Particle–Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations

Steve Plimpton* Roy Pollock† Mark Stevens‡

Abstract

The parallel implementation of a molecular dynamics code suitable for simulation of general molecular systems using spatial–decomposition methods is discussed. Long-range Coulombic forces are computed using a particle-mesh Ewald (PPPM) technique. A multiple–timescale integration method known as rRESPA is also used to improve the computational efficiency. The load–balancing and interprocessor communication issues that arise from implementing PPPM and rRESPA within a spatial–decomposition framework are highlighted. Some performance numbers for large–scale biomembrane simulations on a Intel Paragon are given.

1 Introduction

Molecular dynamics (MD) is a commonly used tool for simulation of the structural, thermodynamic, and transport properties of biological and polymeric systems on the picosecond to nanosecond timescale. During a timestep of the MD simulation, forces are computed on each atom due to its interaction with other atoms, and atoms move by integrating simple Newtonian equations of motion. For general molecular systems, forces are computed from a many-body potential energy functional E for the system of N atoms that is typically of the form

$$E = \sum_{\text{bonds}} K_b (r - r_0)^2 + \sum_{\text{angles}} K_\theta (\theta - \theta_0)^2 + \sum_{\text{dihedrals}} K_{\phi_p} [1 + d_p \cos(n_p \phi)]$$

*MS 1111, Sandia National Laboratories, Albuquerque, NM 87185
†MS L–50, Lawrence Livermore National Laboratories, Livermore, CA 94550
‡MS 1111, Sandia National Laboratories, Albuquerque, NM 87185

$$(1) \qquad + \sum_{i=1}^{N} \sum_{j>i}^{N} \epsilon_{ij} \left[(\frac{\sigma_{ij}}{r})^{12} - (\frac{\sigma_{ij}}{r})^{6} \right] + \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{q_i q_j}{r}$$

The first 3 terms are the *bonded* energy due to interactions within the topology of the molecules. They are written here as sums over simple harmonic 2–body (bond), 3–body (angle) and 4–body (dihedral) interactions, though other force models can also be used. The last 2 terms are the *non–bonded* energy due to interactions between all pairs of atoms in the system. The first of these is van der Waals energy; because it falls off as $1/r^6$ the sum over j is often truncated within a cutoff distance so that only a few neighbors of atom i contribute rather than all N. The last term is Coulombic energy for the collection of N atoms with individual charges q_i. For computational efficiency it is also sometimes evaluated as a truncated sum over neighbors rather than as a full $O(N^2)$ double sum. The former method introduces large inaccuracies and the latter is problematic in periodic systems; consequently, other methods are often used as we discuss below.

The parallel nature of MD simulations has long been recognized [6]. Assuming the Coulombic term in equation 1 can be computed in $O(N)$ time, then on P processors, the overall calculation should scale as N/P. For general molecular systems simulated on message–passing machines, most parallel implementations have used the *replicated–data* technique [14] where a copy of all N atomic positions is stored on each of P processors. This enables easy computation and load–balancing of the 1st 3 terms in equation 1. However at each timestep, the interprocessor communication needed to globally update a copy of the N-vector of atom positions scales as N, independent of P. Thus replicated–data methods do not scale to large numbers of processors. An alternative known as *force–decomposition* scales as $N/\sqrt{P}$, but is still sub–optimal [11]. For large N/P ratios, spatial–decomposition methods are clearly the best algorithmic choice. By subdividing the physical volume among processors, most computations become local and communication is minimized so that optimal N/P scaling can be achieved. Such methods have been extensively used for atomic simulations (see [10] and references therein), and less so for general molecular systems [2, 17]. A chief difficulty is how to compute the bonded terms in parallel. In the next section we describe how our code is structured to enable parallelization of bonded interactions with a spatial–decomposition of molecular connectivity information.

For general molecular systems, two other ideas are often used to enable quick computation of equation 1. The first is to avoid the all–pairs $O(N^2)$

Coulombic computation of the last term. Approximate techniques include multipole methods [4, 7] which scale as N, and Ewald summation [5] which scales as $N^{3/2}$. More recently particle–particle particle–mesh (PPPM) methods [8] which scale as $N\sqrt{\log(N)}$ have been applied to the MD problem in which context they are also known as particle–mesh Ewald methods [3]. We believe PPPM has some distinct advantages [12] over the other methods, particularly in a parallel setting; we describe our parallel implementation of PPPM in section 3.

Another idea that can be exploited is to recognize that not all terms in equation 1 need be computed every timestep to maintain a desired level of accuracy. Physically this means that fast modes such as bond vibrations can be decoupled from slower motions due to the non–bond interactions. A particularly rigorous multiple timescale integrator is known as rRESPA [16]. Our parallel implementation of rRESPA is described in section 4. In both sections 3 and 4 the emphasis is on the parallel issues that must be addressed to efficiently implement PPPM and rRESPA within the spatial–decomposition framework outlined in section 2. Finally, some preliminary timings for our parallel MD code known as LAMMPS (large–scale atomic/molecular massively parallel simulator) are given in section 5.

2 Spatial–Decomposition

The basic idea of a spatial–decomposition method for MD is to divide the physical geometry into small boxes, one per processor. Each processor will compute primarily on atoms within its box. In this paper we assume a regular 3–d division of the domain into equal–sized boxes. This may induce load–imbalance in problems with non–uniform densities, but that is not typical of the problems we have been simulating with LAMMPS.

We distinguish between two types of atoms, those within a processor's box that are "owned" by the processor, and those atoms that are nearby which are owned by "other" processors. For *other* atoms, only a minimal set of information necessary to compute force interactions with those atoms is stored — position, charge, atom type, and global tag (a number from 1 to N). By contrast, for *owned* atoms, all of the molecular connectivity information is also stored in addition to the usual positions, velocities, global tag, etc. This data is initialized when the molecular topology is input at the beginning of the simulation. Each bond, angle, and dihedral interaction is permanently assigned to one of the 2, 3, or 4 atoms involved. The processor owning that atom stores the global tags of all atoms in the bonded interaction as well as a pointer to the set of numerical coefficients needed to compute the interaction.

Using these data structures, a single MD step that advances the positions x and velocities v of atoms of mass m by a single timestep Δt is shown in Figure 1. It is written in a velocity Verlet formulation since we will build on this structure for the rRESPA discussion in Section 4. We assume current forces f are known at the beginning of the timestep.

(1)	$v = v + f\frac{\Delta t}{2m}$
(2)	$x = x + v\Delta t$
(3)	Communicate new x
(4)	Reneighbor if necessary (atoms migrate)
(5)	Compute bonded and non-bonded forces f
(6)	Communicate f if necessary
(7)	$v = v + f\frac{\Delta t}{2m}$

Figure 1: A parallel MD timestep.

In step (1), velocities are advanced half a timestep and used in step (2) to advance the positions a full timestep. Each processor performs these operations only on its owned atoms. To compute bonded and non–bonded forces on its atoms, each processor will need to know the current positions of other atoms in neighboring boxes. These positions are communicated in step (3) which can be done rapidly by 6 swaps with neighboring processors on a 3–d stencil [10]. We use neighbor lists to efficiently compute the non–bond forces. As atoms move, these lists must be reformed every few timesteps in step (4) using both owned and other atoms. At the same time atoms that have entered/exited a processor's box are moved to the new owning processor. When this occurs all of the owned atom information including molecular connectivity must migrate with the atom. Now in step (5) all the forces of equation 1 can be computed. (We defer the discussion of the long–range portion of the Coulombic forces to the next section.) If Newton's 3rd law is used to only compute a force interaction once when 2 (or more) atoms reside in different boxes, then processors will have accumulated partial forces on other atoms at the end of step (5). These forces are communicated in step (6), an operation similar to step (3). Finally velocities of owned atoms are advanced the final half step in step (7) using the new forces.

One point that was skipped in the above discussion is how to *find* the atoms that participate in various bonded interactions. Consider that a processor only stores the global tag numbers of the 2 (or more) atoms in a bonded interaction. Each global tag must be mapped to a local memory index within the *owned* or *other* atom data structures before the relevant atomic information can be accessed. This is complicated by the fact that these data structures change whenever atoms migrate in step (4). The fastest way to find these atoms is to maintain a N-length indirection vector T on each processor where T_i is the local index of the atom with global tag i. Alternatively, if memory constraints do not allow for this storage, a hash table can be maintained which will return the appropriate local index when a value from 1 to N is hashed.

3 Parallel PPPM

We now revisit the computation of the last term in equation 1. A detailed comparison of Ewald, fast multipole (FMM), and particle–mesh (PPPM) methods with an emphasis on relative computational efficiency is given in [12]. A salient conclusion is that in addition to being less complex to implement, PPPM is faster in practice than FMM for systems of any reasonable size. Here we focus on the issues relevant to implementing PPPM in parallel. The basic computational idea is to replace the point–charge Coulombic term in equation 1 with an equivalent expression for extended charges centered on the original atomic positions, namely

$$(2)\ \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{q_i q_j}{r_{ij}}\ erfc\left(G r_{ij}/\sqrt{2}\right) + \frac{1}{2}\int\int \frac{\hat{\rho}(r)\hat{\rho}(r')}{|r-r'|}\ dr dr' - \frac{G}{\sqrt{2\pi}}\sum_{i=1}^{N} q_i^2$$

where $\hat{\rho}_i(r) = q_i (G^2/\pi)^{3/2}\exp[-G^2(r-r_i)^2]$ is the Gaussian density that represents an extended charge and $\hat{\rho}(r) = \sum_i \hat{\rho}_i(r)$.

The first term in this expression is the usual Coulombic interaction now multiplied by an error function that forces it to go (nearly) to zero at a user–specified cutoff distance r_c. The choice of r_c together with a desired accuracy criterion (e.g. 0.01%) determine G. This term is thus the near–field portion of the Coulombic energy and is computed in LAMMPS at the same time as the van der Waals contribution — as a sum over nearby pairwise interactions using a neighbor list. The last term in equation 2 is a constant.

The remaining term is Coulombic energy due to the interaction of the extended charges. In the parallel PPPM algorithm outlined in Figure 2, it is evaluated by first discretizing the charge density $\hat{\rho}$ on a 3–d grid that is superposed over the periodic physical domain of the simulation. Each processor owns a portion of the grid corresponding to its 3–d spatial–decomposition box. Additionally it owns some ghost points that reside in neighboring processor's boxes.

(1) Interpolate extended charges to grid.
(2) Sum charge from grid ghost points.
(3) Send charge values from spatial– to FFT–decomposition.
(4) Solve Poisson's equation on grid via FFTs.
(5) Send field values from FFT– to spatial–decomposition.
(6) Acquire fields for grid ghost points.
(7) Interpolate field values on grid to atoms.

Figure 2: Parallel PPPM computation.

In step (1), each processor maps its extended charges $\hat{\rho}_i$ to its grid and ghost points. Hockney and Eastwood [8] have made this procedure more efficient by replacing $\hat{\rho}_i$ with an assignment function which extends over a relatively small number of grid points and compensate for this replacement by modifying the Coulomb interaction on the grid (see [8] and [12] for details). In step (2), the charge that overlapped onto ghost points is exchanged with neighboring processors.

We are now ready to transform the charge grid into Fourier space, except that on a parallel machine, a 3–d FFT is most efficiently performed if each processor owns entire columns of the 3–d grid. This is so that 1–d FFTs can be performed wholly on–processor using fast library routines. A 3–d FFT then consists of successive 1–d FFTs interleaved with transposes of the 3–d grid to bring new sets of columns onto the processors.

In step (3), we prepare for the FFTs by moving grid data from the spatial–decomposition to an FFT–decomposition where each processor owns a set of 1–d columns of the 3–d mesh. This requires each processor to send all of its grid data to a handful of different processors. Similarly each processor will fill in its FFT columns with the data received from several other processors. Though this communication is unstructured in nature, lists of what data is sent to whom can be pre–computed and used efficiently at every timestep.

Now in step (4) Poisson's equation is solved for the charge density $\rho(r)$ on the grid. A forward FFT is done to convert to $\rho(k)$ and the result is convolved with a pre–computed Green's function to form the electrostatic potential $\phi(k)$. The electric field on the grid is then computed by back–transforming the quantity $-ik\phi(k)$ which requires one FFT for each of the 3 real–space components.

Steps (5) and (6) are the inverse of steps (2) and (3). First the field values must be communicated from the columns each processor owns in the FFT–decomposition to the grid points in the box it owns in the spatial–decomposition. Then each processor's ghost points are filled in with values acquired from neighboring processors. Finally in step (7) the electric field values are used to compute forces on each atom by interpolating from the grid points that lie within the domain of each extended charge.

Two points are worth making at this juncture. First, the bulk of the time in the parallel code is spent in steps (1) and (7) which are all computation and in the 4 FFTs of step (4). The interpolation operations of steps (1) and (7) are costly because each extended charge overlaps with roughly 100 grid points. Communication occurs in steps (2-3) and (5-6) (and the FFTs) but does not dominate the run time [12]. The efficiency of the PPPM method comes from the fact that even if the real–space cutoff is held fixed, as is common in MD codes, the number of grid points will still typically be of the same order as the number of atoms. Then the overall scaling of the method is $N\log(N)$ due to the FFTs.

Second, the communication needed for the parallel PPPM algorithm is different if a replicated–data approach is used for the MD portion of the simulation. In a replicated–data framework each processor can own a portion of the FFT grid with some ghost points that overlap into neighboring processor's FFT grids. This produces a higher surface–to–volume ratio for ghost cell exchange in steps (2) and (6) than in the spatial–decomposition framework, but means steps (3) and (5) can be eliminated. However, in steps (1) and (7) each processor now has to scan the entire list of N atoms to determine which atoms fall within its region of the FFT grid for interpolation purposes.

4 Parallel rRESPA

As applied to MD, the basic idea of a multiple–timestep scheme is to sub–cycle with a short timestep on fast degrees of freedom such as bond vibrations. A CPU savings is achieved since the more costly non–bond and long–range forces are computed only once per long timestep. A variety of methods for doing this have been devised, but a general method, the rRESPA approach [16], has only recently been developed. The rRESPA methodology includes an operator calculus for deriving the equations of motion that maintain time–reversibility. The operator calculus enables one to determine the equations of motion for other relevant ensembles (e.g. NVT, NPT) [13, 16] and to treat a variety of physical problems. Thus, it avoids some of the pitfalls [1] that other multiple–timestep approaches have been subject to.

We adapt the standard timestepping algorithm of Figure 1 to illustrate a 2–level velocity–Verlet rRESPA algorithm in Figure 3. Non–bond forces f_{nb} are computed in the outer level which uses a long timestep of Δt_{nb}. Bonded forces f_b are computed in the inner level with a short timestep of Δt_b.

(1)	$v = v + f_{nb}\frac{\Delta t_{nb}}{2m}$
	Sub–cycle:
(2a)	$v = v + f_b\frac{\Delta t_b}{2m}$
(2b)	$x = x + v\Delta t_b$
(2c)	Communicate new x
(2d)	Compute bonded forces f_b
(2e)	Communicate f_b if necessary
(2f)	$v = v + f_b\frac{\Delta t_b}{2m}$
(4)	Reneighbor if necessary (atoms migrate)
(5)	Compute non–bonded forces f_{nb}
(6)	Communicate f_{nb} if necessary
(7)	$v = v + f_{nb}\frac{\Delta t_{nb}}{2m}$

Figure 3: A long parallel timestep within a 2–level rRESPA integrator.

In step (1) velocities are advanced half a timestep using only the forces due to non–bond interactions. Next, steps (2) and (3) in the standard algorithm of Figure 1 are replaced with steps (2a)-(2f) in the rRESPA formulation. Steps (2a)-(2f) update x and v due only to the bonded forces and are conceptually identical to steps (1)-(7) of the standard algorithm (with the exception of reneighboring). First, in steps (2a) and (2b) the velocities and positions are advanced a short timestep using only forces due to bonded interactions. Updated positions are communicated in step (2c) and the bonded forces (1st 3 terms of equation 1) are computed. Forces computed on other atoms are communicated back to the owning processors in step (2e) and the fully accumulated bonded forces are used to update v another half (short) timestep in step (2f). Depending on the physical problem and desired accuracy, steps (2a)-(2f) can be iterated on 2–5 times. Since reneighboring is only done to insure the non–bond neighbor lists are accurate, it does not need to be done until sub–cycling is ended in step (4). In step (5), the non–bond forces are computed and they are communicated and used to do a final update of v in steps (6) and (7).

We make two additional points. First, rREPSA is a hierarchical method which can be extended to (or contracted by) as many timestep levels as one desires. Note that if the sub–cycle factor is set to unity, the algorithm of Figure 3 becomes identical to that of Figure 1. A 3rd level can be added to Figure 3 by replacing steps (2b)-(2c) with a new level of sub–cycling over steps (3a)-(3f). We have implemented a 4–level rRESPA integrator in LAMMPS. The innermost loop is over bonds; the next level computes 3– and 4–body interactions; short–range non–bond interactions are computed at the next level; the long–range Coulombic portion of equation 2 is computed only at the outermost level. As we show in the next section, we can run with an innermost timestep about 8x smaller than the outermost timestep and still maintain accuracy.

Second, for parallel implementation, there are 2 kinds of interprocessor communication that must be done. The first is the exchange of updated positions. Unfortunately this update occurs in the innermost loop and thus the communication must be done every iteration of the shortest timestep. This is unfortunate since bonded calculations are quick enough that now communication costs will dominate the time to iterate on the innermost loop, thus limiting the speed–up afforded by rRESPA in a spatial–decomposition framework. The impact would be even more serious in a replicated–data algorithm where the cost of position updates is relatively much more expensive. Communication of forces (e.g. steps 2e and 6) must also be done after every force calculation, but only if Newton's 3rd law was used so that processors did not accumulate the entire force on each of its atoms (due to interactions with atoms owned by other processors). LAMMPS has a switch that allows Newton's 3rd law to be turned on or off at various levels of the rRESPA hierarchy. Our tests have shown it is typically fastest to not use Newton's 3rd law in the 2 innermost bonded interaction loops, but to use it for the non–bond computations. This means some bonded interactions are computed multiple times, but step (2e) is eliminated.

5 Results

We now present some performance results for LAMMPS simulations of a solvated biomembrane. The simulation consists of a lipid bilayer surrounded by water in a periodic box that is 34.5 by 60.6 by 34.5 Angstroms in size [15]. There are a total of 7134 atoms in the simulation with similar numbers of bonds, angles, and dihedral interactions. The y-dimension of the simulation box is normal to the membrane plane so by replicating the system in the x and z dimensions, models 4x and 16x larger are created. We simulated these 3 system sizes with a 1.0 femtosecond timestep using a 12 Angstrom cutoff (14 Angstroms for neighbor lists) for the van der Waals interactions and 10 Angstroms for the real–space portion of the Coulombic interactions. For the long–range Coulombic PPPM solver a 0.01% accuracy criterion was used which required a 32x32x32 FFT grid for the 7134–atom simulation, 64x32x64 grid for the 28536–atom case, and 128x32x128 grid for the 114144–atom run. The simulations were performed on an Intel Paragon at Sandia. The timings in Figure 4 are for constant NVE simulations; constant NPT runs are a few percent slower.

Problem Size	P=32	P=64	P=128	P=256	P=512
N = 7134	1.62	0.899	0.500	0.303	0.220
N = 28536			2.01	1.14	0.601
N = 114144					2.02

Figure 4: CPU time per timestep for MD simulations of varying size N on varying number P of Intel Paragon processors.

In these runs, the time spent in the long–range solver is typically less than half of the total time. The timings show reasonable scalability both as N is increased for fixed P and as P is increased for fixed N. On the small problem, load–imbalance begins to detract from the performance scaling when P is increased to the point there are only a few tens of atoms per processor. This is because in the spatial–decomposition boxes, there tend to be a few processors who own significantly more than the average number of atoms. This imbalance is smaller when the average N/P ratio is large.

We have also simulated the 7134–atom membrane system using the 4–level rRESPA algorithm of the previous section. Figure 5 lists the CPU time to run 100 femtoseconds of simulation time (with an earlier, slower version of the PPPM solver) with different rRESPA sub–cycling factors. In all cases energy was conserved over picosecond timescales to within a few tenths of a percent.

Δt_{short}	SC_1	SC_2	SC_3	Δt_{long}	CPU Time
0.5 fs	1	1	1	0.5 fs	90.5
0.5 fs	2	2	2	4.0 fs	22.2
1.0 fs	1	1	1	1.0 fs	46.3
1.0 fs	1	2	2	4.0 fs	24.9

Figure 5: CPU time in seconds for 100 femtoseconds of simulation time using rRESPA on 256 Intel Paragon processors.

The first and third lines are with rRESPA turned off (Sub–Cycle factor $1 = SC_2 = SC_3 = 1$). The total run time for 100 fs differs because timestep sizes of 0.5 and 1.0 fs were used. Lines 2 and 4 show the results when rRESPA is turned on. In line 2, a short timestep of 0.5 fs was used with sub–cycling factors of (2,2,2) which corresponds to a longest timestep of 4.0 fs. This reduced the overall run time by 4x to 22.2 seconds. Similarly, in line 4 with a short timestep of 1.0 fs, the run time was reduced by 2x to 24.9 seconds. In this case a sub–cycling factor of 2 could not be used between 2–body and multi–body force calculations without a significant loss of accuracy because the 1.0 fs timestep is already on the edge of instability. These parallel rRESPA speed-ups of 2-4x are similar to those found by others in their serial benchmarking of various systems [9].

6 Acknowledgments

The work described in this paper was performed as part of a cooperative research and development agreement (CRADA) between SNL and LLNL and

3 industrial partners: Bristol–Myers Squibb, Cray Research, and Dupont. The US DOE funded the work at SNL and LLNL. We give particular thanks to Terry Stouch at BMS for providing us the biomembrane system used for benchmarking purposes.

References

[1] J. J. Biesiadecki and R. D. Skeel, J. Comp. Phys, 109, (1993) p. 318.

[2] T. W. Clark, R. V. Hanxleden, J. A. McCammon, L. R. Scott, in *Proc. of Scalable High Performance Computing Conference*, IEEE Comp. Soc. Press, Los Alamitos, CA, 1994, p. 95.

[3] T. Darden and D. York and L. Pedersen, J. Chem. Phys., 98, (1993) p. 10089.

[4] H. Q. Ding and N. Karasawa and W. A. Goddard III, J. Chem. Phys., 97, (1992) p. 4309.

[5] P. Ewald, Ann. Phys., 64, (1921) p. 253.

[6] D. Fincham, Molec. Sim., 1, (1987) p. 1.

[7] L. Greengard and V. Rokhlin, J. Comp. Phys., 73, (1987) p. 325.

[8] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, Adam Hilger, New York, NY, 1988.

[9] D. D. Humphreys, R. A. Friesner and B. J. Berne, J Phys. Chem, 98, (1994) p. 6885.

[10] S. J. Plimpton, J. Comp. Phys., 117, (1995) p. 1.

[11] S. J. Plimpton and B. H. Hendrickson, J. Comp. Chem., 17, (1996) p. 326.

[12] E. L. Pollock and J. Glosli, Comp. Phys. Comm., 95, (1996) p. 93.

[13] P. Procacci and B. J. Berne, Mol. Phys., 83, (1994) p. 255.

[14] W. Smith, Comp. Phys. Comm., 62, 1991, p. 229.

[15] T. R. Stouch and H. E. Alper and D. Bassolino-Klimas, Supercomputer Applications and High Performance Computing, 8, (1994) p. 6.

[16] M. E. Tuckerman, B. J. Berne and G. J. Martyna, J. Chem. Phys., 97, (1992) p. 1990.

[17] A. Windemuth, in *Parallel Computing in Computational Chemistry*, ed. by T. G. Mattson, American Chemical Society, 1995, p. 151.