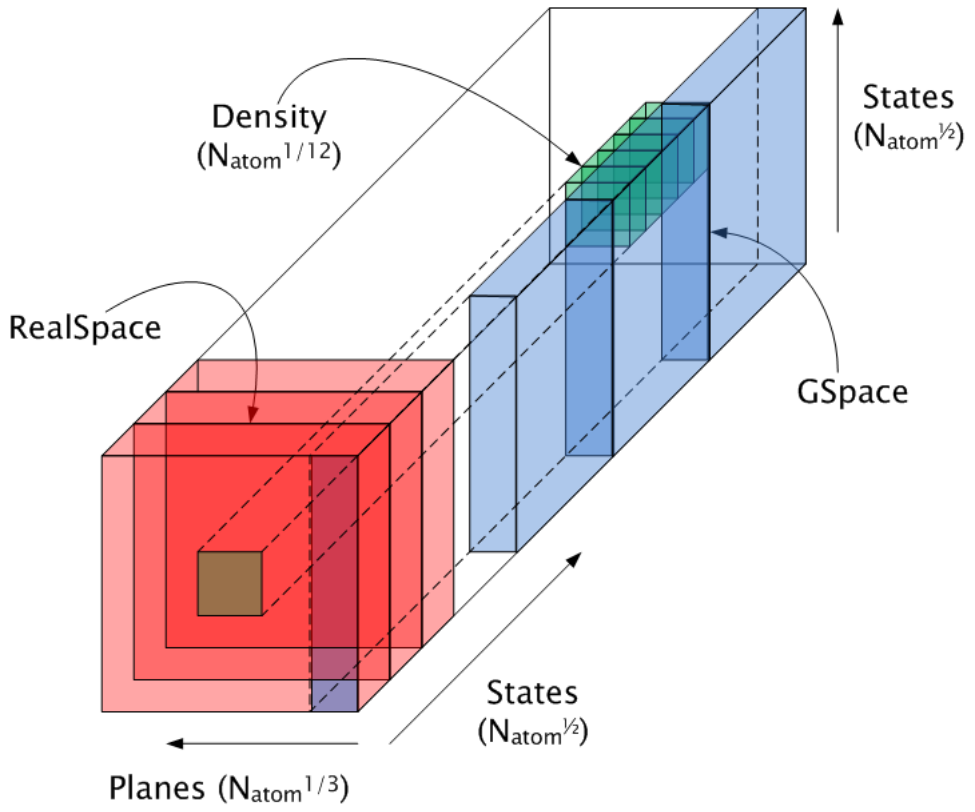

Figure 3. Placement of GSpace, RealSpace and density objects on the 3D torus