Scalable Fine-Grained Parallelization of Plane-Wave–Based *Ab Initio* Molecular Dynamics for Large Supercomputers

RAMKUMAR V. VADALI,[1] YAN SHI,[1] SAMEER KUMAR,[1] LAXMIKANT V. KALE,[1] MARK E. TUCKERMAN,[2] GLENN J. MARTYNA[3]

[1]Department of Computer Science, The Siebel Center, University of Illinois, 201 N. Goodwin Avenue, Urbana, Illinois 61801-2302

[2]Department of Chemistry and Courant Institute of Mathematical Sciences, 100 Washington Square East, New York University, New York, New York 10003

[3]IBM Research, Division of Physical Science, TJ Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598

Received 3 June 2004; Accepted 4 July 2004
DOI 10.1002/jcc.20113
Published online in Wiley InterScience (www.interscience.wiley.com).

Abstract: Many systems of great importance in material science, chemistry, solid-state physics, and biophysics require forces generated from an electronic structure calculation, as opposed to an empirically derived force law to describe their properties adequately. The use of such forces as input to Newton's equations of motion forms the basis of the *ab initio* molecular dynamics method, which is able to treat the dynamics of chemical bond-breaking and -forming events. However, a very large number of electronic structure calculations must be performed to compute an *ab initio* molecular dynamics trajectory, making the efficiency as well as the accuracy of the electronic structure representation critical issues. One efficient and accurate electronic structure method is the generalized gradient approximation to the Kohn–Sham density functional theory implemented using a plane-wave basis set and atomic pseudopotentials. The marriage of the gradient-corrected density functional approach with molecular dynamics, as pioneered by Car and Parrinello (R. Car and M. Parrinello, Phys Rev Lett 1985, 55, 2471), has been demonstrated to be capable of elucidating the atomic scale structure and dynamics underlying many complex systems at finite temperature. However, despite the relative efficiency of this approach, it has not been possible to obtain parallel scaling of the technique beyond several hundred processors on moderately sized systems using standard approaches. Consequently, the time scales that can be accessed and the degree of phase space sampling are severely limited. To take advantage of next generation computer platforms with thousands of processors such as IBM's BlueGene, a novel scalable parallelization strategy for Car–Parrinello molecular dynamics is developed using the concept of processor virtualization as embodied by the Charm++ parallel programming system. Charm++ allows the diverse elements of a Car–Parrinello molecular dynamics calculation to be interleaved with low latency such that unprecedented scaling is achieved. As a benchmark, a system of 32 water molecules, a common system size employed in the study of the aqueous solvation and chemistry of small molecules, is shown to scale on more than 1500 processors, which is impossible to achieve using standard approaches. This degree of parallel scaling is expected to open new opportunities for scientific inquiry.

© 2004 Wiley Periodicals, Inc. J Comput Chem 25: 2006–2022, 2004

Key words: supercomputers; molecular dynamics; AIMD method

Introduction

Modern theoretical methodology, aided by high-speed computing platforms, has advanced to the point that the microscopic details of chemical processes in condensed phases can now be treated on a relatively routine basis. One of the most commonly used theoretical approaches for these studies is the molecular dynamics (MD) method, in which the classical Newtonian equations of motion for a system are solved numerically starting from a prespecified initial state and subject to a set of boundary conditions appropriate for the problem.[1–3] MD methodology allows both equilibrium thermodynamic and dynamical properties of a system at finite temperature to

Correspondence to: G. J. Martyna; e-mail: martyna@us.ibm.com; gmartyna@yahoo.com

Contract grant sponsor: NSF; contract/grant numbers: CHE-012375 and CHE-0310107

Contract/grant sponsor: Camille and Henry Dreyfus Foundation, Inc.; contract/grant number: TC-02-012

Contract/grant sponsor: New York University's Research Challenge Fund (to M.E.T.); contract/grant numbers: NSF 0121357 (to L.V.K.), NSF-0229959 (to G.J.M.), PSC-DMR02000SP (to G.J.M. and M.E.T.)

Contract/grant sponsor: IBM Research (to G.J.M.).

be computed, while simultaneously providing a "window" through which the microscopic motion of individual atoms in the system can be viewed.[4] One of the most challenging aspects of an MD calculation is the specification of the forces. In many applications, these are computed from an empirical model or *force field,* in which simple mathematical forms are posited to describe bond, bend, and dihedral angle motion as well as van der Waals and electrostatic interactions between atoms, and the model parameterized to experimental data at a few state points and/or high-level *ab initio* calculations on small clusters or fragments.[5–7] This approach has enjoyed tremendous success in the treatment of systems ranging from simple liquids and solids to polymers and biological systems such as proteins and nucleic acids.

Despite their success, standard force fields have a number of serious limitations. First, atomic charges are taken to be static parameters, and therefore, electronic polarization effects are neglected. This limitation has long been recognized,[8] and attempts to rectify the problem have been proposed in the form of empirical polarizable models, in which charges and/or induced dipoles are allowed to fluctuate in response to a changing environment.[9–13] Although these models have also enjoyed considerable success, they also have a number of serious limitations, including a lack of transferability and standardization. Second, empirical force fields generally suffer from an inability to describe chemical bond breaking and forming events. The latter problem can be treated in an approximate manner using techniques such as the empirical valence bond method[14] or other semiempirical approaches. However, these methods are, also, not transferable and, therefore, need to be reparameterized for each type of reaction and may, indeed, bias the reaction path in both undesirable and not intuitively obvious ways.

To overcome the limitations of force field-based approaches, one of the most important recent developments in MD, the *ab initio* molecular dynamics (AIMD) method,[15–24] which combines finite temperature dynamics with forces obtained from electronic structure calculations performed "on the fly" as the MD simulation proceeds was, first, conceived.[15] The electronic structure is treated explicitly in AIMD calculations; hence, many-body forces, electronic polarization, and bond-breaking and -forming events are described to within the accuracy of the electronic structure representation employed.

The AIMD method has been used to study a wide variety of chemically interesting and important problems in areas such as liquid structure,[25–27] acid-base chemistry,[28–30] industrial,[31–34] and biological catalysis,[35] geophysical systems,[36,37] and the design and analysis of materials with novel properties. In many of these applications, new physical phenomena have been revealed, which could not have been uncovered using empirical models, often leading to new interpretations of experimental data and even suggesting new experiments to perform.

Not unexpectedly, the power and flexibility of the AIMD methodology comes at the price of a significant increase in computational overhead compared to force field-based approaches. Whereas the latter can currently be employed to routinely simulate systems consisting of 10^4–10^6 atoms and to access time scales on the order of hundreds of nanoseconds or longer, AIMD calculations can currently be employed to routinely simulate systems of just a few tens or hundreds of atoms and to access time scales on the order of tens of picoseconds. The bottleneck in AIMD calculations is clearly the time required to perform the electronic structure calculations. Currently, the most commonly used electronic structure theory in AIMD is the Kohn–Sham formulation of the density functional theory (DFT),[38–40] implemented using a plane-wave basis set expansion of the electronic orbitals. This protocol provides a reasonably accurate description of the electronic structure for many types of chemical environments while maintaining an acceptable computational overhead and constitutes the basis for the original Car–Parrinello formulation of the method (CPAIMD).[15] Finally, AIMD can be combined with force field-based approaches to yield the so-called quantum mechanical/molecular mechanical (or QM/MM) technique,[41–44] which is particularly useful for treating large systems, such as enzymes, where chemical reactivity is localized in a relatively small region of the system.

Given the power of the CPAIMD methodology, it is critical to develop techniques aimed at overcoming the computational bottlenecks that limit its applicability. A complete solution must involve both algorithmic developments and emerging computer architectures. In the present article, the latter issue is addressed by developing scalable fine-grained parallel algorithms for the large-scale computational platforms available today and the, yet larger, architectures of tomorrow such as IBM's BlueGene.[45] Achieving efficient parallel scaling on these machines is a challenging problem. First, the CPAIMD method involves multiple three-dimensional Fast Fourier Transforms (3D FFT). Parallel 3D FFTs are communication intensive due to the all-to-all communication patterns they exhibit. Furthermore, efficient concurrent execution of hundreds or thousands of 3D FFTs is nontrivial. Second, multiplication of large nonsquare matrices is required, an operation that involves movement of large amounts of data among the processors. Parallelization of these tasks necessitates intricate tradeoffs between memory access, load balance, and communication costs.

All previous parallel implementations of CPAIMD employ the following simple protocol[46]: the steps of the calculation are carried out in sequence as in a scalar code and the computational work required by each step partitioned among the physical processors according to a predetermined static scheme. While this type of parallelization can be made effective for small numbers of processors, the scalability of this type of protocol is inherently determined by how fine-grained each step can be made. Eventually, a limit is reached in which the communication overhead exceeds the useful computational work, at which point additional processors can no longer be used effectively. In this article, we introduce a new parallelization strategy for CPAIMD based on the concept of *virtualization,* as embodied in the Charm++ runtime system.[47,48] Virtualization simply means that computational work to be done is divided into individual concurrent entities or "threads," also called *virtual processors* (VPs), that are then mapped onto the physical processors by the runtime system (rather than by the programmer). As the work assigned to a VP is completed, the physical processors on which the VP performed its work is given a new VP, thus preventing the physical processors from becoming idle. Hence, by lifting the restrictions to which a standard approach is inherently subject, virtualization allows the mapping of computational work to processors to be assigned in a flexible manner that achieves an optimal interleaving of computation and communication. In this way, efficient scaling can be obtained even when very fine-grained parallelism is required.

The article is organized as follows. In Section 2 and 3, the new parallelization strategy for CPAIMD calculations using Charm++ is outlined, and the new software that has been developed described. The data structures used by the serial approach and the parallelization of these structures using Charm++ will also be presented. Section 4 discusses the various optimization schemes that have been employed in the course of scaling the new implementation to over 1000 processors. In Section 5, concluding remarks and directions for future development are given.

The CPAIMD Method

In this section, the basic physics underlying the Car–Parrinello *ab initio* Molecular Dynamics method (CPAIMD) is given. First, Density Functional Theory (DFT) with pseudopotentials, a simple but accurate electronic structure theory, is presented followed by a discussion of the implementation of DFT using a plane wave basis set. Given these basic formulae, the Car–Parrinello *ab initio* Molecular Dynamics (CPAIMD) approach is described that allows the nuclei to evolve on the ground-state Born–Oppenheimer surface provided by DFT. Finally, the computer science aspects of the serial CPAIMD algorithm are presented.

Density Functional Theory with Pseudopotentials

Consider a system with n_N nuclei having positions $\mathbf{R}_1, \ldots, \mathbf{R}_{n_N}$ and n_s electronic states or equivalently electronic orbitals, $\mathbf{\Psi}_1(\mathbf{r}), \ldots, \mathbf{\Psi}_{n_s}(\mathbf{r})$, with occupation numbers $f_1, \ldots, f_{n_s}$. If the electronic structure is represented within the Kohn–Sham formulation of density functional theory,[38–40] then the total energy (in atomic units) is given by

$$E[\{\mathbf{\Psi}\}, \{\mathbf{R}\}] = -\frac{1}{2}\sum_{i=1}^{n_s}\langle\mathbf{\Psi}_i|\nabla^2|\mathbf{\Psi}_i\rangle + \frac{1}{2}\int d\mathbf{r}d\mathbf{r}'\,\frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} + E_{\mathrm{xc}}[\rho] + E_{\mathrm{ext}}[\rho, \{\mathbf{R}\}] + V_{\mathrm{nucl}}(\{\mathbf{R}\}) \quad (1)$$

where the electron density, $\rho(\mathbf{r})$, is defined in terms of the states

$$\rho(\mathbf{r}) = \sum_i |\mathbf{\Psi}_i(\mathbf{r})|^2. \quad (2)$$

The states, in turn, are required to satisfy an orthogonality condition of the form

$$\langle\mathbf{\Psi}_i|\mathbf{\Psi}_j\rangle = f_i\delta_{ij}. \quad (3)$$

In eq. (1), the first term is the total kinetic energy of the noninteracting reference system, and the second term is the Hartree energy. The third term, the exchange-correlation functional, $E_{\mathrm{xc}}[\rho]$, since the precise form is not known for $E_{\mathrm{xc}}[\rho]$ reasonable approximations are used,[49–51] is followed by the external potential, $E_{\mathrm{ext}}[\rho, \{\mathbf{R}\}]$, which contains the interaction energy between the electrons and the nuclei. Minimization of the energy functional in eq. (1) with respect to the states subject to the orthonormality condition yields both the ground-state energy and electron density of the system.

It is computationally more efficient to treat only the valence electrons explicitly and to replace the core electrons by norm-conserving nonlocal pseudopotentials.[52] In this case, the external energy becomes state dependent and is expressed in the form

$$E_{\mathrm{ext}} = E_{\mathrm{ext,loc}}[\rho, \{\mathbf{R}\}] + E_{\mathrm{ext,non\text{-}loc}}[\{\mathbf{\Psi}\}, \{\mathbf{R}\}]$$

$$E_{\mathrm{ext,non\text{-}loc}}[\{\mathbf{\Psi}\}, \{\mathbf{R}\}] = \sum_i \langle\mathbf{\Psi}_i|V_{\mathrm{NL}}(\{\mathbf{R}\})|\mathbf{\Psi}_i\rangle \quad (4)$$

where E_{loc} is the local contribution to the external energy already described above. The operator V_{NL} is "nonlocal" because removal of the core electrons requires electrons in different angular momentum channels to interact differently with the nuclei.[52] [Note, eq. (4) is generally reformulated to allow for a faster evaluation as described in the next subsection.] When pseudopotentials are employed, the nuclei together with their core electrons are often referred to as "ions" to indicate the core electrons are included in an effective manner, and the charge on an ion is, hence, $Z_{\mathrm{ion}} = Z_{\mathrm{nucleus}} - en_{\mathrm{core}}$, where e is the charge on the electron and n_{core} is the number of core electrons. For example, a carbon nucleus has charge $Z_{\mathrm{nucleus}} = 6e$, while a carbon ion has charge $Z_{\mathrm{ion}} = 4e$ as the two (1s) core electrons of carbon have been replaced by pseudopotentials.

DFT Using a Plane-Wave Basis Set

In CPAIMD calculations, the states are, typically, expanded in a plane-wave basis set at the Γ-point according to

$$\mathbf{\Psi}_i(\mathbf{r}) = \sum_{\mathbf{g}} \mathbf{\Psi}_i(\mathbf{g})e^{i\mathbf{g}\cdot\mathbf{r}} \quad (5)$$

where $\{\mathbf{\Psi}_i(\mathbf{g})\}$ is a set of expansion coefficients, $\mathbf{g} = 2\pi\mathbf{h}^{-1}\mathbf{n}$ is a reciprocal space or reciprocal lattice vector, $\mathbf{n}$ is a vector of integers, and $\mathbf{h}$ is the matrix of the simulation cell, that is, the matrix whose columns are the cell vectors. In an orthorhombic box with side lengths, L_x, L_y, and L_z, $\mathbf{h} = \mathrm{diag}(L_x, L_y, L_z)$, and $\mathbf{g} = (g_x, g_y, g_z) = (2\pi n_x/L_x, 2\pi n_y/L_y, 2\pi n_z/L_z)$. The expansion in eq. (5) is truncated according to the criterion that $\frac{1}{2}|\mathbf{g}|^2 < E_{\mathrm{cut}}$, where E_{cut} is a suitably chosen energy cutoff. At the Γ-point, the coefficients satisfy the symmetry condition $\mathbf{\Psi}_i^*(\mathbf{g}) = \mathbf{\Psi}_i(-\mathbf{g})$ because the states are real, $\mathbf{\Psi}_i(\mathbf{r}) = \mathbf{\Psi}_i^*(\mathbf{r})$.

A similar plane wave expansion exists for the density,

$$\rho(\mathbf{r}) = \sum_{\mathbf{g}} \rho(\mathbf{g})e^{i\mathbf{g}\cdot\mathbf{r}}, \quad (6)$$

where $\rho^*(\mathbf{g}) = \rho(-\mathbf{g})$ (even away from the Γ-point). However, by virtue of eq. (2), the cutoff for the expansion of eq. (6) must be $4E_{\mathrm{cut}}$ rather than E_{cut}, which means that a larger set of reciprocal space vectors is needed (the set of g-vectors describing the states is a subset of g-vectors describing the density). It is useful to define, $\mathbf{G}/2$ the largest reciprocal lattice vector in each direction in the expansion of the density. Of course, $\mathbf{G}/4$ is then the largest reciprocal lattice vector in the expansion of states.

The expansion coefficients of the density, $\rho(\mathbf{g})$ can be obtained from the expansion coefficients of the states, $\mathbf{\Psi}_i(\mathbf{g})$, exactly by the following procedure: perform a complex-to-real 3D FFT of size $\mathbf{G}$ (e.g., $\{N \times N \times N\}$ in cubic box) on each $\mathbf{\Psi}_i(\mathbf{g})$ to produce $\mathbf{\Psi}_i(\mathbf{r})$ on the discrete mesh, square the function, $|\mathbf{\Psi}_i(\mathbf{r})|^2$, and sum the results over all states to produce $\rho(\mathbf{r})$ on the discrete mesh (e.g., $\{N \times N \times N\}$ in a cubic box). The result can then be inverse transformed by a real-to-complex 3D FFT to generate the desired result.

Next, each term in the energy, eq. (1), will be expressed in terms of the plane-wave basis. The Hartree energy is given by

$$E_{\text{Hartree}} = \frac{1}{2V} \sum_{\mathbf{g} \neq (0,0,0)} \frac{4\pi}{|\mathbf{g}|^2} |\rho(\mathbf{g})|^2 \tag{7}$$

where V is the volume of the system assuming three-dimensional periodic boundary conditions.[53,54]

The local part of the external energy is

$$E_{\text{ext,loc}} = \frac{1}{V} \sum_{\mathbf{g}} \rho^*(\mathbf{g}) \sum_I \tilde{\phi}_{\text{loc,I}}(\mathbf{g}) S_I(\mathbf{g})$$

$$S_I(\mathbf{g}) = \exp(-i\mathbf{g} \cdot \mathbf{R}_I) \tag{8}$$

where $\tilde{\phi}_{\text{loc,I}}(\mathbf{g})$ is the Fourier transform of the local interaction, $\phi_{\text{loc,I}}(r)$, between an electron and the Ith ion and $S_I(\mathbf{g})$ is the ionic structure factor (again assuming three dimensional periodic boundary conditions).

The exchange-correlation energy $E_{\text{xc}}[\rho]$ is not known exactly, and must, therefore, be approximated. In the local density approximation, it is assumed that the density does not vary rapidly in space so that the functional takes the form

$$E_{\text{xc}}[\rho] = \int d\mathbf{r} \rho(\mathbf{r}) \varepsilon_{\text{xc}}(\rho(\mathbf{r})) \tag{9}$$

while in the generalized gradient approximation, which extends eq. (9), the functional is taken to be of the form

$$E_{\text{xc}}[\rho] = \int d\mathbf{r} \rho(\mathbf{r}) \varepsilon_{\text{xc}}(\rho(\mathbf{r}), \nabla\rho(\mathbf{r})). \tag{10}$$

In practice, these integrals are evaluated on a set of equally spaced grid points defined by the size of the 3D FFT,[55] using trapezoidal rule

$$E_{\text{xc}}[\rho] = \Delta^3 \sum_{ijk} \rho(\mathbf{r}_{ijk}) \cdot \varepsilon_{xc}(\rho(\mathbf{r}_{ijk})) \tag{11}$$

where Δ is the grid spacing and $\rho(\mathbf{r}_{ijk})$ is the electron density at a grid point. Hereafter, the indices, ijk, will be suppressed/understood. If generalized gradient functionals are employed, the exchange correlation function depends on both the density, $\rho(\mathbf{r})$ and the gradient of the density, $\nabla\rho(\mathbf{r})$, on the discrete grid. Because

$$\nabla\rho(\mathbf{r}) = \sum_{\mathbf{g}} i\mathbf{g}\rho(\mathbf{g}) e^{i\mathbf{g}\cdot\mathbf{r}}, \tag{12}$$

the desired gradient can be computed by performing an inverse Fourier transform for each of the components, $i\mathbf{g}\rho(\mathbf{g})$, and $\varepsilon_{xc}(\rho(\mathbf{r}), \nabla\rho(\mathbf{r}))$ determined.[55]

In a plane wave basis set, the nonlocal energy in the Kleinman–Bylander form[56] is given by

$$E_{\text{NL}} = \sum_{i=1}^{n_s} \sum_{I=1}^{n_N} \sum_{l=0}^{\bar{l}-1} \sum_{m=-l}^{l} \mathcal{N}_{Ilm} |Z_{i,I,l,m}|^2 \tag{13}$$

where $\bar{l}$ is the highest angular momentum channel treated as a nonlocal component, $\mathcal{N}_{Ilm}$ is a normalization factor, and

$$Z_{i,I,l,m} = \sum_{\mathbf{g}} \mathbf{\Psi}_i(\mathbf{g}) e^{i\mathbf{g}\cdot\mathbf{R}_I} h_{Il}(|\mathbf{g}|) \mathbf{Y}_{lm}(\boldsymbol{\theta}_{\mathbf{g}}, \varphi_{\mathbf{g}}). \tag{14}$$

In eq. (14), $h_{Il}(|\mathbf{g}|)$ is the lth spherical Bessel function transform of the angular-momentum dependent potential function, $h_{Il}(\mathbf{r})$, describing the interaction the electron in angular momentum channel, l, interacting with ion, I, and $Y_{lm}(\boldsymbol{\theta}_{\mathbf{g}}, \varphi_{\mathbf{g}})$ is a spherical harmonic.

The kinetic energy of non-interacting electrons, also, depends on the individual states. It can be expressed as

$$E_{kin} = \frac{1}{2} \sum_i |\mathbf{\Psi}_i(\mathbf{g})|^2 |\mathbf{g}|^2 \tag{15}$$

in the plane wave basis. Finally, in the plane wave basis, the orthogonality condition is

$$\langle \mathbf{\Psi}_i | \mathbf{\Psi}_j \rangle = \sum_{\mathbf{g}} \mathbf{\Psi}_i(\mathbf{g}) \mathbf{\Psi}_j^*(\mathbf{g}) = f_i \delta_{ij}. \tag{16}$$

To obtain physical results, the energy functional must be minimized at fixed nuclear position. This is generally accomplished using a conjugate gradient procedure. Because a plane wave basis is employed, the negative derivative of each term given above with respect to the coefficients of the plane wave expansion must be taken, which is denoted as $F_{\mathbf{\Psi}_i}(\mathbf{g})$, the "force" on the coefficients. The derivative of the kinetic energy of noninteracting electrons and the nonlocal potential is simple because $\mathbf{\Psi}_i(\mathbf{g})$ appears explicitly. The procedure employed to compute the forces due to the local potential, the Hartree, and the exchange correlation energy is slightly more complex because these terms involve the density rather than the states, directly. The forces are evaluated by first, computing the Kohn–Sham (KS) potential on the real space grid,

$v_{KS}(\mathbf{r})$. The Hartree and and local contribution to the KS potential are calculated in g-space,

$$v_{KS}^{(HL)}(\mathbf{g}) = \frac{1}{V}\sum_I \tilde{\phi}_{\mathrm{loc},I}(\mathbf{g})S_I(\mathbf{g}) + \frac{1}{V}\rho(\mathbf{g})\frac{4\pi}{|\mathbf{g}|^2}, \tag{17}$$

and the result $v_{KS}^{(HL)}(\mathbf{g})$ transformed to real space via a 3D FFT and added to the contribution from exchange correlation, which is determined directly in real space,

$$v_{KS}(\mathbf{r}) = v_{KS}^{(HL)}(\mathbf{r}) + \frac{\partial}{\partial\rho(\mathbf{r})}(\rho(\mathbf{r})\varepsilon_{xc}(\rho(\mathbf{r}))), \tag{18}$$

at each grid point. The product of the KS potential and with the real space representation of each state, $v_{KS}(\mathbf{r})\boldsymbol{\Psi}_i(\mathbf{r})$, is constructed by a point by point multiplication. A 3D FFT of the resulting products yields, exactly, the contribution to the force on the coefficients of each state from Hartree, local pseudopotential energy, and exchange-correlation. If gradient corrections are present, the procedure is more complex, involving several 3D FFTs[55] to generate correctly the contribution to $v_{KS}(\mathbf{r})$ from exchange correlation.

Finally, forces on the ions, $\mathbf{F}_I$, which have physical meaning only when the functional is minimized, $F_{\boldsymbol{\Psi}_i}(\mathbf{g}) \equiv 0$, arise from three terms, the local energy external energy, the nonlocal energy, and the ion–ion interaction, $V_{\mathrm{nucl}}(\mathbf{R})$. The ion–ion interaction is general taken to be simply Coloumbs law,

$$V_{\mathrm{nucl}}(\mathbf{R}) = \sum_{\mathbf{S}}\sum_{IJ}{}' \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J + \mathbf{Sh}|}, \tag{19}$$

where $\mathbf{S}$ are the periodic replicas and the prime indicates $I \neq J$ when $\mathbf{S} = 0$. This term is typically evaluated using Ewald summation.[57,58] The computation of the ion forces takes negligible computer time, and will not be discussed further.

Car–Parrinello Molecular Dynamics

In the Car–Parrinello approach, a fictitious dynamics is invented for the plane-wave coefficients $\{\boldsymbol{\Psi}_i(\mathbf{g})\}$ of the states that allows an initially minimized electronic configuration (e.g., via conjugate gradient procedure at fixed nuclear position) to follow the nuclear motion so as to produce nuclear dynamics on the ground state Born–Oppenheimer surface. This dynamics is generated by positing a set of Newton-like equations of motion for the expansion coefficients and creating an adiabatic separation between this fictitious state dynamics and the real dynamics of the nuclei. In the CP scheme, therefore, the electrons and nuclei can be treated on an equivalent mathematical and, hence, algorithmic footing. If the electrons are kept at a temperature T_e that is much less than the nuclear temperature T, and the electrons are allowed to move on a much faster time scale than the nuclei, then the electrons will remain approximately minimized as the nuclei are propagated along a numerical trajectory.[59] The equations of motion take the form

$$\mu(\mathbf{g})\ddot{\boldsymbol{\Psi}}_i(\mathbf{g}) = -\frac{\partial E}{\partial\boldsymbol{\Psi}_i^*(\mathbf{g})} + \sum_j \Lambda_{ij}\boldsymbol{\Psi}_j(\mathbf{g}) = F_{\boldsymbol{\Psi}_i}(\mathbf{g}) + \sum_j \Lambda_{ij}\boldsymbol{\Psi}_j(\mathbf{g})$$

$$M_I\ddot{\mathbf{R}}_I = -\frac{\partial E}{\partial\mathbf{R}_I} = \mathbf{F}_I \tag{20}$$

where $\mu(\mathbf{g})$ is a mass-like parameter (having units of energy $\times$ time^2) for the electronic motion, and Λ_{ij} is a set of Lagrange multipliers for enforcing the orthogonality condition in eq. (3) (see later). In practice, the equations of motion can be integrated using a velocity Verlet scheme combined with Shake and Rattle procedures to enforce the orthonormality conditions expressed in terms of holonomic constraints as described in refs. 60–62.

Serial Description of the Algorithm: Flowchart

The algorithm is summarized in its essential parts in the flowchart of Figure 1. The calculation takes as input the Fourier coefficients of a set of states, the $\boldsymbol{\Psi}_i(\mathbf{g})$ of eq. (5), which satisfy the orthogonality condition given in eq. (16). The calculation splits immediately into two *independent* branches. In the right branch of the figure, the states are transformed into real space by 3D FFT, squared, and summed to form the density in real space following eq. (2). The calculation then bifurcates into two more independent branches. The density in real space is employed to compute the exchange correlation energy and its contribution to KS potential as described in eq. (11) (more 3D FFTs are required if gradient corrected exchange correlation functions are employed). In the other branch, the density in real space is transformed to g-space by 3D FFT and the Hartree and local pseudopotential energy computed using eq. (17). The KS contribution of these terms is computed in g-space, transformed into real space by 3D FFT, and added into the contribution from the exchange correlation function to realize eq. (17). Each state is then multiplied by the KS potential in real space and a 3D FFT is performed to complete the computation of the contribution to the force, $F_{\boldsymbol{\Psi}_i}(\mathbf{g})$, from the Hartree, Exchange-Correlation, and External Pseudopotential energies as described in text below eq. (17). In the left branch of the figure, the kinetic energy of the noninteracting electrons, eq. (15), and the nonlocal pseudopotential energy, eq. (14), are computed along with their contribution to the $F_{\boldsymbol{\Psi}_i}(\mathbf{g})$. At this point the two main branches rejoin, the two force contributions are summed and the states evolved using the total $F_{\boldsymbol{\Psi}_i}(\mathbf{g})$ as part of the numerical integration of the CPMD equations of motion, eqs. (20), or simply in a standard constrained conjugate gradient step at fixed nuclear positions. Once the states are evolved, they will violate orthogonality, slightly due to finite step size errors in the solver. CPMD requires a Shake/Rattle step[60] to "reorthogonalize" the states while a conjugate gradient minimization step will employ either the Gram–Schmidt or the Löwdin orthogonalization procedure. The implementation of the Löwdin method is described in detail later. After the states are "reorthogonalized," the CPAIMD step is complete.

Serial Description of the Algorithm: Data Dependencies

Next, the discussion of the algorithm will be recapitulated but with more attention paid to the data dependencies, a clear understanding

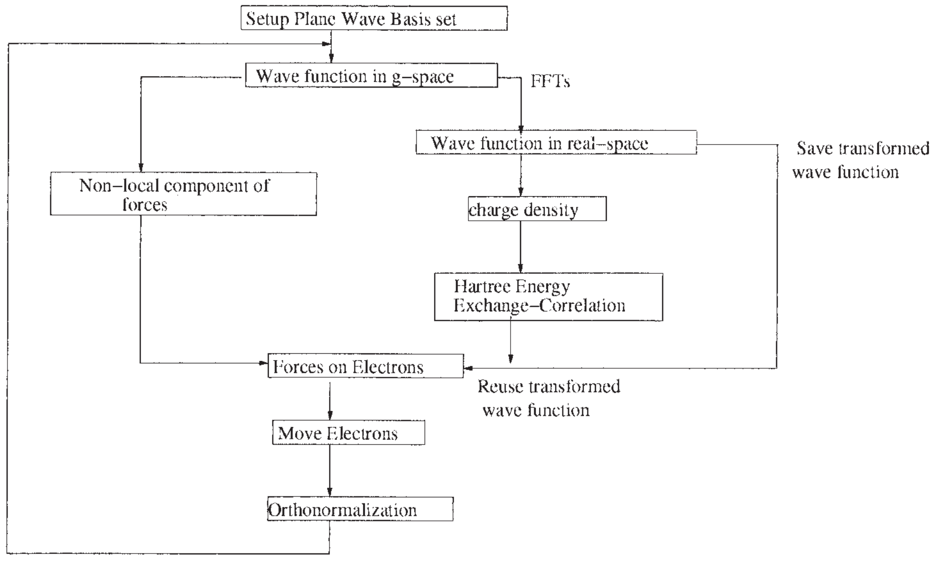

Setup Plane Wave Basis set
Wave function in g–space
FFTs
Wave function in real–space
Save transformed wave function
Non–local component of forces
charge density
Hartree Energy Exchange–Correlation
Forces on Electrons
Reuse transformed wave function
Move Electrons
Orthonormalization

Figure 1. Schematic flowchart of the implementation of the CP algorithm.

of which is required before a scalable parallel algorithm can be developed. The basic objects are the n_s, occupied, real valued electronic states, $\{\boldsymbol{\Psi}_i(\mathbf{r})\}$, and their corresponding complex valued plane-wave expansion coefficients, $\{\boldsymbol{\Psi}_i(\mathbf{g})\}$, which possess the symmetry $\{\boldsymbol{\Psi}_i^*(\mathbf{g})\} = \{\boldsymbol{\Psi}_i(-\mathbf{g})\}$. Each state in real space, $\boldsymbol{\Psi}_i(\mathbf{r})$, is represented by a 3D array ($N \times N \times N$) of real numbers assuming a cubic simulation cell for simplicity ($L_x = L_y = L_z$). Although the g-space representation should, in principle, be a dense $N \times N \times N$ array, a spherical cutoff is employed, $\frac{1}{2}|\mathbf{g}|^2 < E_{\text{cut}}$, which confines the nonzero elements to lie within a sphere of radius proportional to $N/4$. In general, the electron density in real space, $\rho(\mathbf{r})$, is similarly represented by a single 3D array of dimension $N \times N \times N$, with Fourier coefficients, $\rho(\mathbf{g})$. Like $\boldsymbol{\Psi}_i(\mathbf{g})$, the nonzero elements of $\rho(\mathbf{g})$ are also contained within a sphere but this time with radius proportional to $N/2$ (i.e., because the density is equal to $\sum_i |\boldsymbol{\Psi}_i(\mathbf{r})|^2$). In general, the real space representations can be said to be "dense" while the reciprocal space representations are comparatively sparse. Note, while employing sparse matrices reduces computation and communication, it also poses tricky load balance issues in parallel (i.e., some portions of reciprocal space have more nonzero data values than others).

Having described the data dependencies, the states and the electron density are shown, pictorially in Figure 2, evolving along the steps of the algorithm described in the previous section, which is now divided into IX Phases. The algorithm begins with the reciprocal-space expansion coefficients of the states, $\{\boldsymbol{\Psi}_i(\mathbf{g})\}$, which satisfy the orthogonality condition of eq. (16). In Phase I,

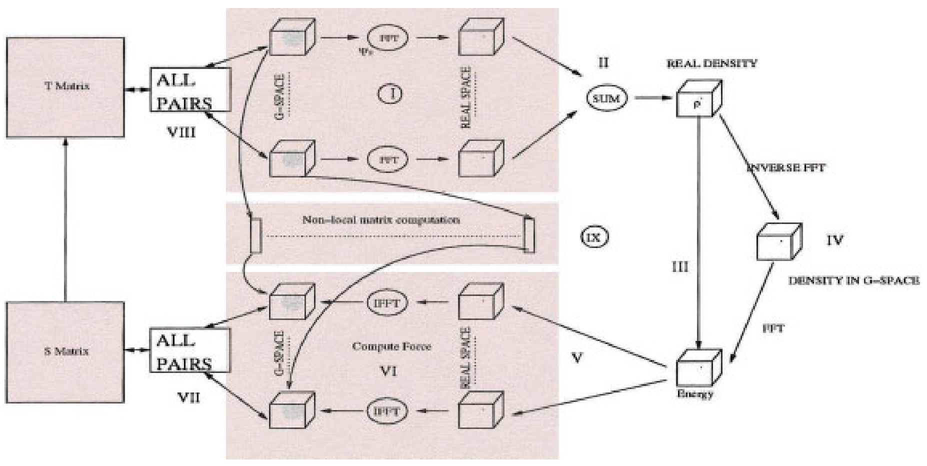


Figure 2. A serial view of the CPAIMD algorithm, showing data dependences.

each state is transformed into its Cartesian or "real"-space" representation via 3D FFTs (Phase I), to yield the functions $\{\boldsymbol{\Psi}_i(\mathbf{r})\}$ on the real-space grid. The real-space representations of all states are then squared and summed to obtain the electron density, $\rho(\mathbf{r})$ in the "Reduction" operations of Phase II. With $\rho(\mathbf{r})$ in hand, the Fourier coefficients of the electron density, $\rho(\mathbf{g})$, can be obtained via an inverse 3D FFT.

Two independent computations are now performed using the electron density and its Fourier coefficients, respectively. First, the exchange-correlation energy, and its contribution to the KS potential are determined from the density in real space (Phase III). Second, the Hartree and local external energies are computed using the Fourier coefficients of the density (Phase IV) along with the contribution of these two terms to KS potential, which is expressed on the same size reciprocal space as that describing the density, $\rho(\mathbf{g})$. The Hartree and local external contributions to the KS potential are then transformed back into real space via 3D FFTs (Phase V) and added to the real space KS contribution determined in Phase III.

The two independent computational threads rejoin in Phase VI. In Phase VI, each state in real space (dense array of $N \times N \times N$) is multiplied by the KS potential (dense array of $N \times N \times N$) and Fourier Transformed back into reciprocal space to generate the force due to Hartree, local external, and exchange correlation energies in reciprocal space. In reciprocal or g-space, the forces on the coefficients of the states, $F_{\boldsymbol{\Psi}_i}(\mathbf{g})$, are sparse arrays with nonzero values that lie within a sphere of radius proportional to $N/4$, (i.e., the forces in g-space possess the same data structure as the states in g-space).

Independent of Phases I–VI, the kinetic energy of noninteracting electrons, the nonlocal energy, and the forces arising from these two terms are computed in reciprocal space during Phase IX using the sparse reciprocal space representation of the states, $\{\boldsymbol{\Psi}_i(\mathbf{g})\}$. The forces due to the kinetic energy of noninteracting electrons and the nonlocal energy are added to the forces from Phase VI in Phase VII where the states in reciprocal space form, $\{\boldsymbol{\Psi}_i(\mathbf{g})\}$, are evolved to the next step. Orthogonality is enforced using the reciprocal space representation in Phase VIII via nonsquare matric multiplications of the form $A(n_s \times n_s)$ times $B(n_s \times N^3)$ after which Phase I can, again, be instantiated.

Parallelization of the CPAIMD Method

Roadmap

In this section, the fine grained parallelization of the CPAIMD method across thousands of processors is described in detail. First, the basic strategy based on virtualization is discussed. Although powerful, the utility of virtualization has not been widely appreciated in the physical sciences and, indeed, without sophisticated software tools, the application of this approach would be difficult. Hence, the widely used Charm++ parallel middleware[47,48] is employed to implement the new parallel design. Second, the data decomposition used in the new fined grained algorithm is motivated. Third and fourth, the specific techniques required to achieve parallel scaling of two crucial steps of the CPAIMD method, orthonormalization, and the multiple, concurrent 3D Fast Fourier Transforms are given. Although the orthonormalization dominates the computation in large systems, in moderate size systems, the prefactors are such that the FFT work is dominant on a single processor. However, efficient parallelization of the 3D FFTs leads to a shift of the bottleneck to the orthonormalization step, requiring a finely honed parallel algorithm for the latter to be developed. Last, a discussion of the mapping of virtual processors to true processors is provided followed by a description of communication patterns employed to optimize the parallel implementation and achieve scaling of small systems on thousands of processors.

Overall Strategy: Virtualization Using the Charm++ Middleware

The design of parallel algorithms can be viewed as a two-stage process. In the first stage, the computational work and data are decomposed into portions that require the minimal amount of communication between them to function and, in the second stage, these discretizations are assigned to processors. Note, there are two operative words, "discretization" and "processor." Physical scientists are generally trained to parallelize their methods without recourse to the separate nature of these two concepts.[46] That is, the number of processors is given absolute control over the number and type of discretizations, inhibiting the development of more flexible parallelization schemes.

To "liberate" the concepts of processor and discretization, computer scientists have created a third concept called *virtual processors* (VPs) or *virtualization.*[47] In this framework, the work is divided into a number of objects or virtual processors that are independent of the number of true processors available, and typically should substantially exceed the number of true processors for effective computation/communication interleaving. Communication patterns among the virtual processors are developed without regard to the precise mapping of these entities to true processors. Multiple virtual processors can be mapped dynamically and simultaneously to the physical processors as the calculation proceeds. When a physical processor exhausts the work corresponding to a VP or if an active VP is blocked, another VP residing on the same processor can immediately be launched. Although conceptually simple and intuitively appealing, the problem of developing software tools and data structures that can instantiate the VPs and then map them in an optimal way to physical processors is a nontrivial one. The challenge is to realize a mapping that minimizes communication and maximizes performance by interleaving communication and computation in the most effective manner. However, assuming the existence of the sophisticated "parallel middleware" required by the VP paradigm, it is clear that the routine use of this model would result in more effective parallelization. Indeed, as the number of physical processors increases to thousands, it is difficult to envision parallelization schemes that would perform well without employing virtualization.

The parallel middleware of choice to enable the VP model is Charm++.[47,48] In Charm++, the VPs are objects (called *chares,* which is the Old English word for chore), that communicate with each other via asynchronous method invocations or "messages." The Charm++ run-time system controls the placement of the chares on the physical processors, is capable of remapping the chares, on the fly, to improve load balance, and employs sophisticated communication optimization strategies or recognized patterns to speed up the message throughput and minimize latencies on the communication network connecting the processors. The applications programmer is left with the task of mapping his problem onto VPs and writing code using Charm++ extensions to realize the VPs as individual chares or sometimes, more conveniently, as arrays of chares.[63,64]

To apply the VP programming model to CPAIMD, the authors, among whom are both physical and computer scientists, worked in close collaboration. In the process of designing the novel parallel algorithm described below, both the CPAIMD software and the parallel middleware were improved individually. Details of the new parallel CPAIMD scheme are described in the following subsections. In the next major section, it is demonstrated that the new parallel algorithm, which is a synergistic union of the parallel middleware and the application software, is able to achieve good scaling on thousands of processors with minimal effort on the part of the applications programmer.

Decomposition of Data: The First Step in Parallel Algorithm Design

In the implementation of this article, the orbitals in g-space and real-space, which are sparse and dense cubes of data, respectively, are each decomposed into planes. The work related to each plane is performed by a Charm++ virtual processor (or chare). In accordance with the philosophy of processor virtualization, the number of virtual processors depends only on issues such as the work to communication ratio but is independent of the total number of physical processors. A collection G holds *objects* $G(i, p)$ corresponding to plane, p, of state, i, in g-space. The plane index, p, is identical to the x coordinate in g-space, g_x, and the object $G(i, g_x)$ houses the coefficients $\mathbf{\Psi}_i(g_x, g_y, g_z)$ for all values of g_y and g_z. Similarly, another collection R holds real-space planes $R(i, p)$ corresponding to plane p of state i. However, the axis of decomposition is different for G and R, due to the way the FFTs are implemented (to be described in due course). In addition, there are one-dimensional chare arrays corresponding to the electron density in real-space, $\rho(\mathbf{r})$ and in g-space, $\rho(\mathbf{g})$.

The real-space planes are dense, and each state has precisely the same number of planes as the electron density, that is, N. The g-space planes are sparse, and only g-space planes with nonzero elements are included in the calculation. There are roughly twice as many nonzero g-space plane in the reciprocal space representation of the electron density as in the corresponding reciprocal space representation of a state.

It is worth noting that in a benchmark computation on a system of 32 water molecules, which has $n_s = 128$ states and $N = 100$, this scheme decomposes the computation into 12,800 virtual processors corresponding to states in real-space, and about half as many virtual processors corresponding to the planes in g-space. The reason for this reduction is due to the use of a spherical cutoff in g-space. In addition, there are 100 virtual processors each corresponding to density in real-space and reciprocal space.

The resultant decomposition of the problem is shown in Figure 3. Next, the parallel implementation of the each phase of the calculation, using the aforementioned decomposition scheme, is described, starting with the orthonormalization phase. This is followed by a discussion of the mapping of virtually processors to physical processor. Finally, optimizations of the communication patterns that arise from the mapping are given.

Maintaining the Orthogonality of the States: Phases VII and VIII

After the evolution of plane wave coefficients by their forces either during a conjugate gradient minimization procedure or during a full Car–Parrinello *ab initio* molecule dynamics calculation (see earlier), rigorous orthogonality of the states that must be maintained is violated, slightly, due to finite step size errors in the solver(s). Thus, an orthonormalization step is performed in Phases VII and VIII of the algorithm as given in Figures 2 and 3. The basic operations required to enforce orthonormality are the computation of one or more overlap matrices, processing of the overlap matrices to produce a "transformation matrix," followed by the application of the matrix on the nonorthogonal states to generate a rigorously orthonormal set.

The parallelization of the most conceptually simple orthogonalization algorithm, the Löwdin method, will be discussed for clarity. (Note that the Löwdin method can only be used in conjunction with conjugate gradient minimization of the electronic energy.) In the Löwdin method, the matrix, $S_{ij} = \langle \mathbf{\Psi}_i | \mathbf{\Psi}_j \rangle$ is computed, the transformation matrix, $T_{ij} = (S^{-1/2})_{ij}$, is formed, and the transformation $|\mathbf{\Psi}_i^{\text{new}}\rangle = \sum_j T_{ij} |\mathbf{\Psi}_j\rangle$ to the orthonormal set performed. Only the first and third steps of the method are carried out in parallel. The computational cost required to form T for a system of 128 doubly occupied states is very small. In general, the cost of this step is order n_s^3, and could be parallelized. Note, Gram–Schmidt orthogonalization and the Shake/Rattle algorithms [which are appropriate for use with eqs. (20)][60] can be implemented using the basic tools required for the Löwdin and, therefore, will not be described here.

In more detail, the overlap matrix S_{ij} is real and symmetric because the states are taken to be real valued, $\mathbf{\Psi}_i(x, y, z) = \mathbf{\Psi}_i^*(x, y, z)$, which leads to $\mathbf{\Psi}_i(g_x, g_y, g_z) = \mathbf{\Psi}_i^*(-g_x, -g_y, -g_z)$. Again, i is the state label and (g_x, g_y, g_z) is the g-vector label. The overlap is computed in g-space as

$$S_{ij} = \sum_{g_x} \sum_{g_y} \sum_{g_z} \mathbf{\Psi}_i(g_x, g_y, g_z) \mathbf{\Psi}_j^*(g_x, g_y, g_z)$$
$$= 2Re\left[\sum_{\mathbf{g}>0} \mathbf{\Psi}_i(\mathbf{g}) \mathbf{\Psi}_j^*(\mathbf{g})\right] + \mathbf{\Psi}_i(0)\mathbf{\Psi}_j^*(0), \quad (21)$$

where the latter formula takes advantage of the fact that the states are real valued. The restriction, $\mathbf{g} > 0$, indicates that only the upper half sphere of nonzero elements of the $\{\mathbf{\Psi}(\mathbf{g})\}$ are employed, and *Re* indicates the real part is to be taken. Once S is calculated, its inverse square root, T, can be computed, and the orthonormal orbitals formed in reciprocal space by the transformation

$$\mathbf{\Psi}_i^{(\text{new})}(g_x, g_y, g_z) = \sum_j T_{ij} \mathbf{\Psi}_j(g_x, g_y, g_z) \quad (22)$$

as discussed above.

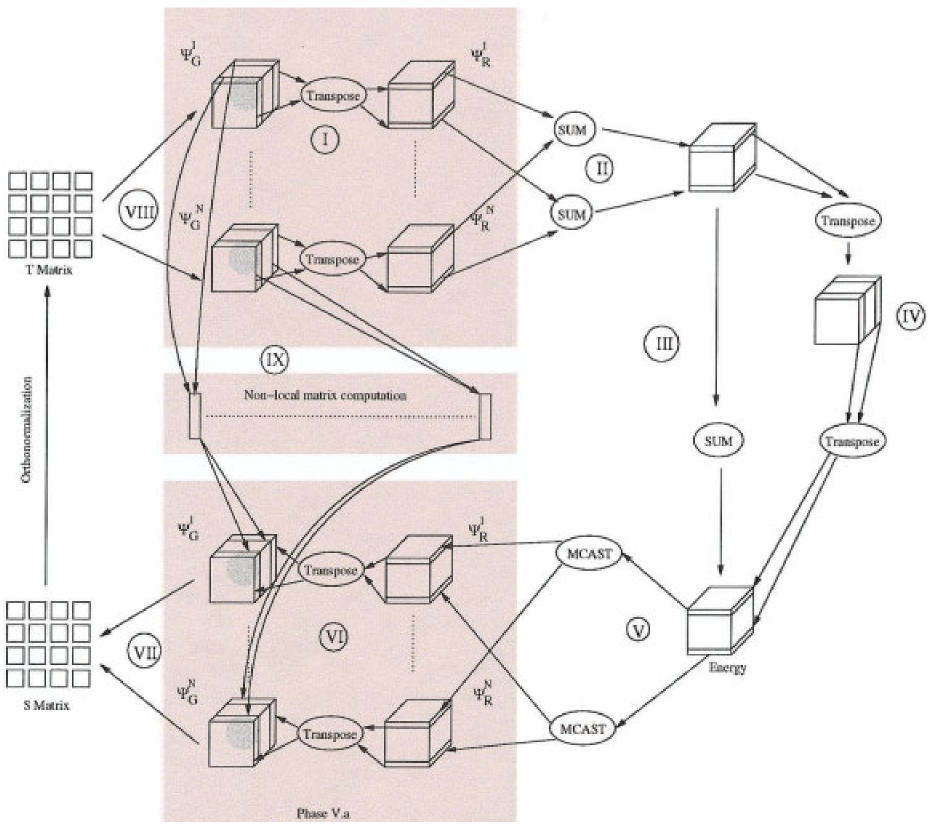


Figure 3. Parallel structure of our implementation. Roman numbers indicate phases.

Now, consider the parallel implementation of eq. (21). The overlap matrix, S, is computed in two steps, taking advantage of the fact that each state in g-space has already been decomposed into multiple g-space planes (see Fig. 3). The plane chare array, $G(i, g_x)$, holds all Fourier coefficients $\{g_y, g_z\}$ of state, i, at fixed g_x. Thus, it is natural to compute an intermediate overlap matrix,

$$S^{(\text{int})}_{ij}(g_x) = \sum_{g_y}\sum_{g_z} \mathbf{\Psi}_i(g_x, g_y, g_z)\mathbf{\Psi}^*_j(g_x, g_y, g_z) \tag{23}$$

which can then be employed to compute the desired result,

$$S_{ij} = \sum_{g_x} S^{(\text{int})}_{ij}(g_x) \tag{24}$$

by a reduction operation. Clearly, to evaluate eq. (23), all pairs of coefficients labeled, i, j in the same plane indexed by g_x, contained in the chare arrays, $G(i, g_x)$ and $G(j, g_x)$, need to "meet" on a processor.

One natural way to ensure that all pairs of planes meet on a processor is to employ a ring-based approach which requires n_s steps. In the first step, each $G(i, g_x)$ sends all its g-space data to $G((i + 1)\%n_s, g_x)$, upon receipt of which, $G((i + 1)\%n_s, g_x)$ computes $S^{(\text{int})}_{i,(i+1)\%n_s}(g_x)$, where the "%" sign indicates the periodicity of the ring, that is, state n_s must send to state 1. In each subsequent step, $G(i, g_x)$ forwards the data that it received in the preceding step. This can be visualized as N rings of size n_s formed from the N planes. After n_s steps, all entries of $S^{(\text{int})}(g_x)$ will have been computed, albeit by different chare arrays, $G(i, g_x)$. The different parts of $S^{(\text{int})}(g_x)$ are then collected and used to compute the matrix S, as per eq. (24). The matrix, T, is then computed, and relevant portions are communicated to all chare arrays, $G(i, p)$. The method is then driven in reverse to calculate the new states via eq. (22). That is, at each hop, another portion of $\mathbf{\Psi}^{(\text{new})}_i(g_x)$ can be generated, $\mathbf{\Psi}^{(\text{new})}_i(g_x) \leftarrow \sum_j T_{ij}\mathbf{\Psi}_j(g_x)$. Although a ring is a better strategy than naively implementing an all-to-all communication operation between the $G(i, p)$, the data contained in each $G(i, p)$ is still sent through the network n_s times. Because the amount of g-space data is more than 100 MB in, for example a system of 32 water molecules (the benchmark system we consider in the Performance and Optimization section), the ring approach requires tremendous communication bandwidth. Also, the size of the rings is equal to the number of state, n_s, which implies a long delay before the computation of the S matrix is completed.

To reduce the communication overhead required to compute the overlap matrix, an alternative approach has been devised that shall be referred to as the *block-pairs* algorithm. In this method, the matrix $S^{(\text{int})}$ is subdivided into $\gamma \times \gamma$ regions or blocks. A chare array of virtual processors, $S^{(\text{arr})}$ of size equal to the number of blocks, is created. Each element is assigned a block and given the task of computing all overlaps within its block. Thus, the data of $G(i, g_x)$ and $G(j, g_x)$ "meet" on the virtual processor whose block

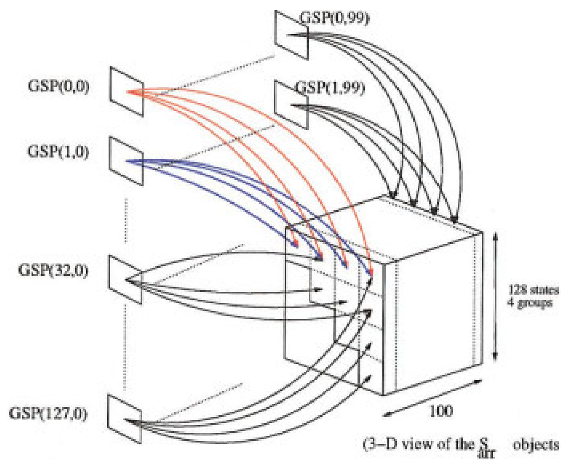


Figure 4. Computing the entries of the S matrix: each array element $S_{aux}(s, s')$, $s \leq s'$ computes at least γ^2 entries of the S matrix.

contains the i, j element of $S^{(\text{int})}$ as shown in Figure 4. The block size, γ, is adjusted to obtain the highest parallel efficiency for a given problem size and physical processor number.

The block-pairs algorithm can be described as follows: The chare array element $S^{(\text{arr})}_{i/\gamma,j/\gamma}(g_x)$ receives g-space data from $G(i, g_x)\cdots G(i + \gamma - 1, g_x)$ and $G(j, g_x)\cdots G(j + \gamma - 1, g_x)$, that is, 2γ messages. Using this data, a $\gamma \times \gamma$ part of the $S^{(\text{int})}$ matrix is computed as per eq. (23). Note, S is a real symmetric matrix and $S^{(\text{arr})}_{i/\gamma,j/\gamma}(g_x)$ can do the work of $S^{(\text{arr})}_{j/\gamma,i/\gamma}(g_x)$. Thus, $S^{(\text{arr})}_{ij}(g_x)$ are created only for $i \geq j$. This is illustrated in Figure 4. Once the $S^{(\text{arr})}$ have completed their assigned tasks, the overlap matrix, S, can be calculated by performing a reduction operation over the $S^{(\text{arr})}$ virtual processor array.

Because the original states are already present in the $S^{(\text{arr})}$ chare array, the new g-space states can be computed by the same set of chares. To compute the new states via eq. (22), each $S^{(\text{arr})}_{i/\gamma,j/\gamma}(g_x)$ needs a $\gamma \times \gamma$ portion of the T matrix. The T matrix was made available by sending the entire matrix to all processors via a machine level broadcast, as this approach was more efficient than making many multicasts of small $\gamma \times \gamma$ portions of the matrix. Once the new g-space planes of the states, $\mathbf{\Psi}_i(g_x)\cdots\mathbf{\Psi}_{i+\gamma-1}(g_x)$, are computed in the $S^{(\text{arr})}$ object, they are sent back to the appropriate original chare arrays, $G(i, g_x)$.

To compute the communication cost of the block-pairs algorithm, consider the operations defined by the $G(i, g_x)$ chare arrays. Each $G(i, g_x)$ has to multicast its data to the n_s/γ elements of $S^{(\text{arr})}$. Hence, the total g-space data is communicated n_s/γ times, instead of n_s times, as in the case of the ring method. The larger γ, the smaller the communication cost is for the $G(i, g_x)$ chare array. However, if γ is chosen to be too large, the number of messages increases, creating latency problems, although the total bandwidth decreases. In addition, the work load of each S_{aux} chare array increases, and the degree of parallelism afforded by the algorithm suffers.

Parallelization of Three-Dimensional FFTs: General Considerations

The parallelization of the 3D FFT has been considered in detail by a variety of researchers (cf. refs. 65 and 66). The most commonly employed algorithm, which is adopted here, is based on transposes.[65] Briefly, a 3D FFT consists of a series of three 1D FFT, that is, one FFT to transform each index, $\mathbf{\Psi}_i(g_x, g_y, g_z) \rightarrow \mathbf{\Psi}_i(x, g_y, g_z)$ for all g_x, g_z, $\mathbf{\Psi}_i(x, g_y, g_z) \rightarrow \mathbf{\Psi}_i(x, y, g_z)$ for all x, g_z and, finally, $\mathbf{\Psi}_i(x, y, g_z) \rightarrow \mathbf{\Psi}_i(x, y, z)$ for all x, y. In transpose parallelization, each 1D FFT is performed on a single (virtual) processor that requires a transpose operation to collect the required data. Although it is possible to distribute all three indices across VPs, here, only planes of data are distributed. This requires only one transpose operation as opposed to two and was found to be sufficient. That is, $\mathbf{\Psi}_i(g_x, g_y, g_z) \rightarrow \mathbf{\Psi}_i(x, g_y, g_z)$ is parallelized using the g_z index so that complete $\{g_x, g_y\}$ planes are distributed across the virtual processors (e.g., all points $\{g_x, g_y\}$ and, hence, $\{x, g_y\}$ are together). Upon completion of the first set of 1D FFTs, a transpose is performed to parallelize the x index, collecting complete $\{g_y, g_z\}$ planes. At this point, the remaining two sets of 1D FFTs can be performed on the same processor without communication. The 1D FFTs were computed using FFTW.[67]

It is important to note that in performing the FFTs, the sparse nature of the states in reciprocal space is used to reduce both the computation and the communication. The nonzero values of $\mathbf{\Psi}_i(g_x, g_y, g_z)$ form a sphere of radius, $N/4$ inside a cube of side N. Upon performing the first set of 1D FFTs, the nonzero values form a cylinder of height N but radius $N/4$. Upon performing the second set of 1D FFTs a slab of nonzero data of size $N \times N \times N/4$ is created. The third set of 1D FFT creates a full data set of size $N \times N \times N$. Thus, there are many 1D FFT that yield identically zero and need not be computed; there are many data points that are identically zero and need not be transposed. Indeed, the single transpose required by the plane decomposed parallel algorithm is performed on the cylinder of data where communication load is lowest. This is illustrated in Figure 5.

Finally, for calculations at the Γ point, the states are real and, hence, $\mathbf{\Psi}_i(-g_x, -g_y, -g_z) \equiv \mathbf{\Psi}^*_i(g_x, g_y, g_z)$. Thus, the first set of 1D FFTs is performed on the top half of the sphere. The second set of 1D FFTs is performed on the top half of the cylinder. The third set is, in fact, real to complex 1D FFT, because $\psi(x, y, -g_z) \equiv \psi^*(x, y, g_z)$. Again, both communication and computation are reduced. It is more efficient using Charm++ to create many parallel light-weight objects and, hence, this nonstandard Γ-point optimization is preferred. The standard optimization, also, relies on the fact that the states are real ($\mathbf{\Psi}(\mathbf{r}) = \mathbf{\Psi}^*(\mathbf{r})$). Pairs of real valued states are stored together as a complex number; one state as the real part of a complex number, and the other as the imaginary part, and a complex to complex 3D FFTs performed. This reduces the number of FFT/parallel objects by a factor 2, but increases the communication volume of each 3D FFT by a factor 2 (because the 3D FFT are complex to complex instead of real to complex).

Performing the Multiple Parallel 3D FFTs: Phases I, IV, and VI

The algorithm described above requires multiple 3D FFTs to be performed, one for each state. For each of the n_s 3D FFTs, there is a computational component (the FFT) and a communication step (the transpose). On each processor, planes belonging to different states can reside together without competition because each is

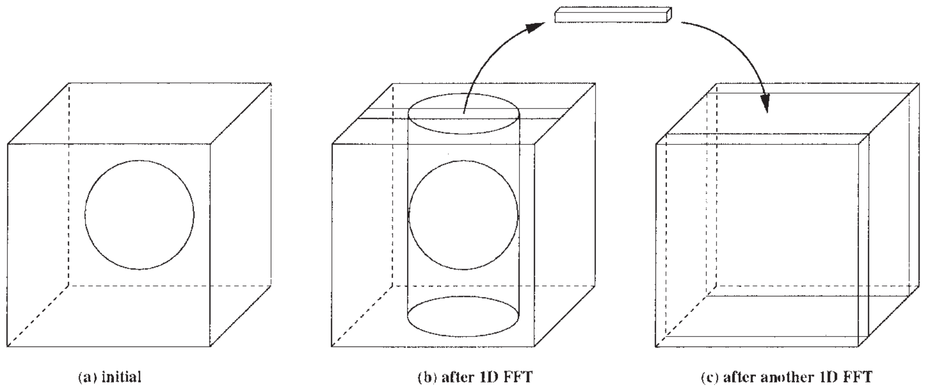


Figure 5. CPAIMD-specific 3D FFT implementation. 1D FFTs in the “j” direction are performed first in the planes. Pencils of data are communicated and two sets of 1D FFTs are done in the “i-k” planes.

involved in an independent 3D FFT. Therefore, the computation of one 3D FFT can easily overlap with the communication of a second 3D FFT, as illustrated in Figure 6. This optimization is particularly important on machines like IBM’s BlueGene, where the nodes have a communication coprocessor that allows the main processor to continue computation while it transfers data via remote direct memory access without processor intervention (as seen in ref. 68). The facile overlap between computation and communication is made possible by the Charm++ embodiment of virtualization, and represents significant advantages over the sequential model that physical scientists have heretofore adopted.[46]

The Construction of Electron Density in Real Space: Phase II

Upon completion of the 3D FFTs, the states are available in real-space and the electron density can be computed, exactly, at each point on the equally spaced real-space grid $\mathbf{r} = \{x, y, z\} = \{i, j, k\}$, as

$$\rho(x, y, z) = \sum_i |\Psi_i(x, y, z)|^2. \tag{25}$$

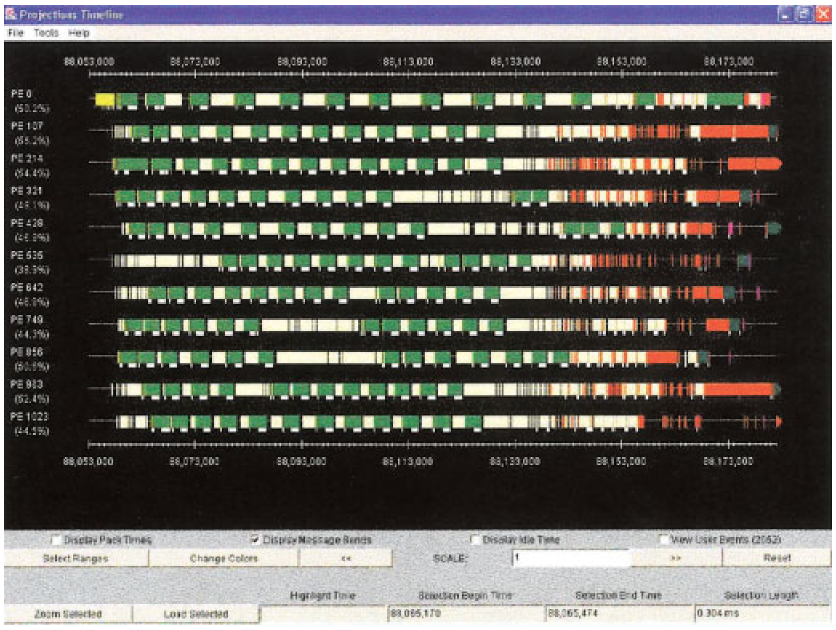


Figure 6. Overlapped transposes in concurrent FFTs in Phase I, as seen in a *timeline* view. The x-axis shows the time, and the y-axis shows messages being processed on a subset of the processors. Some messages (shown with tails) create new messages, the communication of which is overlapped with processing of other messages. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

Because the states are already decomposed into planes, the density is naturally parallelized over each plane index, z, in real-space (e.g., all points $\{x, y\}$ forming a plane with fixed z are together on a VP). However, to form the density, multiple reductions must progress simultaneously; one reduction per plane of points in real-space is required to perform the sum over the states for the plane in question. These simultaneous reductions involve large data sets because the states are dense in real-space. In a system of 32 water molecules, for example (see below), the data sets contain on the order of 10^4 double precision numbers. However, the choice of mapping of VPs to physical processors can be used to reduce the communication in this phase. If all the virtual processors mapped to the same physical processor tend to have the same plane number, p, this will permit the Charm++ run-time system to add many of the contributions together, locally, before sending the local sum(s) to a processor-level reduction over any remaining states with the same plane number assigned to other physical processors (see below). Because Charm++ provides built-in support for reductions among subsets of virtual processors belonging to a single collection (here, states with the same plane number), optimization of the desired reduction can be achieved, seamlessly.

Processing the Electron Density: Phases IV and VI

Once the density, $\rho(x, y, z)$, is constructed, the exchange-correlation energy can be computed. If generalized gradient functionals are employed, then additional terms depending on the three spatial components of the gradient of the density, $\nabla\rho(x, y, z)$, are present. Three additional 3D FFTs are required to compute the gradient, and two more 3D FFTs are required to fully specify its contribution to the KS potential.[55] Concurrent with the calculation of the exchange-correlation energy, the Fourier coefficients of the density are used to determine the Hartree Energy and local external energy and their contribution to the KS potential in reciprocal space. For this operation, a copy of the density is subjected to an inverse 3D FFT (phase IV). Finally, the Hartree Energy and local external energy contributions to the KS potential are transformed back into real-space by another inverse 3D FFT and added to the contribution from exchange correlation. Note, each 3D FFT requires a transpose communication operation.

The Hartree, exchange correlation, and local external energies, thus computed, are summed (in parallel) and multicasts of degree n_s performed to position each of the N planes of KS potential in real-space with the corresponding plane of each of the n_s electronic states. Each (real-space) state objects performs a point by point multiplication by the KS potential on its state and launches an inverse 3D FFT (phase VI), a procedure that is essentially the reverse of a 3D FFT, to generate the forces on states in g-space. The forces are then used to update the states after adding in the forces from the nonlocal energy and the kinetic energy of noninteraction electrons as shown in Figure 2.

Parallelization of the Nonlocal Energy and Kinetic Energy of Noninteracting Electrons: Phase IX

The computation of the nonlocal interaction between the electrons and the nuclei and the kinetic energy of the noninteracting electrons, Phase IX of Figure 2, has been explicitly parallelized because these terms and the forces derived from them can be determined independently from the other phases. The kinetic energy of the noninteracting electrons, eq. (15), is simple. Equation (13) describes the more complex evaluation of the nonlocal energy that is based on an $n_s \times n_N$ matrix, $\mathbf{Z}$, given by eq. (14) where n_s is the number of states and n_N the number of nuclei. The computation of $\mathbf{Z}$ requires a sum reduction over the reciprocal space of the states [cf. eq. (14)].

The nonlocal computation is performed by "particle plane objects." For each g-space plane object, $G(i, g_x)$, where i labels a state and g_x a plane of reciprocal space, a particle plane object, $PP(i, g_x)$, is created that resides on the same physical processor as the g-space plane object. Each particle plane object computes the $\mathbf{Z}$ matrix element for its state and reciprocal space plane. The g-space plane objects' contributions to the $\mathbf{Z}$ matrix are summed across all the planes through a reduction and the final value stored in the first particle plane object of each state, $PP(i, 0)$. The first particle plane object computes the contribution to the nonlocal energy from its state, contributes to the reduction required to determine the total nonlocal energy via eq. (13), and then multicasts the $\mathbf{Z}$ matrix element to all the other planes (e.g., associated with the same state i). Upon receiving the required $\mathbf{Z}$ matrix element, each object computes the nonlocal contribution to the force on its g-space plane and state, $F_{\boldsymbol{\Psi}}(i, g_x)$.

Mapping the Virtual Processors to Physical Processors

An intelligent mapping of VPs to physical processors affords numerous opportunities to optimize load balance and communication overhead simultaneously. The Charm++ run-time system has a default mapping function of virtual processors to real processors that the user can tune/override. In addition, the Charm++ run-time system is capable of dynamically changing the mapping to achieve an optimal load balance, due to unexpected changes in the computation or simply due to a less than optimal default mapping function. In a CPAIMD computation, the load on each virtual processor is predictable and static (i.e., it does not change during the run). Consequently, it proves advantageous to define an optimal mapping function manually (at the implementation level) for this algorithm.

The mapping of the real-space quantities is influenced by the multicast (Fig. 3, phase V) from the KS potential, $v_{KS}(\mathbf{r})$, on the real-space grid to each of the states on the real-space grid, which is required to generate the forces on states from the exchange correlation and Hartree energies. It is desirable to have as many multicast target objects on the same physical processor as possible. The targets in this case are the planes in real-space, coming from different states but belonging to the same spatial region, that is having the same plane index but different state index (e.g., both the states the KS potential are parallelized by plane and they possess the same total number of planes, N). This leads to a mapping function

$$\mathrm{pe}(R(i, p)) = \mathcal{P} \times \left[\frac{p \times n_s + i}{N \times n_s}\right] \tag{26}$$

where pe is the physical processor number, $\mathcal{P}$ is the total number of physical processors, and p is the plane index. Note that the mapping function could have been chosen to minimize the communication that occurs during the multiple concurrent transposes associated with the 3D FFTs (phases I and VI) by assigning multiple real-space planes with same state index but different plane index together. This option is not preferable to eq. (26) because the amount of communication bandwidth involved in the transpose phase is small due to the sparse data set in g-space provided by the use of a spherical cutoff.

The mapping of the g-space objects is slightly more complicated. Earlier, it was observed that $S^{(\mathrm{aux})}_{i/\gamma,j/\gamma}(p)$ requires communication with $G(i\cdots(i+\gamma-1), p)$ and $G(j\cdots(j+\gamma-1), p)$. Hence, this part of the computation would benefit if all the $G(i, p)$ chares were distributed in a manner similar to $R(i, p)$. Unfortunately, using a mapping similar to that in eq. (26) causes a problem of load imbalance: the distribution of computational work among planes in g-space should be allocated based on the number of g-space points in discrete planar sections of a sphere. That is, the spherical truncation of the reciprocal space described, in detail, in the CPAIMD Method section, yields an unequal distribution of nonzero elements in the planes. Some g-space planes are very densely populated and some g-space planes are very sparsely populated. To treat this load imbalance, a different mapping is required. Details of a mapping designed to distribute the g-space planes more optimally is presented in the Performance and Optimization Section, where the performance tools provided by Charm++ are employed to diagnose poor mappings and to suggest improvements.

Optimizing Multicasts Operations in the Parallel Scheme

The multicast operations (Fig. 3, phase V) are each-to-many type of communication operations of degree n_s. They can be performed[69] in several ways.

1. **Direct:** Each of the N planes of ρ sends n_s messages to the corresponding real-space planes. This approach floods the network card with messages, which causes inordinate delays in message delivery.
2. **K-send:** Each of the N planes sends k messages, waits for a prespecified amount of time, and then sends again. This eases the network congestion and, hence, improves on the direct approach. This method is problematic, as the waiting period is difficult to gauge correctly and the problem of network contention is not addressed.
3. **Ring:** Each of the N planes sends data to one of the n_s states, which is then forwarded to another state. The advantage of this approach is that at any stage the number of messages in flight is equal to the number of processors, P, and network contention is reduced.
4. **Multiring:** The diameter of the ring determines the completion time of the multicast. To reduce the completion time, multiple rings of smaller diameters are created that perform the multicast in parallel. This approach works well with the mapping scheme described for real-space planes, where several "next-hops" in the ring reside on the same processor.

All of the above methods were tested. The ring method was found to function most effectively. Note that the all-pairs communication pattern involved in orthonormalization (for which the ring was not a good algorithm) is different from the communication pattern of the multicast. In all the multicast strategies described above, the total volume of communication entering a processor is identical. In contrast, the **block-pairs** approach reduces total volume of communication compared to the all-pairs pattern.

Performance and Optimization of Fine-Grained Parallel CPAIMD

Description of the Physical System

To test the novel fine-grained parallel CPAIMD algorithm described in the previous section, a 32-water molecule system was selected for study. This size system is ubiquitous in the literature because it is just large enough to study the properties of water and small solutes in water without finite size effects. Indeed, pioneering work on proton transfer in water and the autodissociation of water was performed in a periodic box containing 32 water molecules.[28–30] The standard cutoff, $E_{\mathrm{cut}} = 70$ Ry was employed in conjunction with a standard set of pseudopotentials (Troullier–Martins). The cutoff gives rise to 32,000 g-space orbital coefficients per state, 260,000 g-space density coefficients, and a real-space grid of 100^3 points. Exchange and correlation was treated using the generalized gradient exchange functional of Becke[50] and the generalized gradient correlation functional Lee, Yang, and Parr.[51]

Description of the Computer

The new software was tested on PSC-Lemieux, a 750 node, 3000 processor cluster. Each node in Lemieux is a Quad 1 Ghz Alpha server connected by Quadrics Elan, a high-speed interconnect with 4.5-μs latency.

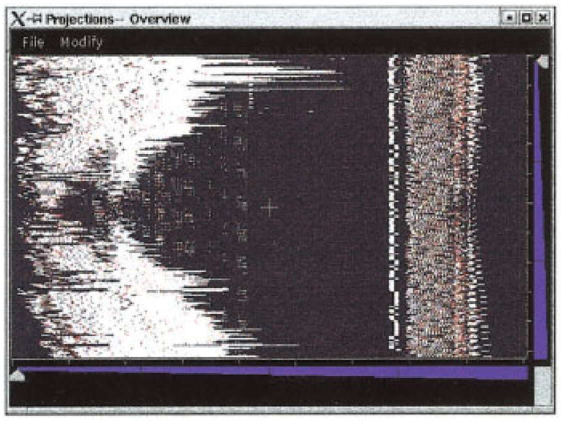
Load Balance

It is clear from the discussion of the previous section that Charm++'s mapping of the VPs to physical processors is a key point in algorithm optimization. Here, different mapping schemes are explored and the optimal mapping to achieve good load balance determined.

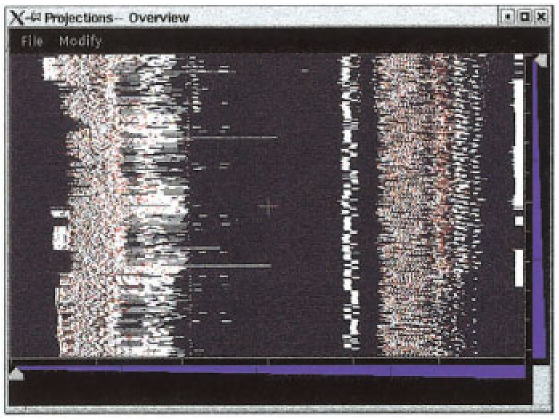
The effect of different mapping schemes is shown in Figure 7, with "overviews" created by the *Projections* performance tool, for a set of 1024-processor runs. Each horizontal line corresponds to one processor, with the x-axis showing progress in time (white indicates a busy processor). Using a map similar to eq. (26) causes a load imbalance. The problem is seen clearly in Figure 7a, where the middle processors are significantly underloaded compared with processors 0 and 1023 due to the sparse data distribution in the g-space planes.

The load imbalance was partially alleviated by using a modified version of eq. (26):

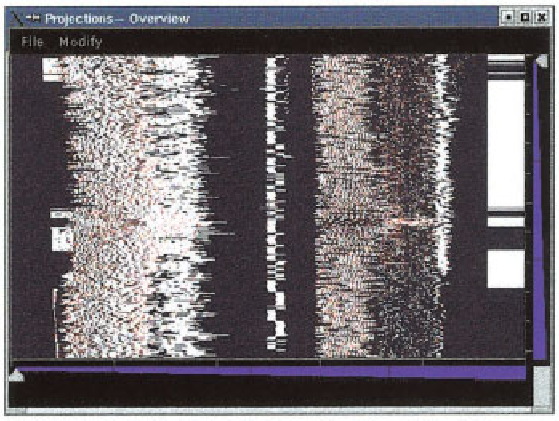
$$\mathrm{pe}(R(i, p)) = \left(4\mathcal{P} \times \left[\frac{p \times n_s + i}{N \times n_s}\right] \% \mathcal{P}\right) \qquad (27)$$



(a) Load imbalance due to phases I and IX (900ms)



(b) Improvement by using “wrapping”(590ms)



(c) Final result with load-vectors(480ms)

Figure 7. Solving the problem of load imbalance on 1024 processors.

This scheme assumes four virtual processors will be spawned per physical processors, P, where p is the plane number. The virtual processor number is “wrapped” over the available number of physical processors using periodic boundary conditions, thus diversifying the sizes of planes present on each processor (e.g., sparse planes and dense planes are intermingled). The result of this mapping is seen in Figure 7b. Although better than before, it is clear (by looking at a few long lines in the overview figure) that it does not achieve good load balance, leaving a few overloaded processors.

Next, the number of nonzero lines on each plane (alluded to in section 3.10) is explicitly considered. A “load-vector” $\mathscr{L}$ and a “cumulative” load-vector $\mathscr{C}$ of size N was defined and a mapping developed in terms of these new variables.

$$\mathscr{L}[p] = \text{number of nonzeros in plane } p \text{ over all states}$$

$$\mathscr{C}[p] = \sum_{q<=p} \mathscr{L}[q]$$

$$\mathrm{pe}(G(i,p)) = \frac{\mathscr{C}[p-1] + \frac{i}{n_s} \times \mathscr{L}[p]}{l} \tag{28}$$

where

$$l = \frac{\sum_p \mathscr{L}[p]}{\mathscr{P}}$$

is the desired average load per processor. The best performance, as might be expected, was obtained through this explicit consideration of the load through eq. (28) as demonstrated in Figure 7.

It is clear that transforming the knowledge of the number of nonzero lines in each plane into a balanced map is, in general, difficult. In the future, Charm++’s automatic load balancers will be modified to allow the user to specify partial mappings (as

Table 1. Execution Times on PSC Lemieux, for 128 States.

N	P	P/N	*t* (s)	GFLOPS	Speedup	N	P	P/N	*t* (s)	GFLOPS	Speedup
8	16	2	13.26	4	16.0	43	129	3	2.20	22	96.4
16	32	2	6.17	8	34.3	83	258	3	1.30	37	163.2
32	64	2	3.11	15	68.2	171	513	3	0.68	71	312.0
64	128	2	2.07	23	102.5	256	768	3	0.60	80	353.6
128	256	2	1.18	41	179.8	512	1536	3	0.40	121	533.1
256	512	2	0.65	74	326.4						
512	1024	2	0.48	100	442.0						

Using more processors, P, per node, N, effects performance adversely, as the bandwidth is a limited resource.

required for R, explained earlier). The preliminary results in this directions have shown some promise.

Scaling Performance

Table 1 shows the scalability of the code using all the optimizations described in the text. A peak floating point operations rate of 121 GFlops on 1536 processors was obtained. Although this is a small fraction of the machine's capability, it is acceptable due to the communication intensive nature of this application. The performance of the code degrades only slightly as the number of processors per node is increased from two to three. Upon full utilization of all four processors per node, performance is seriously degraded due to intermittent system calls (not shown), a known problem with the Lemieux software, that is overcome by leaving one processor free (e.g., using at most only three processors per node) to handle these operations.

Communication Analysis of Optimized Implementation

Figure 8 shows the communication behavior of the code on 1024 processors. There is a total of 6.6 GB of nonlocal or interprocessor communication per iteration of the computation. During the same iteration the maximum data received *per processor* is as high as 11.5 MB.

Notice that there are two prominent modes message sizes in the plot: the messages related to the 3D FFTs have sizes about 1 KB, and the messages related to the matrix computations of the orthonormalization phase have sizes around 20 KB. These constitute the largest fraction of the nonlocal communication.

Also observe that there are relatively fewer messages that have sizes greater than 100 KB. The multicast in phase V and orthonormalization are responsible for these large messages. However, their number is smaller compared with the other phases mentioned above. This can be accredited to the mapping scheme used for real-space planes, eq. (26), along with the multicast strategy (ring), which causes most of the hops in the ring to be on the same physical processor.

To arrive at these results, several optimizations, both algorithmic and platform-specific, were performed. The load-balance related optimizations are explained in detail above. The optimizations were motivated and enabled by Charm++'s internal performance tools that are designed to pinpoint bottlenecks in calculations run on large-scale parallel architectures.

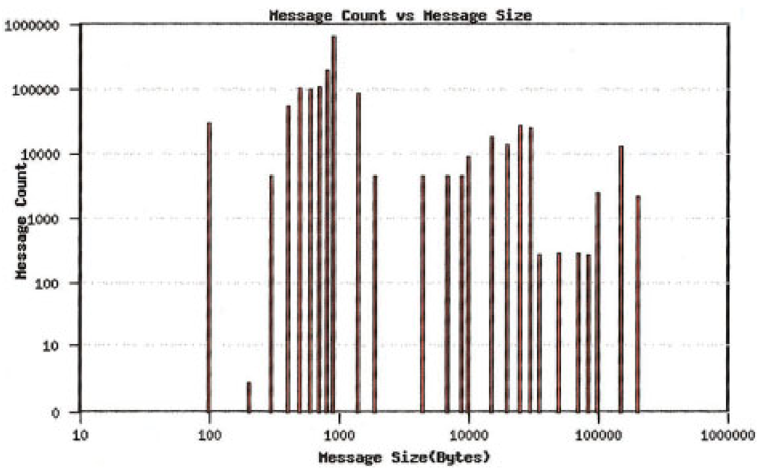


Figure 8. Total number of nonlocal messages in one iteration for different message sizes. Data is obtained on 1024 processors, using mapping scheme described in eq. (26) and a ring multicast strategy. There is a total of 6.6-GB nonlocal communication per iteration.

Conclusion

The large-scale, fine-grained, parallelization of the Car–Parrinello molecular dynamics method[15] has been described. Using the Charm++ parallel middleware and its powerful embodiment of the concept of virtualization and, hence, its natural ability to dynamically load balance and interleave computation and communication, parallel speed up on up to 1500 processors has been demonstrated for a 32-water molecule system. This is a remarkable achievement, given that the system only contains 128 electronic states and standard approaches do not scale well beyond 64 processors.[46] It is important to note that this system size is typically employed to study aqueous solvation and chemistry of relevant small molecules, and has been employed in the ground-breaking work that elucidated the mechanism of proton transport in water.[29,30,70] The improvement in scaling presented, herein, will therefore be employed to explore atomic level processes of this general class of systems at long time scales. In addition, given the advent of large scale computer systems containing 10,000 to 64,000 processors such as IBM's BlueGene,[45] the new parallelization scheme will prove invaluable in the study of larger systems at the long time scales required to observe key dynamical and structural properties that have been, heretofore, unapproachable.

Acknowledgment

Computer time is gratefully acknowledged from NSF, the Camille and Henry Dreyfus Foundation, Inc., and New York University's Research Challenge Fund. G.J.M. and M.E.T. would like to thank O. Lawlor and G. Zheng for their suggestions and timely assistance, as well as the Kale group for their hospitality during the coauthors' visits to UIUC.

References

1. Rahman, A. Phys Rev A 1964, 136, 405.
2. Allen, M. P.; Tildesley, D. J. Computer Simulations of Liquids; Clarendon Press: Oxford, 1989.
3. Goldstein, H. Classical Mechanics; Addison-Wesley Pub. Co.: Reading, MA, 1980.
4. McQuarrie, D. A. Statistical Mechanics; Harpers & Row: New York, 1976.
5. MacKerell, A., Jr., et al. J Phys Chem B 1998, 102, 3586.
6. Chen, B.; Martin, M. G.; Siepmann, J. I. J Phys Chem B 1998, 102, 2578.
7. Cornell, W. D.; Cieplak, P.; Bayly, C. I.; Gould, I. R.; Mertz, K. M.; Ferguson, D. M.; Spellmeyer, D. C.; Fox, T.; Caldwell, J. W.; Kollman, P. A. J Am Chem Soc 1995, 117, 5179.
8. Bernal, J. D.; Fowler, R. H. J Chem Phys 1933, 1, 515.
9. Stillinger, F. H.; David, C. J Chem Phys 1979, 71, 1647.
10. Sprik, M. J Phys Chem 1991, 95, 2283.
11. Deng, Z.; Klein, M. L.; Martyna, G. J. J Chem Phys 1994, 100, 7590.
12. Catlow, C. R. A.; Freeman, C. M.; Vessal, B.; Tomlinson, S. M.; Leslie, M. J Chem Soc Faraday Trans 1991, 87, 1947.
13. Madden, P. A.; Wilson, M. Chem Soc Rev 1996, 25, 339.
14. Warshel, A.; Weiss, R. M. J Am Chem Soc 1980, 102, 6218.
15. Car, R.; Parrinello, M. Phys Rev Lett 1985, 55, 2471.
16. Remler, D. K.; Madden, P. A. Mol Phys 1990, 70, 921.
17. Payne, M. C.; Teter, M.; Allan, D. C.; Aria, T. A.; Joannopolous, J. D. Rev Mod Phys 2002, 64.
18. Galli, G.; Parrinello, M. Comput Simulat Mater Sci 1991, 3, 283.
19. Tuckerman, M. E.; Ungar, P. J.; von Rosenvinge, T.; Klein, M. L. J Phys Chem 1996, 100, 12878.
20. Gillan, M. J. Contemp Phys 1997, 38, 115.
21. Parrinello, M. Solid State Commun 1997, 102, 107.
22. Marx, D.; Hutter, J. In Modern Methods and Algorithms of Quantum Chemistry; Grotendorst, J. Ed.; Forschungszentrum: Juelich (NIC Series), 2000, p. 301, vol. 1.
23. Car, R. Quant Struct Act Rel 2002, 21, 97.
24. Tuckerman, M. E. J Phys Condensed Matter 2002, 14, R1297.
25. Silvestrelli, P. L.; Parrinello, M. Phys Rev Lett 1999, 82, 3308.
26. Pasquarello, A.; Petri, I.; Salmon, P. S.; Parisel, O.; Car, R.; Toth, E.; Powell, D. H.; Fischer, H. E.; Heim, L.; Merbach, A. E. Science 2001, 291, 856.
27. Deng, Z.; Martyna, G. J.; Klein, M. L. Phys Rev Lett 1992, 68, 2496.
28. Marx, D.; Tuckerman, M. E.; Hutter, J.; Parrinello, M. Nature 1999, 367, 601.
29. Geissler, P. L.; Dellago, C.; Chandler, D.; Hutter, J.; Parrinello, M. Science 2001, 291, 2121.
30. Tuckerman, M. E.; Marx, D.; Parrinello, M. Nature 2002, 417, 925.
31. Boero, M.; Parrinello, M.; Terakura, K. J Am Chem Soc 1998, 120, 2746.
32. Charlier, J. C.; De Vita, A.; Blase, X.; Car, R. Science 1997, 275, 646.
33. De Vita, A.; Galli, G.; Canning, A.; Car, R. Nature 1996, 379, 523.
34. Galli, G.; Martin, R. M.; Car, R.; Parrinello, M. Science 1990, 250, 1547.
35. Carloni, P.; Bloechl, P. E.; Parrinello, M. J Phys Chem 1995, 99, 1338.
36. Alfe, D.; Gillan, M. J.; Price, G. D. Nature 1999, 401, 462.
37. Cavazzoni, C.; Chiarotti, G. L.; Scandolo, S.; Parrinello, M. Science 1999, 283, 44.
38. Kohn, W.; Sham, L. J. Phys Rev 1965, 140, A1133.
39. Parr, R. G.; Yang, W. Density Functional Theory of Atoms and Molecules; Oxford University Press: Oxford, 1989.
40. Dreizler, R. M.; Gross, E. K. U. Density Functional Theory; Springer-Verlag: Berlin, 1990.
41. Warshel, A.; Levitt, M. J Mol Biol 1976, 103, 227.
42. Yarne, D. A.; Tuckerman, M. E.; Martyna, G. J. J Chem Phys 2001, 115, 3531.
43. Zhang, Y.; Lee, T.; Yang, W. J Chem Phys 1999, 110, 46.
44. Eichinger, M.; Tavan, P.; Hutter, J.; Parrinello, M. J Chem Phys 1999, 110, 10452.
45. Allen, F., et al. IBM Syst J 2001, 40, 310.
46. Tuckerman, M. E.; Yarne, D. A.; Samuelson, S. O.; Hughes, A. L.; Martyna, G. J. Comp Phys Comm 2000, 128, 333.
47. Kale, L. V.; Krishnan, S. Sigplan Notices 1993, 28, 91.
48. Kale, L. V.; Bhandarkar, M.; Brunner, R. Lect Notes Comput Sci 1998, 1388, 111.
49. Perdew, J. P.; Zunger, A. Phys Rev B 1981, 23, 5048.
50. Becke, A. D. Phys Rev A 1988, 38, 3098.
51. Lee, C.; Yang, W.; Parr, R. G. Phys Rev B 1988, 37, 785.
52. Bachelet, G.; Hamann, D.; Schluter, M. Phys Rev B 1982, 26, 4199.
53. Martyna, G. J.; Tuckerman, M. E. J Chem Phys 1999, 110, 2810.
54. Minary, P.; Tuckerman, M. E.; Pihakari, K. A.; Martyna, G. J. J Chem Phys 2002, 116, 5351.
55. White, J. A.; Bird, D. M. Phys Rev B 1994, 50, 4954.
56. Kleinman, L.; Bylander, D. M. Phys Rev Lett 1982, 48, 1425.
57. Hansen, J. P. In MD Simulations of Statistical Mechanical Systems. North Holland Physics: Amsterdam, 1986.
58. deLeeuw, S. W.; Perram, J. W.; Smith, E. R. Proc R Soc Lond A 1980, 373, 27.

59. Marx, D.; Tuckerman, M. E.; Martyna, G. J. Comp Phys Commun 1999, 118, 166.
60. Tuckerman, M. E.; Parrinello, M. J Chem Phys 1994, 101, 1301.
61. Tuckerman, M. E.; Parrinello, M. J Chem Phys 1994, 101, 1316.
62. Tuckerman, M. E.; Hutter, J.; Parrinello, M. J Chem Phys 1955, 102, 859.
63. Lawlor, O.; Kalé, L. V. In Proceedings of International Symposium on Computing in Object-Oriented Parallel Environments, Stanford, CA, June 2001.
64. Sky Lawlor, O.; Kalé, L. V. Concurr Comput: Practice Exp 2003, 15, 371.
65. Board, J. A.; Cramer, C. E. The 4th Annual Linux Showcase and Conference, 2000, p. 121.
66. Haynes, P. D.; Cote, M. Comput Phys Commun 2000, 130, 130.
67. Frigo, M.; Johnson, S. G. The Fastest Fourier Transform in the West. Technical Report MIT/LCS/TR-728, 1997.
68. Kale, L. V.; Kumar, S.; Vardarajan, K. A Framework for Collective Personalized Communication, Communicated to ipdps 2003. Technical report, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana–Champaign, 2002.
69. Kale, L. V.; Kumar, S. Scaling Collective Multicast on High Performance Clusters. Technical report, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana–Champaign, 2003.
70. Tuckerman, M. E.; Marx, D.; Klein, M. L.; Parrinello, M. Science 1997, 275, 817.