Parallel Computing 38 (2012) 245–259

Contents lists available at SciVerse ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques

H.M. Aktulga [a,*], J.C. Fogarty [b], S.A. Pandit [b], A.Y. Grama [a,**]

[a] Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA
[b] Department of Physics, University of South Florida, Tampa, FL 33620, USA

A R T I C L E I N F O

Article history:
Received 23 May 2010
Received in revised form 6 April 2011
Accepted 25 August 2011
Available online 28 September 2011

Keywords:
Scientific computing
High performance computing
Parallel scaling and performance
Molecular dynamics
Reactive force fields
Application of algorithms
Numerical techniques
Krylov subspace methods

A B S T R A C T

Molecular dynamics modeling has provided a powerful tool for simulating and understanding diverse systems – ranging from materials processes to biophysical phenomena. Parallel formulations of these methods have been shown to be among the most scalable scientific computing applications. Many instances of this class of methods rely on a static bond structure for molecules, rendering them infeasible for reactive systems. Recent work on reactive force fields has resulted in the development of ReaxFF, a novel bond order potential that bridges quantum-scale and classical MD approaches by explicitly modeling bond activity (reactions) and charge equilibration. These aspects of ReaxFF pose significant challenges from a computational standpoint, both in sequential and parallel contexts. Evolving bond structure requires efficient dynamic data structures. Minimizing electrostatic energy through charge equilibration requires the solution of a large sparse linear system with a shielded electrostatic kernel at each sub-femtosecond long time-step. In this context, reaching spatio-temporal scales of tens of nanometers and nanoseconds, where phenomena of interest can be observed, poses significant challenges.

In this paper, we present the design and implementation details of the Purdue Reactive Molecular Dynamics code, PuReMD. PuReMD has been demonstrated to be highly efficient (in terms of processor performance) and scalable. It extends current spatio-temporal simulation capability for reactive atomistic systems by over an order of magnitude. It incorporates efficient dynamic data structures, algorithmic optimizations, and effective solvers to deliver low per-time-step simulation time, with a small memory footprint. PuReMD is comprehensively validated for performance and accuracy on up to 3375 cores on a commodity cluster (Hera at LLNL-OCF). Potential performance bottlenecks to scalability beyond our experiments have also been analyzed. PuReMD is available over the public domain and has been used to model diverse systems, ranging from strain relaxation in Si–Ge nanobars, water–silica surface interaction, and oxidative stress in lipid bilayers (bio-membranes).

1. Introduction

Conventional atomistic modeling techniques rely on quantum–mechanical methods or on traditional molecular dynamics approaches. Quantum-scale modeling requires computationally expensive solution of the electronic Schrödinger equation,

* Corresponding author.
** Principle corresponding author.
E-mail addresses: hmaktulga@lbl.gov (H.M. Aktulga), jcfogart@mail.usf.edu (J.C. Fogarty), pandit@cas.usf.edu (S.A. Pandit), ayg@cs.purdue.edu (A.Y. Grama).

0167-8191/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2011.08.005

restricting its applicability to systems on the order of thousands of atoms and picosecond simulation timeframes. Classical molecular dynamics (MD) approaches, on the other hand, overcome the time and size limitations of quantum methods by approximating the nucleus together with its electrons into a single basis. These methods rely on careful parametrization of various atomic interactions corresponding to bonds, valence angles, torsion, van der Waals interactions, etc. based on detailed quantum mechanical simulations. In spite of these approximations, classical MD methods have been successful in elucidating various phenomena inaccessible to either quantum methods or to experiments.

Classical MD approaches typically rely on static bonds and fixed partial charges associated with atoms. These constraints limit their applicability to non-reactive systems only. A number of recent efforts have addressed this limitation [1–5]. Among these, ReaxFF, a novel reactive force field developed by van Duin et al. [6], bridges quantum-scale and classical MD approaches by explicitly modeling bond activity (reactions) and charge equilibration. The flexibility and transferability of the force field allows ReaxFF to be easily extended to systems of interest. Indeed, ReaxFF has been successfully applied to diverse systems, ranging from materials modeling to biophysical systems [6–9].

ReaxFF is a classical MD method in the sense that atomic nuclei, together with their electrons, are modeled as basis points. Interactions among atoms are modeled through suitable parametrizations and atomic motion obeys laws of classical mechanics. Accurately modeling chemical reactions, while avoiding discontinuities on the potential energy surface, however, requires more complex mathematical formulations of interactions than those found in most classical MD methods (bond, valence angle, dihedral, van der Waals potentials). In a reactive environment, in which atoms often do not achieve their optimal coordination numbers, ReaxFF requires additional modeling abstractions such as lone pair, over/under-coordination, and 3-body and 4-body conjugation potentials, which further increase its computational complexity. This increased computational cost of bonded interactions (due to the reconstruction of all bonds, 3-body and 4-body structures at each time-step and much more complex bonded interaction formulations) approaches the cost of non-bonded interactions for ReaxFF, as we discuss in Section 5. Note that for typical conventional MD codes, the time spent on bonded interactions is significantly lower than that spent on non-bonded interactions [26].

An important part of ReaxFF is the charge equilibration procedure. Charge equilibration (QEq) procedure [17] approximates the partial charges on atoms by minimizing the electrostatic energy of the system. Charge equilibration is mathematically formulated as the solution of a large sparse linear system of equations. This solve needs to be performed accurately at each time-step because it significantly impacts forces and total energy of the system. Since partial charges on atoms are fixed in conventional MD, this is not a consideration for conventional methods. The time-step lengths for ReaxFF simulations is typically an order of magnitude smaller than conventional MD (tenth of femtoseconds as opposed to femtoseconds), therefore scaling the solve associated with charge equilibration is a primary design consideration for parallel formulations of ReaxFF. Suitably accelerated Krylov subspace methods are used for this purpose (see Section 4.5). One of the major challenges overcome by our effort is the scaling of this solve to thousands of processing cores.

In this paper, we present the design and implementation details of the PuReMD (**Pu**rdue **Re**active **M**olecular **D**ynamics) code, along with a comprehensive evaluation of its performance on a large commodity cluster (Hera at the Lawrence Livermore National Laboratory's Open Computing Facility, LLNL-OCF) using over 3000 processing cores. PuReMD incorporates several algorithmic and numerical innovations to address significant computational challenges posed by ReaxFF. It achieves excellent per time-step execution times, enabling nanosecond-scale simulations of large reactive systems (Section 5). Using fully dynamic interaction lists that adapt to the specific needs of simulations, PuReMD achieves low memory footprint. Our tests demonstrate that PuReMD is up to five times faster than competing implementations, while using significantly lower memory.

The rest of this paper is organized as follows: we overview related work on parallel ReaxFF implementations in Section 2. In Section 3, we discuss critical design choices for parallelization of ReaxFF. In Section 4, we outline various algorithms and numerical techniques used to achieve excellent computational times per simulation time-step. We comprehensively analyze the performance of PuReMD in Section 5. We conclude with a discussion of potential bottlenecks to further scaling, solutions to these bottlenecks, and techniques for further improvements to overall simulation time.

2. Related work

The first-generation ReaxFF implementation of van Duin et al. [6] demonstrated the validity of the method in the context of various applications. Thompson et al. [10] successfully ported this initial implementation, which was not developed for a parallel environment, into their parallel MD package LAMMPS [12]. Except for the charge equilibration part, the ReaxFF implementation in LAMMPS is based on the original FORTRAN code of van Duin [6], significant portions of which were included directly (as Fortran routines called from C++) to insure consistency between the two codes since the Fortran ReaxFF underwent rapid development at the time. Thus the LAMMPS implementation does not take advantage of certain optimizations described in this paper. The charge equilibration calculation currently in LAMMPS uses a standard parallel conjugate gradient algorithm for sparse linear systems [11]. Indeed, efforts are under way to integrate our kernel parallel ReaxFF functions into LAMMPS, in close collaboration with Aidan Thompson at Sandia National Labs. Our preliminary tests on the sample systems discussed in this paper reveal that PuReMD is up to five times faster than LAMMPS on a single processor. Detailed single-processor performance comparisons of the two codes are presented in [14]. In Section 5, we present comparisons of LAMMPS and PuReMD codes under weak and strong scaling scenarios.

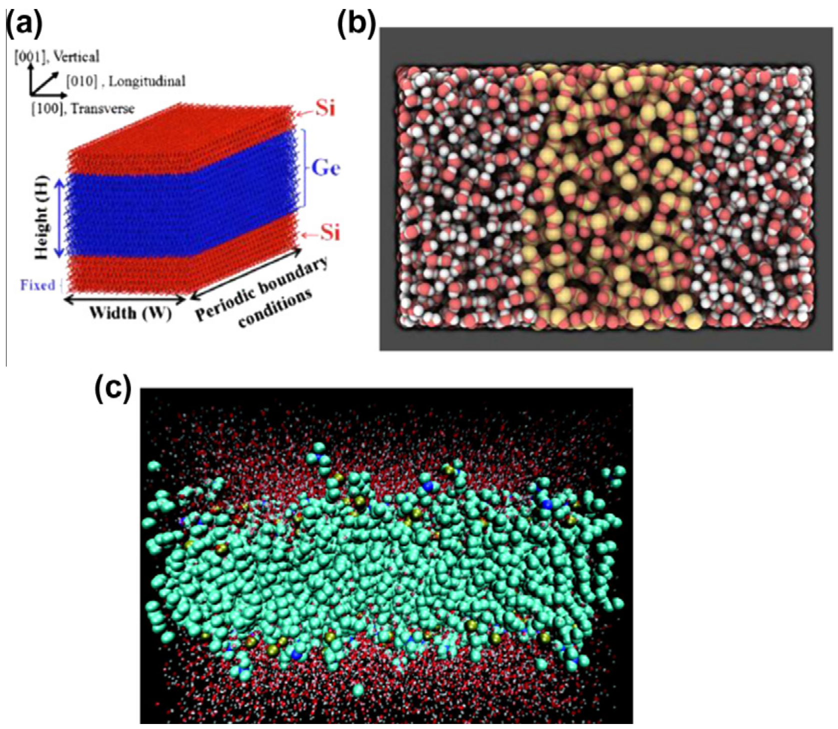


Fig. 1. Applications using PuReMD: (a) Si–Ge nanobar system, (b) Water–silica interface, (c) lipid bilayer system.

To the best of our knowledge, the only other reported parallel ReaxFF formulation is due to Nomuro et al. [13], where they demonstrate good scaling results. However, their per time-step per atom execution times are up to an order of magnitude slower than the results we report in this paper. Consequently, even though their reported efficiencies at approximately 3000 cores are higher than ours (our code achieves about 80% efficiency), their simulation time is significantly higher than that reported in this paper.

PuReMD is a publicly available parallel ReaxFF implementation with demonstrated scalability to thousands of processing cores. It has been validated by us and by other research groups on diverse systems, ranging from strain relaxation in Si–Ge nanobars [15], water–silica systems [16], and oxidative stress in lipid bilayers (membranes) (Fig. 1).

3. Parallel formulation of ReaxFF

In this section, we discuss the two important aspects of parallel ReaxFF implementation – problem decomposition and inter-process synchronization/communication. We refer to the domain of simulation specified in input as the *simulation box*, and the part assigned to a process under 3D domain decomposition as the *sub-domain* of that process. A significant fraction of the computation associated with an atom involves other atoms within a prescribed distance from the source atom. To facilitate these computations, we construct a list of neighbors for each atom. These neighbor lists are generated by embedding a 3D grid within each process' sub-domain. Partitions induced by this 3D grid are called *cells* or *grid cells*.

Problem decomposition and inter-process communication patterns are determined by the mathematical formulations of various interactions in ReaxFF. Precise mathematical details of the energy and force formulations in ReaxFF are beyond the scope of this paper; we refer the readers to the original ReaxFF formulation described in [6]. An important aspect of ReaxFF that significantly impacts design choices is that it uses shielded electrostatics, modeled by range-limited pairwise interactions with Taper corrections. This obviates the need for the computation of long range electrostatic interactions.

3.1. Problem decomposition

Commonly used decomposition techniques for MD simulations include atom decomposition, interaction decomposition, and domain decomposition. The specific choice among these is influenced by the characteristics of the force field, with a view to minimizing communication overhead and load imbalance. While atom and force decomposition techniques deliver good load balance, movement of atoms and associated data results in a highly dynamic inter-process communication pattern which is generally handled by periodic re-decompositions. However, the simplicity of domain decomposition techniques coupled with its natural handling of the range-limited interactions make it a popular choice [22,24,25]. Load balancing in domain decomposition is achieved by suitably partitioning the simulation box into sub-domains with almost equal computational loads. Volumes of sub-domains are altered dynamically to ensure equal workload among processes as the simulation progresses.

Atom or force decomposition techniques are not well-suited to ReaxFF implementations primarily due to the dynamic nature of bonds. The presence of charge equilibration and associated linear system solve, which takes up a significant portion of the total simulation time, poses additional considerations not present in conventional MD codes. PuReMD adopts a 3D domain decomposition technique with wrap-around links (a torus) for periodic boundary conditions. This domain decomposition also induces a partition of the degrees of freedom for the parallel solution of the linear system associated with the charge equilibration problem.

3.2. Boundary regions

In addition to problem decomposition, a number of further design choices critically impact the performance of PuReMD. These primarily relate to handling of inter-process communications and synchronizations.

- **Interactions spanning process boundaries:** In order to avoid unnecessary computations while ensuring accurate calculation of energy and forces resulting from interactions spanning process boundaries, an efficient coordination mechanism among processes must be in-place. Since interactions are primarily range-limited, a shell of the process' sub-domain is associated with inter-process interactions. We refer to this shell as the outer-shell of a process sub-domain. The way communications (one-sided vs. symmetric data transfers) and computations (redundant computation of symmetric terms vs. communication of computed terms) associated with the outer shell is handled has significant impacts on performance.
- **Inter-process communication**: Once we determine atoms whose information must be communicated based on the outer-shell type chosen, communication can take place either through direct messaging or in a staged manner [25].

We now discuss these issues in more detail and explain how we handle each issue in PuReMD.

3.2.1. Interactions spanning multiple processes

We first describe our handling of interactions that span multiple processes, since this motivates our choice of the outer-shell type. We specifically focus on bond-order potentials, and associated dynamic bonded interactions in ReaxFF because handling range-limited non-bonded interactions have already been well-studied in literature [23–25]. After carefully analyzing different ways of handling bonded interactions in ReaxFF that span multiple processes, we outline the scheme used in PuReMD below:

- **Bond(i,j):** The process that owns the atom with the smaller index (indices are unique and are determined by a field in the input file) handles the bond.
- **LonePair(i):** This is a single body potential and the owner of atom i computes the energy and forces resulting from the unpaired electrons of atom i.
- **Over/Under-coordination(i):** These are multi-body interactions, directly involving all bonded neighbors of atom i, computed by the owner of atom i.
- **Valence Angle(i,j,k):** This includes the valence angle, penalty, and three-body conjugation potentials, all of which are computed by the owner of middle atom j.
- **Dihedral Angle(i,j,k,l):** This includes the torsion and four-body conjugation potentials, both of which are handled by the owner of middle atom with the smaller index. Middle atoms here are j and k.
- **Hydrogen Bond(x,H,z):** Here the hydrogen atom H is bonded to atom x and interacts with the atom z through a hydrogen bond. The presence of a bond between atoms x and H implies that the process owning the H atom computes this hydrogen bond interaction.
- **Non-bonded(i,j):** As in the bonded case, the owner of the atom with the smaller index computes the van der Waals and Coulomb interactions between atoms i and j.

Establishing this coordination mechanism enables us to avoid redundant computation of interactions straddling process boundaries. The ratio of such interactions to those entirely within process sub-domains can be significant, especially as sub-domain volumes decrease. The potential drawback of this approach is the reverse messages containing forces required at the end of each time-step, when processes need to compute the total force on each atom assigned to them. We adopt this approach in PuReMD because force computations in ReaxFF are relatively expensive compared to the associated additional reverse communication step.

While we try to avoid redundant computations of expensive potential terms, we intentionally performed redundant computations during the matrix–vector multiplications associated with the iterative charge equilibration solver. This was done in order to avoid the reverse communication step, which is much more expensive than the cost of redundant computations. This strategy results in slightly worse performance on small numbers of processing cores due to redundant computations; however, as we scale to larger number of cores, it delivers better performance and scaling results by eliminating a costly communication step, which might have to be executed tens of times by the iterative solver in a single time-step.

3.2.2. Choosing the outer-shell

The range-limited nature of force fields, associated symmetries, and relative speed of computation and communication of a parallel platform motivate the choice of full-shell, half-shell, midpoint-shell or neutral territory (NT) methods [27], see (Fig. 2). In full-shell methods, interactions between atoms *i* and *j* are computed at processes that hold both atoms *i and j*. This implies that data needed to compute these interactions must be symmetrically exchanged, resulting in higher communication costs. However, where such interactions are symmetric, the results do not need to be communicated. In half-shell methods, interactions between atoms *i* and *j* are computed at the process responsible for either atom *i* or atom *j*. This choice is uniformly enforced by convention (in our case for example, the process that owns the atom with the lower index). The communication overheads associated with a half-shell implementations is lower since the data required does not need to be symmetrically exchanged. In the mid-point method, the interaction between *i* and *j* is computed by the process which owns the geometric mid-point of *i* and *j*, hence the name mid-point method. This method requires the exchange of a thinner outer-shell thereby reducing the communication bandwidth requirements. Finally, in neutral territory methods, forces between atom pairs are not necessarily calculated by processes that own either atom in the pair. These methods have been shown to yield lower communication overheads by making very efficient use of data communicated between processes [27]. However, their applicability is restricted to the context of range-limited N-body simulations.

To motivate our choice of the outer shell, we illustrate in Fig. 3, the range of atoms at a neighboring process P_2 whose position information needs to be known by the process P_1 so that P_1 can compute all ReaxFF interactions that it is responsible for, based on the conventions we have adopted in Section 3.2.1. In ReaxFF, there are different cut-off distances for different types of interactions: r_{bonded} is the distance cut-off for determining bonds, r_{hbond} is the distance between the donor and acceptor atoms in a Hydrogen bond interaction and finally r_{nonb} denotes the cut-off distance for non-bonded interactions. Taking the maximum span among all interactions, we determine the minimum outer-shell width r_{shell} to be

$$r_{shell} = max(3 \times r_{bond}, r_{hbond}, r_{nonb}) \tag{1}$$

A careful inspection of Fig. 3 reveals that the nature of bonded interactions in ReaxFF does not allow the use of half-shell boundaries or NT methods. Due to over/under-coordination and valence angle interactions, even when the midpoint boundary method is used, r_{shell} does not shrink at all. Consequently, we use the full-shell scheme in spite of its higher communication cost, but our careful analysis on the minimum value of r_{shell} helps us to minimize the cost of communication there.

3.2.3. Inter-process communication

With the choice of 3D domain decomposition technique and full-shell scheme for the outer-shells of processes, inter-process communication can be performed using either direct messaging or staged messaging schemes (see Fig. 4). In direct messaging, every process prepares a separate message for each of its neighbors containing the required data and sends these messages using point-to-point communication primitives. The upside of this scheme is that communication and computation can be overlapped, i.e., after preparing and sending a message using a non-blocking send operation, a process can immediately start preparing the message for its next neighbor without having to wait for the completion of the send operation. The downside is the number of messages that need to be sent by each process. Even when we restrict the sub-domain dimensions to be greater than r_{shell}, each process needs to talk to 26 other processes (3^3 minus the self box) and the communication pattern does not truly follow the 3D torus topology that we adopt.

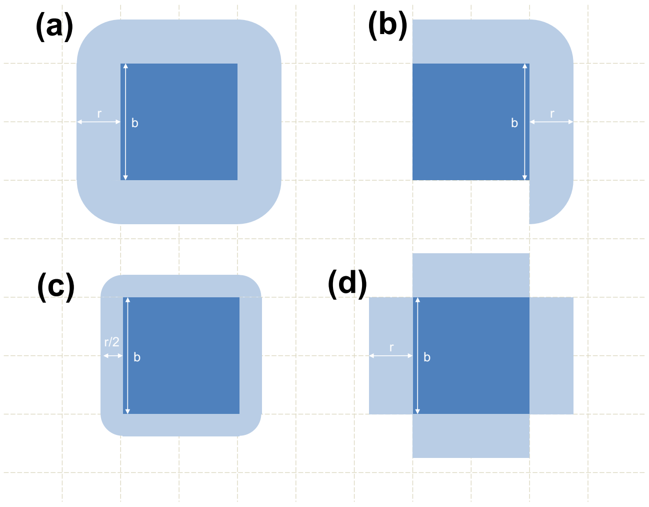


Fig. 2. Different outer-shell types commonly used in MD codes shown in 2D for purposes of simplicity: (a) full-shell, (b) half-shell, (c) midpoint-shell, (d) an example among various possible NT shells with similar characteristics: tower-plate shell. In all cases we assume the process sub-domain to be an orthogonal cube whose sides have length *b*. *r* denotes the width of the outer-shell.

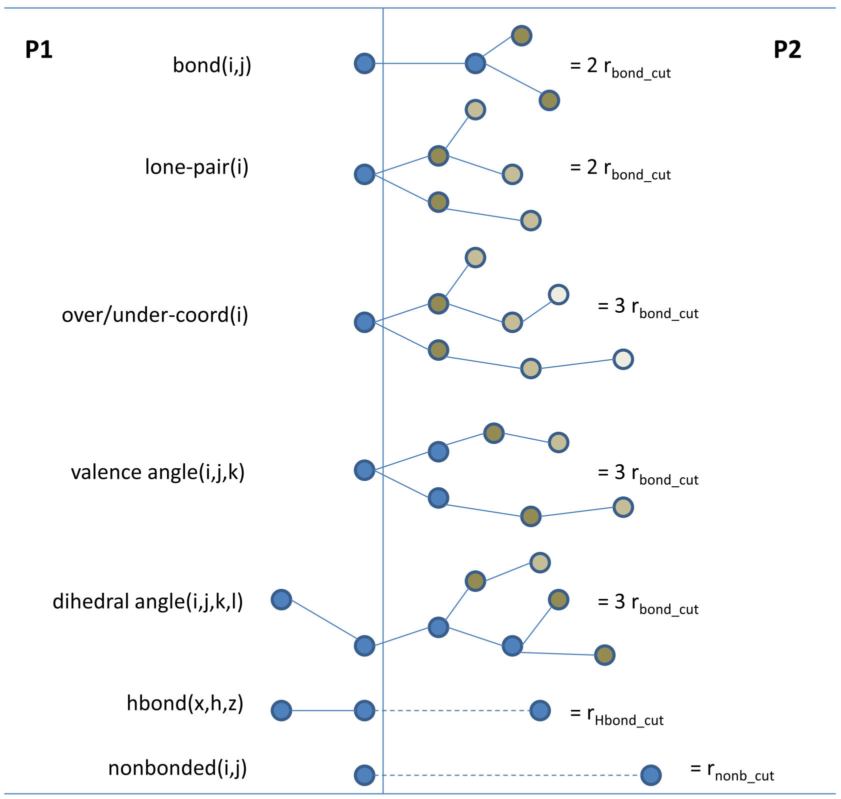
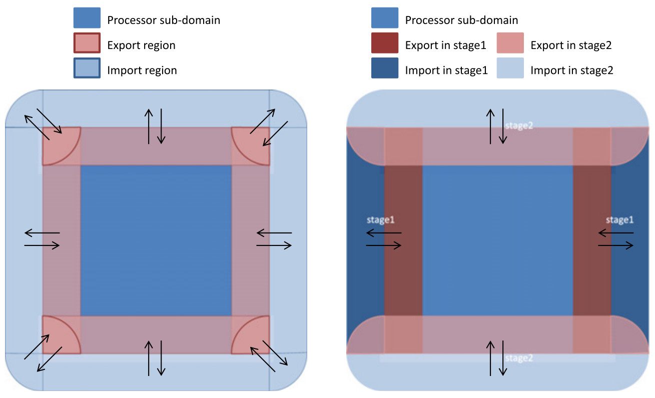


Fig. 3. Handling of each interaction in ReaxFF when it spans multiple processes. Blue colored circles represent atoms that directly participate in the interaction. Gray colored circles represent atoms that directly or indirectly affect the interaction's potential and therefore experience some force due to it. We show only such atoms in the neighboring process for clarity. Lighter tones imply weaker interaction. Next to each interaction, we note its maximum span in terms of the cut-off distances in ReaxFF. (For interpretation of the references in color in this figure legend, the reader is referred to the web version of this article.)



(a) Direct messaging in 2D (b) Staged messaging in 2D

Fig. 4. Direct messaging vs. staged messaging shown in 2D for simplicity.

In the staged messaging scheme, every process sends/receives messages along a single dimension in each stage. In a three-stage communication scheme, for example, each process sends/receives atoms in $-x$, $+x$ dimensions first, then in $-y$, $+y$ dimensions and finally in $-z$, $+z$ dimensions; at each stage augmenting its subsequent messages with the data it receives in previous stages. The upside of this scheme is that each process needs to communicate with fewer number of processes, e.g., in a 3D torus staged messaging would require only 6 messages per processor, compared to the 26 messages per processor of the direct messaging scheme). Moreover, the communication pattern of staged messaging respects the 3D torus topology, which assumes direct connections between nodes in $-x$, $+x$, $-y$, $+y$, $-z$ and $+z$ directions only. The potential

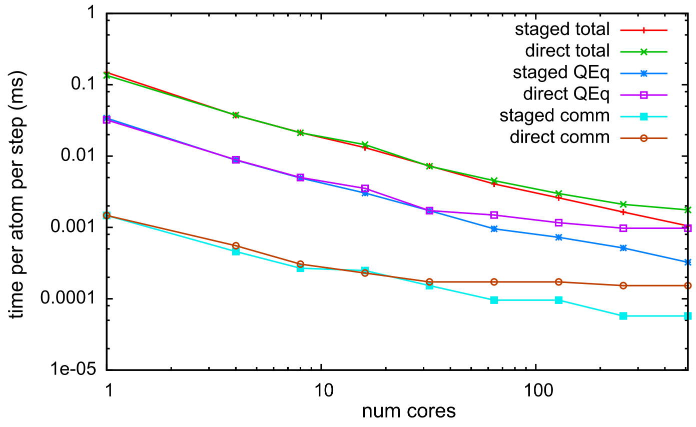


Fig. 5. Comparison of PuReMD performances using staged vs. direct messaging schemes on 2180 water molecules (6540 atoms) in a $40 \times 40 \times 40$ Å^3 orthogonal box. Total time per atom per step performance is presented together with the performances of communication bound parts in each case: *QEq* and *comm* (explained in Section 5).

drawback of the staged messaging scheme is that it requires a stop-process-forward mechanism and does not allow overlapping communication and computation. For example, after a process receives its messages in $-x$, $+x$ directions, it needs to sort the incoming data to determine what needs to be forwarded in $-y$, $+y$, $-z$, $+z$ directions.

Since both schemes have advantages and disadvantages that are hard to quantify in a platform-independent manner, we implement both schemes and perform strong scaling tests. We present the results of strong scaling tests with 2180 water molecules (6540 atoms) in a $40 \times 40 \times 40$ Å^3 orthogonal box in Fig. 5. Performance of both schemes is almost identical up to 64 cores. Beyond this, the staged messaging scheme clearly outperforms direct messaging. At 512 cores, while direct messaging shows signs of hitting the scalability barrier, staged communication scheme's curve suggests that it can scale nicely to larger number of cores. Based on these experiments, PuReMD uses a staged messaging scheme for all communications between neighbor processes. In fact, other MD packages such as Desmond [25] and LAMMPS [12] use similar messaging schemes as well.

4. Algorithmic and numerical techniques

PuReMD features several algorithmic and numerical techniques to achieve excellent per-time-step execution time. Its fully dynamic and adaptive interaction lists further improve performance and enable the simulation of large systems on platforms with limited resources. In this section, we provide a summary of the algorithms and techniques used; for a more comprehensive description of these techniques, we refer readers to [14].

4.1. Generation of neighbors lists

As in most MD codes, we use the method of binning for efficiently generating neighbor lists for atoms. This requires construction of a 3D grid within each process' sub-domain. Based on their spatial coordinates, each atom is mapped into its corresponding cell. In order to discover neighbors of an atom, it is sufficient to search within the neighboring/nearby cells (provided bin-sizes are suitably selected). Empirically, we determine that setting the dimensions of grid cells to half of the neighbor cut-off distance r_{nbrs} yields best performance. Furthermore, reordering atoms so that atoms mapped to the same grid cell are clustered together in the atom list improves the performance of neighbor generation due to cache effects. This reordering also has significant impact on the performance of force computation routines and matrix–vector multiplications in charge equilibration. Reordering atoms also allows us to cut the number of look-ups in neighboring cells by half, on average. This is because it is enough for each atom to search for its neighbors inside only neighboring cells that contain atoms with higher indices in the reordered atom list. We further reduce the number of neighbor cell look-ups by first checking the distance between the atom and the closest point of the neighboring cell to that atom. All these optimizations give us a very efficient neighbor generation procedure.

4.2. Eliminating bond order derivative lists

All bonded potentials (including the hydrogen bond potential) depend on the bond order (or bond strength) between atoms. Bond order concept, which constitutes the heart of the dynamic bonding scheme in ReaxFF, depends on the type of the two atoms forming the bond, the distance between them and the presence of other atoms within the bond cut-off distance r_{bond}. All forces arising from bonded interactions depend on the derivative of the bond order terms [6].

Let BO_{ij} denote the bond order between atoms i and j, and dBO_{ij}/dr_k denote the derivative of this bond order term with respect to the position of atom k. The strength of i–j bond is affected by the presence of other atoms around atoms i and j. Therefore, the expression dBO_{ij}/dr_k would evaluate to a non-zero value for all atoms k that share a bond with either atom i or j. The number of such atoms can run up to 20 to 25, or higher in most systems. Considering the fact that a single bond takes part in various bonded interactions, we may need to evaluate the expression dBO_{ij}/dr_k several times over a single time step. An obvious approach to efficiently computing forces from bond order derivatives is to evaluate the bond order derivative expressions at the start of each time-step and to use repeatedly, as necessary. Besides the large amount of memory required to store the bond order derivative list, this approach also has implications for costly memory lookups during the time-critical force computation routines.

We eliminate the need for storing the bond order derivatives and frequent memory lookups by delaying the computation of the derivative of bond orders until the end of each time-step. During the computation of bonded potentials, we accumulate the coefficients for the corresponding bond order derivative terms arising from various interactions into the scalar variable $CdBO_{ij}$. At the end of each time-step, we evaluate the expression dBO_{ij}/dr_k and add the force $CdBO_{ij} \times \frac{dBO_{ij}}{dr_k}$ to the net force on atom k directly.

$$\left(c_1 \times \frac{dBO_{ij}}{dr_k} + c_2 \times \frac{dBO_{ij}}{dr_k} + \cdots + c_n \times \frac{dBO_{ij}}{dr_k}\right) = (c_1 + c_2 + \cdots + c_n) \times \frac{dBO_{ij}}{dr_k} = CdBO_{ij} \times \frac{dBO_{ij}}{dr_k} \tag{2}$$

Eq. (2) illustrates the idea explained above, which is essentially making use of the distributive property of multiplication over addition. This simple technique enables us to work with much larger systems on a single processor by reducing the memory requirements of bonded interaction lists significantly. Furthermore, it improves the running time of bonded force computations by reducing the number of flops and also lookups to memory.

4.3. Truncating bond related computations at the outer-shell

As mentioned before, bonded interactions in ReaxFF are expensive. To correctly compute bonded interactions at the boundaries, we need to compute the bonds in the outer-shell as well. If this is not done appropriately, scalability of bond related computations can be severely constrained. Since we choose the outer-shell to be a full-shell, the ratio of the outer-shell volume to the subdomain volume can be as high as 20 when we take $b = r$ in Fig. 2, where b denotes the length of a side of the process sub-domain which we assume to be an orthogonal cube and r corresponds to the outer-shell width. Therefore depending on their proximity to process boundaries, some bonds might need to be computed several times in the extreme case of $b = r$.

A close examination of Fig. 3 reveals that for each atom, we need to know bonds that are only three hops into the outer-shell. Consequently, in PuReMD, we restrict the computation of bonds inside the outer-shell to those that are at-most three hops away from the subdomain of a process. As we show in Section 5, we obtain excellent scaling for bond related computations in PuReMD.

4.4. Lookup tables for non-bonded interactions

In general, computing non-bonded forces is more expensive than computing bonded forces, due to the larger number of interactions within the (larger) cut-off radius, r_{nonb}, associated with non-bonded interactions. However, the simple form of non-bonded interactions (pairwise interactions) allows the use of a lookup table and approximation of complex expressions by means of interpolation. This is a common optimization technique used by many MD codes that yields significant performance improvements with relatively little impact on accuracy. In PuReMD, we make use of this technique through cubic spline interpolations. All test results presented in Section 5 utilize this optimization.

4.5. Charge equilibration

Charge equilibration corresponds to the problem of assigning partial charges to atoms with a view to minimizing electrostatic energy under constraints of charge neutrality. In the absence of electronic degrees of freedom, we do this using the QEq method developed by Rappe and Goddard [17]. We follow the mathematical formulation of Nakano [18] for our QEq solver. Using the method of Lagrange multipliers to solve the minimization problem described in detail in [17], we obtain two sparse linear systems:

$$-\chi_k = \sum_i H_{ik} s_i \tag{3}$$

$$-1 = \sum_i H_{ik} t_i. \tag{4}$$

H is an $N \times N$ coefficient matrix, N being the number of atoms in the system. Entries of H are computed based on the distances and atom types of non-bonded pairs in the system. χ is a vector of force field parameters specified as input to the simulation, s and t are pseudo-charge vectors which we seek the solutions for. Partial charge on an atom, denoted by q_i, is computed using the corresponding pseudo-charge values:

$$q_i = s_i - \frac{\sum_i s_i}{\sum_i t_i} t_i \tag{5}$$

The high computational cost of direct solvers for large systems (10^7 degrees of freedom and beyond) renders them unsuitable for our application. We rely on well-known Krylov subspace methods – our sequential implementation [14] relies on an ILUT (incomplete LU factorization with thresholds) preconditioned GMRES method [20,21], and our parallel implementation described here relies on the Preconditioned Conjugate Gradients (PCG) method [19]. The coefficient matrix H carries a heavy diagonal, therefore we use diagonal scaling as preconditioner to the CG algorithm which works nicely as a cheap and effective preconditioner in a parallel environment. All results reported in Section 5 use the diagonally scaled parallel CG solver for charge equilibration.

It is important to solve the QEq problem to high accuracy (low residual), otherwise the energy of the system shows unacceptable drifts during constant energy (NVE) simulations. A relative residual norm of 10^{-6} generally provides satisfactory results. However, even at this tolerance level, the charge equilibration part requires significant computation and communication time as discussed in Section 5. Therefore it is important to improve the performance of the QEq solver in order to achieve good performance and scalability results. Below, we describe simple, yet effective techniques used in PuReMD.

Make a good initial guess: An important observation is that in ReaxFF time-steps are on the order of tenths of femtoseconds. Therefore, positions of atoms change very slightly between successive time-steps. This observation implies that solutions to Eqs. (3) and (4) in prior time-step(s) yield good initial guesses regarding solutions at the current time-step. Indeed, by making linear or quadratic extrapolations on the solutions, better initial guesses can be obtained for the QEq problem.

In Table 1, we present the effect of different extrapolation schemes on the number of iterations required to solve Eqs. (3) and (4) for a bulk water system. As can be seen, convergence characteristics of the two systems are different from each other. While we can capture the evolution of the solution to Eq. (4) best with a quadratic extrapolation scheme, the evolution of Eq. (3)'s solution looks more like a cubic curve. Consequently, we obtain a simple but effective solver for the charge equilibration problem, namely a diagonally scaled parallel PCG solver that relies on cubic extrapolations for solving Eq. (3) and quadratic extrapolations for Eq. (4). We would like to note that for different systems and simulation settings, depending on the characteristics of the system, other combinations of extrapolation schemes might be more suitable.

Iterating Jointly: The PCG algorithm [19] includes one matrix–vector product and two dot products as its major parts, in each iteration. In a sequential context, matrix–vector products dominate the QEq solve time. However, in a parallel context, a significant portion of the QEq solve time is spent in communications: two local communications (one staged messaging step for sharing the updated vector contents with neighboring processes and another one for tallying the partial results from matrix–vector multiplication) and two global communications (two all-reduce operations for dot products). As mentioned in Section 3.2.1, we avoid the reverse communication at the expense of some redundant computations. We further reduce the total number of communication steps by iterating both systems and communicating their data together until one of them converges (typically solution to Eq. (4) converges first) and after that point we iterate the remaining system by itself. For example, in a typical scenario, the QEq solve takes $11 + 6 = 17$ iterations (and matrix–vector multiplications) and $17 \times 3 = 51$ communication steps if both systems described in Table 1 are solved separately. By iterating them together, we still have to perform 17 matrix–vector multiplications but now we need much fewer communication operations, $max(11,6) \times 3 = 33$ to be precise.

4.6. Data-structures and reallocation

In a reactive force field, the dynamic nature of bonds, three-body and four-body interactions together with the significant amount of book-keeping required for these interactions require large amounts of memory and sophisticated procedures for managing allocated memory. With suitable choices for data structures for various lists maintained by ReaxFF, we can minimize the memory footprint of PuReMD, while still providing efficient access to all lists for force computations.

We store neighbor lists and the QEq matrix in compressed sparse row (CSR) format. Both of these lists are half lists, i.e., we store only the upper half of the matrix in each case. The manner in which these lists are generated and accessed is well-suited to the CSR format. Bond lists are stored as full lists in *modified* CSR format which is a full list because higher order

Table 1
Average number of iterations required by the diagonally scaled PCG solver using different initial guesses during the simulation of a bulk water system under the *NVE* ensemble. PCG tolerance is set to 10^{-6}. Extrapolation schemes yielding the best performance for each linear system are indicated in bold.

Initial guess for step t	Eq. (3)	Eq. (4)
Initial guess = 0	42	33
Solution from step $t-1$	27	16
Linear extrapolation	19	11
Quadratic extrapolation	15	**6**
Cubic extrapolation	**11**	9

bonded interactions (three-body and four-body) are derived from the bond list. We call the format of our bond lists *modified* CSR format, because the space reserved for each atom on the list is contiguous, but the actual data stored is not. Before allocating the bond lists, we estimate the number of bonds for each atom. Let eb_i denote the number of estimated bonds for atom i. We allocate $max(2eb_i, MIN\_BONDS)$ slots to atom i in the bond list. This conservative allocation scheme prevents any overwrites in subsequent steps, while reducing the frequency of bond list reallocations through the simulation as the system evolves and bonding patterns change.

Three-body interaction list is built from the bond list and stored in CSR format indexed not by the individual atoms but by the bonds in the bond list. Four-body structures are constructed from the three-body interactions. Four-body structures are not stored, since there are no higher order interactions in ReaxFF. Energy and forces due to the discovered four-body interactions are computed on the fly.

We maintain a dedicated hydrogen bond list, because in ReaxFF, hydrogens are often bonded to more than one atom, and the neighbors of hydrogen atoms are spread throughout the entire neighbors list (recall that the neighbors list is a half list). The hydrogen bond list uses the same *modified* CSR format described above.

In order to minimize the memory footprint of PuReMD, we adopt a three stage memory management scheme: estimation, monitoring, and reallocation. At the start of the simulation, the sizes of each list are estimated just by counting (but not storing) the neighbors and bonds of each atom – this is a one-time, inexpensive calculation. During each step of the simulation, the utilization of lists are monitored carefully. If the utilization of a list reaches a prescribed threshold, that list is reallocated by conservative estimations of its size based on the current utilization. To avoid significant overheads with reallocations (such as copying of stored data), we ensure that the reallocation daemon is invoked only at specific instances.

5. Performance characterization

In this section, we present a comprehensive analysis of the performance of PuReMD. We examine its performance from two perspectives: *weak scaling*, where we increase the system size (number of atoms and the simulation domain volume) linearly with the number of cores, and *strong scaling*, where we measure the scalability while increasing the number of cores used for a given physical system (fixed number of atoms and fixed simulation domain volume). For all tests, we use the Hera cluster at the LLNL-OCF. Hera is comprised of 800 batch nodes, each with four AMD Opteron quad-core processors clocked at 2.3 GHz (for a total of 12800 cores, 127.2 TFLOP/s) and 32 GB memory (Fig. 6). Nodes are connected through an InfiniBand interconnect. MVAPICH2 implementation is used for message passing.

We perform all our tests using a bulk water system under the micro-canonical (*NVE*) ensemble. The primary reason we choose a bulk water system for our performance experiments is that the ReaxFF model for water includes almost all interactions present in the ReaxFF formulation. Furthermore, water is almost ubiquitous in MD simulations and it has been the focus of many scientific studies.

To better understand the results of our experiments, we identify six key parts of PuReMD:

- **comm:** initial communications step with neighboring processes for atom migration and boundary atom information exchange.

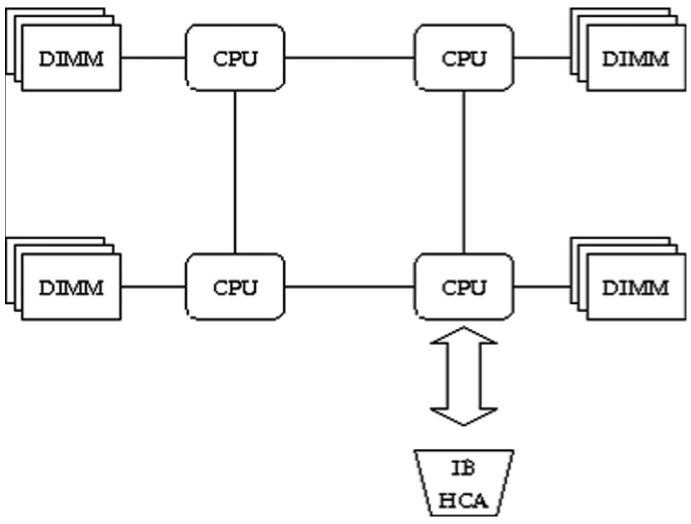


Fig. 6. Architecture of a node in the Hera cluster [28]. Each processing unit is an AMD Opteron quad-core having 8 GB of dedicated memory. Total memory per node is 32 GB. Only one InfiniBand interface is available per node. Processors not directly connected to the InfiniBand interface transfer their data through the HyperTransport links first.

- **nbrs:** neighbor generation step, where all atom pairs falling within the neighbor cut-off distance r_{nbrs} are identified.
- **init_forces:** generation of the charge equilibration (QEq) matrix, bond list, and H-bond list based on the neighbors list.
- **QEq:** is the charge equilibration part that solves a large sparse linear system using the PCG algorithm with a diagonal preconditioner. This involves costly sparse matrix–vector multiplications (SpMV) and both local and global communications.
- **bonded:** is the part that includes computation of forces due to all interactions involving bonds (hydrogen bond interactions are included here as well). This part also includes identification of 3-body and 4-body structures in the system.
- **nonb:** is the part that computes forces due to non-bonded interactions (van der Waals and Coulomb).

Each of these parts has different characteristics: some are compute-bound, some are memory-bound while others are inter-process communication-bound. Together they comprise almost 99% of the total computation time for typical systems. We perform detailed analyses of these major components to better understand how PuReMD responds to increasing system sizes and increasing number of cores. We also use these results to infer the impact of various machine parameters on performance.

5.1. Weak scaling results

Bulk water system that we used for weak scaling tests consists of 2180 water molecules (6540 atoms) inside a $40 \times 40 \times 40$ Å^3 orthogonal box. This setup yields a water system of ideal density at room temperature.

Fig. 7 shows variation in simulation time per time-step as a function of number of cores used in weak scaling experiments. The increase in CPU time from 1 to 16 cores is primarily due to the *init_forces*, *QEq* and *nonb* parts. In a single process run, we use only one core out of the 16 cores available on a node; in a 4 processes run, we use only one core on each processor; only when we go to 16 processes do we utilize a node fully. Construction of the QEq coefficient matrix and interaction lists during *init_forces*, SpMV's during *QEq*, and computation of non-bonded forces in *nonb* put considerable stress on the memory system. When a processor is fully utilized (all four cores), this represents the major performance bottleneck – consequently the degradation in performance. To be able to correctly measure the weak scaling characteristics of PuReMD, we use the 16 core runs, where a single node is fully utilized, as our base case in Table 2.

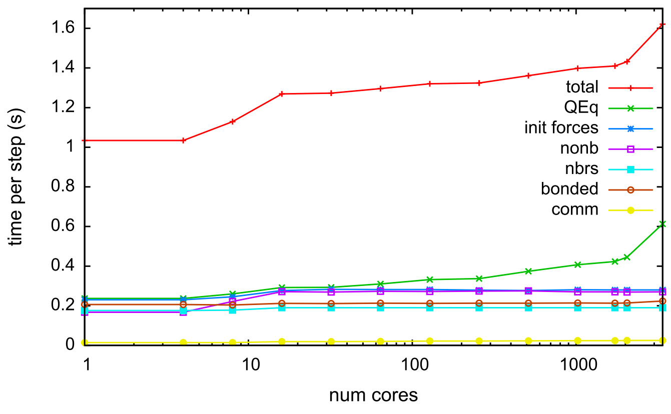


Fig. 7. Weak scaling: total time per step as a function of increasing number of cores and system size. At 3375 cores, size of the simulated system is approximately 22 M atoms. Per-step time for the six major PuReMD components are also shown.

Table 2
Scalability of PuReMD for the water system demonstrating over 78% efficiency at 3375 cores under weak scaling.

#Cores	$\frac{QEq}{Total}$ (%)	Eff. (%)
16	23	100
32	23	99
64	23	97
128	25	96
256	25	95
512	27	93
1024	29	90
1728	30	90
2048	31	88
3375	37	78

As we move beyond 16 cores, we observe that all parts except for *QEq* and *comm* (which are communication-bound) scale nearly ideally. The increase in *comm* time is negligible compared to that of *QEq*. As we mentioned before, in a typical simulation, the QEq solver takes 10–15 iterations and therefore 30–45 communication operations in total per time-step, on average. While two thirds of these communication operations are all-reduce operations, as is the case with a dot product, the remaining one third is staged messaging operations related to the SpMV's. This significant communication requirement results in some performance degradation of *QEq* beyond 16 cores. Note however that the optimizations we have described in Section 4.5 are critical and that QEq performance degradation does not significantly impact the overall efficiency of our code (78% efficiency at 3375 processing cores) (see Fig. 8).

5.2. Strong scaling results

For strong scaling tests, we perform our experiments on the same water system described in Section 5.1. To be able to work with large number of cores, we have replicated that water system twice in each dimension, yielding a system of 52320 atoms inside an $80 \times 80 \times 80$ Å^3 orthogonal simulation box.

Fig. 9 and Table 3 present results of our strong scaling tests. Parts of PuReMD that do not require significant communication or redundant computations, *i.e.*, *bonded* and *nonb* parts, scale well with the increasing number of cores. In *nbrs* part, besides generating the neighbors of local atoms, we need to generate the neighbors of outer-shell atoms within the r_{bond} cutoff as well. This is required for bond related computations in the *bonded* part, and suggests a significant increase in the overall ratio of redundant *nbrs* computations to the total *nbrs* computations as the process sub-domains become smaller. Consequently, we observe a slightly worse scaling behavior from *nbrs* compared to *bonded* and *nonb* parts.

The *init_forces* part is also affected by the same redundant computations within the outer-shell because it is responsible for the computation of uncorrected bond orders for the outer-shell atoms. However, these redundant computations do not propagate to the *bonded* part because we truncate most redundant bond related computations at the outer-shell after a pro-

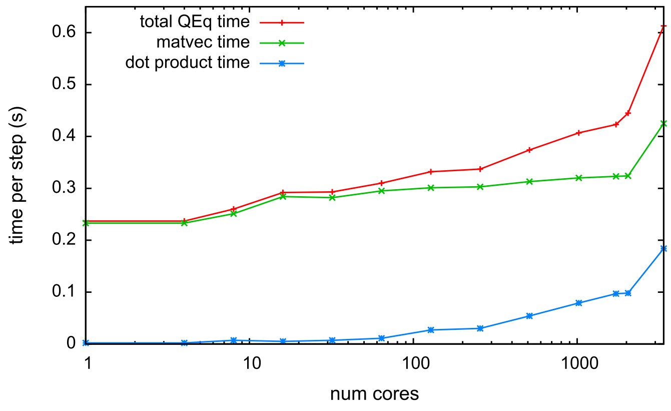


Fig. 8. QEq solver requires 11 PCG iterations and 17 matrix–vector multiplications per time-step on average. The total *QEq* time is dominated by the sparse matrix vector multiplication (SpMV) time for small number of cores. However, *QEq* scaling is significantly impacted by the time required for global reductions (dot-products) as the number of cores increases.

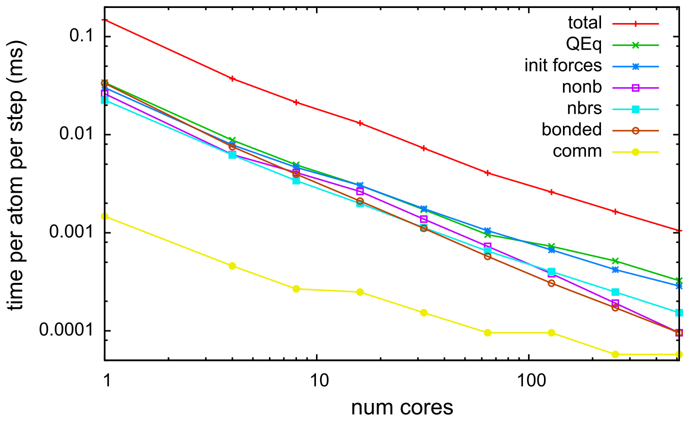


Fig. 9. Strong scaling: Total time per step as a function of increasing number of cores for a fixed system size. Per-step timings for the six major PuReMD parts are also shown.

Table 3
Strong scaling test results for 52320 atom bulk water system.

#Cores	$\frac{Atoms}{Core}$	Time per step	$\frac{QEq}{Total}$ (%)	Eff. (%)	Throughput $\left(\frac{ns}{day}\right)$
1	52320	7.776	22.9	100	0.003
4	13080	1.952	23.6	99	0.011
8	6540	1.119	23.1	86	0.019
16	3270	0.687	23.1	70	0.031
32	1635	0.381	23.6	63	0.057
64	817	0.213	23.5	57	0.101
128	408	0.136	27.9	44	0.159
256	204	0.086	31.4	35	0.251
512	102	0.055	30.9	27	0.393

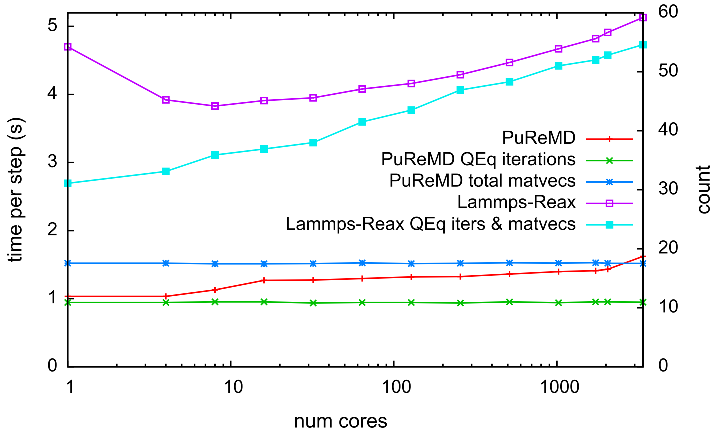


Fig. 10. Comparison of PuReMD and LAMMPS codes under weak scaling. The left *y*-axis shows the per time-step running times for both codes. The right *y*-axis is the average matrix–vector multiplication and linear solver iteration counts per step. Since LAMMPS always does a single matrix–vector multiplication per iteration, we show its iteration and matrix–vector multiplication counts in a single curve.

cess computes all uncorrected bond orders that it is responsible for, as was explained in Section 4.3. However, the *init_forces* part scales worse than the *nbrs* part. This is because it is susceptible to another form of redundancy – double computations during the construction of the *H* matrix in order to avoid the reverse communication step in *QEq*. In fact, these redundancies make *init_forces* the most expensive part together with *QEq*, as we scale to large number of processes.

As is the case with most parallel applications, communication bound parts (*comm* and *QEq*) do not scale well. Poor scalability of *comm* is not a major concern, since ReaxFF is an expensive force field. In Fig. 9, it can be seen that *comm* takes a very small share of the total time per time-step; even with 512 cores only about 5% of the total execution time. On the other hand, *QEq* is one of the most expensive parts in PuReMD. NT methods introduced by Shaw et al. [27] have been shown to reduce the communication bandwidth requirements of range limited N-body simulations significantly. ReaxFF is a range-limited force field as well. While the dynamic nature of bonds and bonded interactions prevent us from directly adopting NT methods for all communications, computation and communication patterns in *QEq* are suited to adopting NT methods exclusively for the *QEq* part. We intend to include this optimization in a future release of PuReMD.

A more effective way of improving the scalability of *QEq* would be to reduce the iteration count of its linear solver through an effective preconditioner. One commonly used preconditioning technique is based on the incomplete LU factorization of the coefficient matrix [21]. While ILU-based preconditioners are effective in reducing the iteration counts of iterative solvers, their major drawback is the high computational cost associated with the factorization stage. In our sequential version [14], we amortize the cost of ILUT factorization by reusing the preconditioner over several time-steps. In a serial context, we have observed that the performance of ILUT-based schemes are far superior to the diagonally scaled schemes described in Section 4.5. These results suggest that using ILUT-based preconditioners in *QEq*, it is possible improve the performance and scalability of PuReMD significantly. Realizing these improvements, though, poses significant challenges. Scalability limitations of ILUT preconditioners in a parallel context prevent their use on large machine configurations. Our initial tests using the P-SPIKE algorithm [29] show promising results, both in terms of solve time and scalability of the *QEq* part.

5.3. Comparison with LAMMPS

In this section, we present the performance comparison of our code with the only other publicly available parallel ReaxFF implementation, ReaxFF package in LAMMPS. We repeat the same weak scaling and strong scaling tests described above

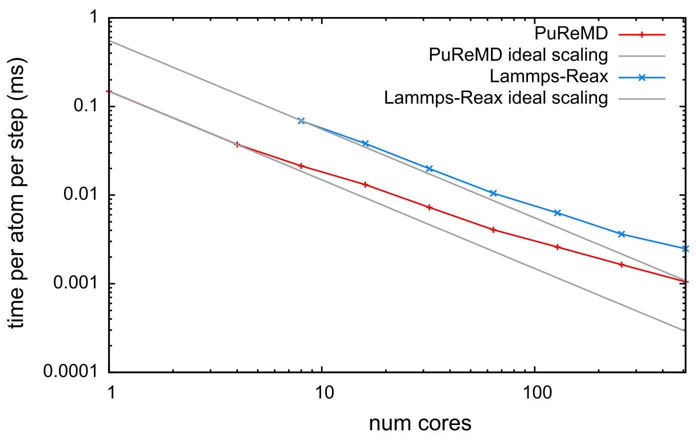


Fig. 11. Comparison of PuReMD and LAMMPS codes under strong scaling. Gray lines indicate the ideal scaling curves for both codes.

using the LAMMPS code and provide comparisons. Both codes have been compiled using the same compilers (Intel C/C++ compilers), compiler flags (-O3 -funroll-loops -fstrict-aliasing), and MPI library (MVAPICH2) on the Hera cluster. Neither code has been tuned to the specific architecture of the machine. It is possible that the performance of these codes may be further improved through platform-specific optimizations.

Fig. 10 shows the comparison of both codes under the weak scaling test. On a single core, PuReMD is about five times faster than LAMMPS. However, LAMMPS code shows a surprising drop in the per time-step running time while going from 1 to 4 and then 8 cores. Despite the drop, PuReMD is still about four times faster than LAMMPS. Taking the 16 cores runtime as basis, as we did in Section 5.1, LAMMPS achieves a weak scaling efficiency which is very close to that of PuReMD (76% vs. 78%, respectively). The main reason behind the increase in LAMMPS's time per time-step is the increasing number of iterations required by its charge equilibration solver as the system size increases. By using a different mathematical formulation for solving the charge equilibration problem and applying key optimizations described in Section 4.5, we are able to maintain constant matrix–vector multiplication and PCG iteration counts at a much lower level.

Fig. 11 shows the results of our strong scaling comparisons. We could not run the 52320 bulk water simulation with LAMMPS code using fewer than 8 cores due to memory limitations. At 8 cores, PuReMD is about three times faster than LAMMPS, and we are able to maintain this ratio all the way through 512 cores.

Finally, we would like to note that PuReMD has been designed and developed to be modular and extensible, like LAMMPS. Consequently, it is quite easy to make improvements and modifications on it. Modifying the PuReMD code to work with other bond-order potentials would primarily involve modifying the force computation routines.

5.4. Memory usage

Another important aspect of PuReMD is its small memory footprint and its ability to adapt to the memory needs of the system to be simulated. We were unable to measure the precise memory footprint of PuReMD on the Hera cluster. However, our tests have shown that PuReMD is able to simulate a 296,960 atom PETN system on a single processor using an estimated memory of 15 GB. To the best of our knowledge, PuReMD is the only ReaxFF implementation that can simulate such large systems with great ease – *i.e.*, without requiring any tuning of compilation and runtime parameters.

It is important to note that in spite of the critical performance analysis presented in this section, PuReMD achieves high parallel efficiency under typical weak-scaling workloads (over 78% efficiency using 3375 processing codes) with a small memory footprint. It does so at excellent per-step per-particle simulation time (which renders achieving this high parallel efficiency more difficult) and yields perfect agreement with the ReaxFF model potentials. To this end, PuReMD provides a unique simulation capability.

6. Concluding remarks

In this paper, we have presented an efficient and scalable parallel implementation for ReaxFF in C using MPI. Our open-source implementation is shown to be (i) 3–5 times faster compared to other implementations, (ii) has a significantly smaller memory footprint, and (iii) has been demonstrated to scale to more than three thousand computational cores under weak-scaling scenarios, yielding over 78% efficiency. Its modular and extensible design makes further improvements and enhancements very easy.

PuReMD's accuracy has been verified against the original ReaxFF code by comparing the energy and forces due to every single interaction in the Reax formulation under many diverse simulation scenarios. We have compared the net forces on individual atoms and verified that any differences are within expected numerical deviations. In addition to the systems used in this paper, PuReMD has been used by other research groups to study such diverse systems as strain relaxation in Si–Ge nanobars, water–silica systems under pressure, and Ti-silica systems under impact stress.

Acknowledgments

The authors thank Adri van Duin for significant help through all aspects of software development, Aidan Thompson and Steve Plimpton for various discussions on parallel ReaxFF implementation and their comments on an initial version of this paper. The authors would like to acknowledge grants from the Department of Energy and the National Science Foundation, and computational resources at the LLNL-OCF.

References

[1] T.A. Halgren, W. Damm, Polarizable force fields, Current Opinion Struct. Biol. 11 (2001) 236–242.
[2] J.E. Davis, G.L. Warren, S. Patel, Revised charge equilibration potential for liquid alkanes, J. Phys. Chem. B 112 (2008) 8298–8310.
[3] D.W. Brenner, Empirical potential for hydrocarbons for use in simulating the chemical vapor deposition of diamond films, Phys. Rev. B 42 (1990) 9458–9471.
[4] D.W. Brenner, O.A. Shenderova, J.A. Harrison, S.J. Stuart, S.B. Sinnott, A second-generation reactive empirical bond order (REBO) potential energy expression for hydrocarbons, J. Phys. Condens. Matter. 14 (2002) 783–802.
[5] S.J. Stuart, A.B. Tutein, J.A. Harrison, A reactive potential for hydrocarbons with intermolecular interactions, J. Chem. Phys. 112 (2000) 6472.
[6] A.C.T. van Duin, S. Dasgupta, F. Lorant, W.A. Goddard III, ReaxFF: A reactive force field for hydrocarbons, J. Phys. Chem. A 105 (2001) 9396–9409.
[7] K.D. Nielson, A.C.T. van Duin, J. Oxgaard, W-Q. Deng, W.A. Goddard III, Development of the ReaxFF reactive force field for describing transition metal catalyzed reactions, with application to the initial stages of the catalytic formation of carbon nanotubes, J. Phys. Chem. A 109 (2005) 493–499.
[8] K. Chenoweth, S. Cheung, A.C.T. van Duin, W.A. Goddard III, E.M. Kober, Simulations on the thermal decomposition of a poly(dimethylsiloxane) polymer using the ReaxFF reactive force field, J. Am. Chem. Soc. 127 (2005) 7192–7202.
[9] M.J. Buehler, Hierarchical chemo-nanomechanics of proteins: Entropic elasticity, protein unfolding and molecular fracture, J. Mech. Material Struct. 2 (6) (2007) 1019–1057.
[10] A. Thompson, H. Cho. (2010, Apr.) LAMMPS/ReaxFF potential. [Online]. Available: <http://lammps.sandia.gov/doc/pair_reax.html>.
[11] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, second ed., SIAM, 1994.
[12] S.J. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1995) 1–19.
[13] A. Nakano, R.K. Kalia, K. Nomura, A. Sharma, P. Vashishta, F. Shimojo, A.C.T. van Duin, W.A. Goddard, R. Biswas, D. Srivastava, L.H. Yang, De novo ultrascale atomistic simulations on high-end, Int. J. High Perf. Comput. Apps. 22 (1) (2008) 113–128.
[14] H.M. Aktulga, S. Pandit, A.C.T. van Duin, A. Grama, Reactive molecular dynamics: numerical methods and algorithmic techniques, SIAM J. Sci. Comput. (2011).
[15] Y. Park, H.M. Aktulga, A.Y. Grama, A. Strachan, Strain relaxation in Si/Ge/Si nanoscale bars from MD simulations, J. Appl. Phys. 106 (2009) 034304.
[16] J.C. Fogarty, H.M. Aktulga, A.C.T. van Duin, A.Y. Grama, S.A. Pandit, A reactive simulation of the silica-water interface, J. Chem. Phys. 132 (2010) 174704.
[17] A.K. Rappe, W.A. Goddard III, Charge equilibration for molecular dynamics simulations, J. Phys. Chem. 95 (1991) 3358–3363.
[18] Aiichiro Nakano, Parallel multilevel preconditioned conjugate-gradient approach to variable-charge molecular dynamics, Comput. Phys. Commun. 104 (1997) 59–69.
[19] J.R. Shewchuk, An introduction to the conjugate gradient method without the agonizing pain, School of Computer Science, Carnegie Mellon University, Tech Report, August, 1994.
[20] Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual method for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869.
[21] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, 2003.
[22] B.G. Fitch, R.S. Germain, M. Mendell, J. Pitera, M. Pitman, A. Rayshubskiy, Y. Sham, F. Suits, W. Swope, T.J C. Ward, Y. Zhestkov, R. Zhou, Blue Matter, an application framework for molecular simulation on Blue Gene, J. Parallel Distributed Comput. 63 (2003) 759–773.
[23] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kal, K. Schulten, Scalable molecular dynamics with NAMD, J. Comput. Chem. 26 (16) (2005) 1781–1802.
[24] B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation, J. Chem. Theor. Comput. 4 (3) (2008) 435–447.
[25] K.J. Bowers, E. Chow, H. Xu, R.O. Dror, M.P. Eastwood, B.A. Gregersen, J.L. Klepeis, I. Kolossvry, M.A. Moraes, F.D. Sacerdoti, J.K. Salmon, Y. Shan, D.E. Shaw, Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters, SC06, Tampa, Florida, Nov 11–17, 2006.
[26] D.E. Shaw, M.M. Deneroff, R.O. Dror, J.S. Kuskin, R.H. Larson, J.K. Salmon, C. Young, B. Batson, K.J. Bowers, J.C. Chao, M.P. Eastwood, J. Gagliardo, J.P. Grossman, C.R. Ho, D.J. Ierardi, I. Kolossvry, J.L. Klepeis, T. Layman, C. McLeavey, M.A. Moraes, R. Mueller, E.C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, S.C. Wang, Anton: A special-purpose machine for molecular dynamics simulation, in: ISCA'07, San Diego, California, Jun 9–13, 2007.
[27] K.J. Bowers, R.O. Dror, D.E. Shaw, Zonal methods for the parallel execution of range-limited N-body simulations, J. Comput. Phys. 221 (2007) 303–329.
[28] Linux Clusters Overview, 2009.<https://computing.llnl.gov/tutorials/linux_clusters/#OpteronMemoryConsiderations>.
[29] M. Manguoglu, A. Sameh, O. Schenk, PSPIKE: a parallel hybrid sparse linear system solver, Lec. Notes Comput. Sci. 5704 (2009) 797–808.