SAND 2000-0157J

Parallel Atomistic Simulations*

Grant S. Heffelfinger
Materials Simulation Science Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1111

* This work was performed at Sandia National Laboratories which is operated for the DOE under contract number DE-AC04-94AL85000.

ABSTRACT

Algorithms developed to enable the use of atomistic molecular simulation methods with parallel computers are reviewed. Methods appropriate for bonded as well as non-bonded (and charged) interactions are included. While strategies for obtaining parallel molecular simulations have been developed for the full variety of atomistic simulation methods, molecular dynamics and Monte Carlo have received the most attention. Three main types of parallel molecular dynamics simulations have been developed, the replicated data decomposition, the spatial decomposition, and the force decomposition. For Monte Carlo simulations, parallel algorithms have been developed which can be divided into two categories, those which require a modified Markov chain and those which do not. Parallel algorithms developed for other simulation methods such as Gibbs ensemble Monte Carlo, grand canonical molecular dynamics, and Monte Carlo methods for protein structure determination are also reviewed and issues such as how to measure parallel efficiency, especially in the case of parallel Monte Carlo algorithms with modified Markov chains are discussed.

I. INTRODUCTION

Classical simulations of materials and fluids have long consumed enormous CPU-hours of available computing resources. Historically, simple interaction potentials such as hard-spheres and Lennard-Jones were employed to model small atomic systems. As computers became increasingly powerful, more realistic interaction potentials were employed to model ever-increasing systems sizes, including molecular systems. While massively parallel (MP) computing hardware arrived over 10 years ago, widespread acceptance was delayed due to ensuing debate over whether such architectures could yield speedups anywhere near the number of processors employed. Furthermore, widespread use of MP hardware required the development of new companion parallel algorithms for simulation methods, the difficulty of which is largely dependent on the method. This work contains a review of the effort to meet that need for atomistic simulations, including molecular dynamics (MD), Monte Carlo (MC), and related methods. In the interest of focus, some related topics have been omitted (e.g. simulation methods involving a hybridization between an atomistic method and a non-atomistic method, such as quantum MD). For similar reasons, some of the parallel algorithms designed or optimized for certain computer architectures (e.g. special purpose computer hardware) have also been omitted.

For uniformity, algorithms have generally been characterized by their speedup or efficiency, both indications how well they utilize the available number of processors. In general, speedup and efficiency are defined as

$$S = \frac{C_p}{C_P}, \text{ and} \tag{1}$$

$$E = \frac{pS}{P} \times 100\,\%, \tag{2}$$

where C_p is the CPU time on a smaller number of processors (usually 1), p, and C_P is the CPU time on a larger number of processors, P. Thus if an algorithm requires 28 seconds on 4 processors and 4 seconds on 32 processors, the speedup and efficiency are 7 and 87.5%, respectively. The reader should be aware that such measures of parallel efficiency are not only algorithm dependent but also machine dependent, often due to the relative speeds of the machines communication and computation capabilities. While such metrics are best for characterizing parallel algorithms they are less useful for predicting which approach to a simulation problem will actually produce faster results in wall clock time. However, given that such questions are highly machine dependent and that much of the data reported in the literature is for hardware no longer in existence, wall clock comparisons on different hardware are avoided.

II. PARALLEL SIMULATION METHODS

Parallel algorithms developed for atomistic simulation methods fall into two categories, those which simply involve distributing the computational workload of the existing serial algorithm over more than one processor, and those which involve modifying the simulation method itself so as to make it parallelizable. While molecular dynamics falls into the first category, Monte Carlo and other methods can be in either category.

A. Parallel Algorithms for Molecular Dynamics Simulations

Molecular dynamics is a widely used simulation tool for investigating dynamic molecular-scale phenomena such as diffusion as well as obtaining equilibrium thermodynamic data useful for verifying statistical mechanical theories [1].

While the computational requirements of molecular dynamics have been carefully analyzed [2, 3], it is agreed that the most time-consuming step of a molecular dynamics simulation is the calculation of the forces. Because this calculation is inherently parallel, parallel MD algorithms were among the first parallel algorithms to be developed. Several parallel MD algorithms have emerged over the years which have become sufficiently sophisticated to enable the simulation of increasingly complicated systems (e.g. charged molecular systems) on massively parallel supercomputers. As Smith [4] pointed out in an early discussion (1990) of parallel algorithms for molecular dynamics, three algorithms with good load balancing are replicated data, systolic loop, and parallelized link-cells (which will henceforth be referred to spatial decomposition). Replicated data and spatial decomposition (and variations) survive today as competitive parallel MD algorithms and are discussed along with other, newer, methods below.

Because applying molecular dynamics to biomolecules, which are generally charged and solvated in water, is of great interest to the simulation community, several methods have been developed to calculate the Coulombic contribution to the potential energy and forces in a molecular dynamics simulation [1, 5]. In the text which follows, the parallel implementations of such methods are briefly discussed according to their parallel decomposition strategy.

Replicated Data Parallel Molecular Dynamics Algorithm

Replicated data (also known as the atom decomposition) is intuitive: the *N* atoms in a molecular dynamics simulation are split evenly across *P* processors. In its simplest form, atomic fluids or solids with short-ranged two-body interaction potentials (such as Lennard-Jones), the method is quite simple to implement, especially for atomic systems [4, 6-12].

Briefly, each processor owns *N/P* atoms for the duration of the simulation. At the beginning of each time-step, a processor computes the forces between its atoms and the rest of the system's atoms, updates the positions and velocities of its atoms, and then participates in an all-to-all communication in which every processor obtains the new positions of all atoms in the system in preparation for the next time-step. This communication in which each processor sends its *N/P* updated atom coordinates to all other processors is the key limitation of the method because it scales as *N*, independent of *P*, even while the computational cost is an optimal *O(N/P)*. Thus the communication cost of the algorithm sabotages performance on large numbers of processors. Plimpton [13] and others [14] provide a thorough discussion of the method for simple Lennard-Jones benchmark fluids and discusses the merits of the replicated data algorithm relative to two other parallel decompositions for such systems and so we turn our attention next to replicated data implementations for multi-atom species.

The replicated data algorithm has been widely used for molecular systems because the calculation of the intra-molecular forces, bond, angle, and torsional, is easily load-balanced across processors. In early work on polymers Drake et al. [15] achieved efficiencies approaching 100% on up to 64 processors of an Intel iPSC hypercube. Using replicated data with the PVM (Parallel Virtual Machine, available from Oak Ridge National Laboratory) message passing library. Müller-Plathe et al. [16] tested four benchmark systems, obtaining speedups of up to 38 on 65 processors of an Intel Paragon with a code named **PARALLACS**. The **GROMOS** (**GRO**ningen **MO**lecular Simulation Software) code was parallelized with the replicated data algorithm [17, 18] yielding a speedup of about 67 on 128 processors. Follow-on work with **GROMOS** consisted of implementing the neighbor list/linked cell force calculation algorithm [1] in parallel and a discussion of parallel strategies [19] for the SHAKE algorithm [20]. In similar work, the **CHARMM** (Chemistry at **HAR**vard Macromolecular Mechanics) code, including constraint dynamics with the SHAKE algorithm, was also parallelized with replicated data, yielding a speedups of 80 on 128 processors in one effort [21] and 4 on 32 processors in another [22]. **CHARMM** has also been parallelized for clustered workstations communicating with PVM library and for a Cray YMP [23]. Even with PVM's limited communications capability, a speedup of 6 on 8 processors was obtained [24]. Other work on coupled workstations includes an implementation of replicated data molecular dynamics in a code named **WESDYN**. Using FORTRAN and Linda, a speedup of 3 on 4 processors was obtained [25]. The **AMBER** molecular dynamics code has also been parallelized with the replicated data algorithm, not only for the Cray YMP [23] but for MIMD parallel machines as well. Timings on an nCUBE [26] demonstrated a speedup of 50 on 64 processors for one benchmark problem with long-ranged forces (30Å) while a 4343 atom lipid bilayer test case yielded a speedup of 23 on 32 processors of a Cray T3D [27].

Replicated data methods have also been used with systolic loop methods [28] for communication to parallelize bonded molecular dynamics simulations [29, 30] and with non-equilibrium molecular dynamics (NEMD). In the case of NEMD, the implementation of the "sliding boundary" conditions necessary for planar Couette flow is fairly straightforward yielding parallel simulations for investigating the viscosity for molecular liquids [31, 32]. As with atomic fluids, Plimpton and coworkers have provided a detailed discussion of the replicated data algorithm for uncharged bonded systems and a careful comparison to two other parallel molecular dynamics algorithms [33] on systems of up to a million atoms and 1024 processors.

Various treatments of long range forces have been used with the replicated data parallel molecular dynamics algorithm including direct calculation [29] and the use of cut-off's [16], both of which are intuitively easy to implement with the replicated data algorithm. Another approach which has been employed is the Ewald summation [34] which scales as $N^{3/2}$. In this method, the total force is split into three parts, 1) a sum in real space with quadratic dependence on N, the number of ions in the system, 2) a sum in reciprocal space, with linear dependence on N, and 3) a self-interaction part which can be calculated at the beginning of the simulation and remains constant throughout. Normally the real space summation is truncated with the same cut-off as the usual van der Waals forces meaning that this part of the method can be handled with established molecular dynamics (including parallel) algorithms and becomes near linear in N rather than quadratic. Smith [35] presents two strategies for employing replicated data with the Ewald sum, the "reduced ion list" (RIL) in which each processor owns a subset of the ions in the system, and the "reduced k vector list" (RKL) in which each processor owns a unique subset of the k vectors in the Ewald sum. A speedup of 23 was obtained on 32 processors of an Intel iPSC/860 for a system of 4096 ions of an alkali halide system with the RIL algorithm. In follow-on work, Smith et al. [36] discuss a parallel replicated data code name **DL_POLY_2.0** which can treat Coulombic interactions with the Ewald sum method, cut-offs, or the reaction field method, although details are not given. Others have employed the Ewald sum with replicated data in the *NPT* ensemble on a shared memory machine and workstation clusters [37].

Another form of Ewald summation, particle-mesh methods [38] which scale as $N\log N$ for fixed cut-offs, have also been implemented in parallel molecular dynamics codes employing the replicated data molecular dynamics. As with regular Ewald summation, the total force is split into real-space, reciprocal space, and self-interaction parts with the real-space part handled similarly to the van der Waals forces, decomposed by atom over the available processors. The reciprocal part, however, is handled differently with the atomic charges and positions replaced by charges on a regular 3-dimensional grid which is then solved via FFT's, making the reciprocal space sum more computationally tractable. This calculation is parallelized by distributing the charge grid over the processors, taking care to optimize the FFT calculation and minimize interprocessor communication [39]. A recent comparison of the particle-mesh method indicates that it is faster than the fast multipole method (discussed below) and easier to implement [40], however, as Brown et al. [41] point out, given the history of comparisons between the two methods, it still unclear as to which is the best method. In any case, Crowley et al. achieved speed-ups in the reciprocal sum calculation of about 40 on 150 processors and 125 on 200 processors for the particle-mesh Ewald calculation itself with this approach [39].

Spatial Parallel Molecular Dynamics Algorithm

A second class of methods for parallelizing molecular dynamics simulation is known as spatial- or geometric-decomposition [7]. In this method, the simulation box is subdivided into cubic domains, one per processor. Each processor is responsible for computing the forces and new positions for the atoms inside its box. Interprocessor communication is required as each processor calculates the short-range force on its atoms but sizing the dimension of the domains equal to or larger than the cut-off of the interaction potential enables this communication to be limited, e.g. each processor needs only to know the coordinates of the atoms in surrounding domains. (Thus one limitation of the spatial decomposition is that this is not always possible, depending on N, P, and r_c, the number of atoms in the system, the number of processors, and the cutoff, respectively.) Additional interprocessor communication is required if an atom migrates from one domain (and processor) to another; possession of this atom is "handed-off" in a similar communication.

While the computation in spatial parallel molecular dynamics scales as N/P, the communication cost scales as $(N/P)^{2/3}$ due to the fact the interprocessor communication for this scheme is normally carried out through a three-step process through the 2d surfaces of a 3d volume (Figure 1), although systolic loop [42, 43] and other methods have also been used [30, 44]. While this is the method of choice for very large systems (tens of thousands of atoms), its chief limitation is load balancing difficulties experienced for systems with heterogeneous density. The spatial decomposition is also more difficult to implement than replicated data, particularly for molecular systems where molecules may straddle domains meaning that arrays describing each atom must also contain the topological information about the molecule to enable the computation of the intra-molecular forces.

In early work, Rappaport and Clementi [45, 46] coupled 4 FPS 264 array processors to simulate 2d flow around a sphere with molecular dynamics by assigning domains sized larger than cutoff of the interaction potential to each processor. For a system of 170,000 atoms, roughly 40,000 per processor, approximately 1000 atoms were close enough to the domain borders to be included in the interprocessor communication, which was carried out through shared memory and accounted for 3% of the overall time. The spatial decomposition method was also used to investigate 2-dimensional spinodal decomposition for a 7688 atom Lennard-Jones system, achieving an efficiency of 52% on 26 transputers [47]. Pinches et al. [48] carried out an investigation of the efficiencies of this algorithm achieving speedups of approximately 42 on 64 processors of an Intel iPSC/2 for a 35,152 atom three-dimensional short-ranged Lennard-Jones system. Other reports on algorithmic investigations for various implementations and simple systems, including the embedded atom method (EAM) for metals, followed this work [13, 14, 49-53] including the significant improvement of using large domains on each processor with linked-cells and/or using linked cells to construct neighbor-lists to enable simulations of simple systems with hundreds of millions of atoms [13, 54-58] to be achieved, generally with greater than 600 atoms per processor [57]. However, recent improvements to the spatial decomposition have enabled good scaling behavior down to a few hundred atoms per processor as well as about 10% improvement in performance for the spatial decomposition for some systems [59]. Another group of researchers implemented a version of the decomposition which employs a multiple time-step method [60] and three-body interactions [14, 61, 62] and then applied to several problems in porous silica [63, 64].

Plimpton's most recent results demonstrate that not only is the parallel spatial molecular dynamics algorithm well-suited for traditional large-scale massively parallel platforms (e.g. the Intel Teraflops machine at Sandia National Laboratories) but it also scales well on clusters of commodity computing and networking components (e.g. the Cplant machine [65] at Sandia National Laboratories which consists of hundreds of DEC alpha processors connected by a Myrinet communications network). A comparison of the CPU time per time-step for the spatial decomposition on the Intel Teraflops and Cplant for a fixed system size (N = 32,000 atoms) in increasing numbers of processors and for scaled system sizes which increase with numbers of processors (32,000 atoms per processor) can be found in Figure 2 and Figure 3, respectively [66].

Implementations of the spatial parallel molecular dynamics algorithm have also been developed for uncharged bonded systems, despite the added difficulties that such systems present, largely due to the fact that molecules may straddle processor domains and for some systems load imbalance may occur due to density nonhomogeneity. Such systems imply additional interprocessor communication due to the fact that information about the intramolecular connectivity of atoms in a molecule lying in different processor domains will have to be exchanged between processors. Furthermore, additional communication will be required to achieve bond-length constraints with algorithms such as SHAKE [20].

Esselink and Hilbers demonstrated a spatial decomposition for such systems which required no extra interprocessor communication as long as the range of the bonded potentials is no greater than that of the non-bonded potentials but did not incorporate bond constraints [67]. Brown et al. [68] presented an implementation of the spatial decomposition which accommodates bond constraints through the use of the SHAKE algorithm. They tested their method on a system of 64,000 atoms consisting of 640 chain molecules with 100 CH_2 units per chain. Their algorithm runs 5.7 times faster on 1000 processors than on 64 processors of an Fujitsu AP1000 machine, for an efficiency of 36%. Their algorithm has also been generalized to molecules of arbitrary connectivity [41]. Eisenhauer and Schwan [69] tested a similar algorithm on a system of 300 *n*-hexadecane molecules (16 atoms each) on a substrate of 1350 gold atoms achieving efficiencies of 25% on 60 processors of a KSR computer.

A version of **GROMOS** called **EULERGROMOS** which uses the spatial decomposition method has also been developed [70]. Three systems were tested, a dipeptide with 337 solute and solvent atoms, a 10,914 atom myoglobin molecule, and a 10,406 atom acetylcholinesterase dimer (AChE) in 121,257 solvent atoms. On 128 processors, efficiencies of about 50% for the medium-sized myoglobin system and 15% for the small dipeptide system were obtained while on 512 processors of the Intel Touchstone Delta system at CalTech, efficiencies of about 25% for the large AChE system and 15% for the medium myoglobin system were obtained. These performance drops with decreasing system sizes are due to relative magnitude of the communication and computation requirements; economies of scale are obtained in communication with larger computation requirements for the spatial decomposition. Clark et al. [70] also compare **EULERGROMOS** to **UHGROMOS**, the replicated data implementation of **GROMOS** [18] on the medium-sized myoglobin system. The spatial decomposition **EULERGROMOS** was found to outperform the replicated data **UHGROMOS** on more than 100 processors.

As with the replicated data algorithm, the spatial decomposition has also been employed to parallelize NEMD although the implementation of "sliding boundary" conditions necessary for planar Couette flow, requires a bit more effort than before [71-73, 32]. Hansen et al. [71] have enabled parallel spatial NEMD simulations with moving boundary conditions by allowing the system itself to deform with shear rather than having the periodic images of the system move relative to the system itself (which would require complicated time-dependent interprocessor communication patterns). As Allen and Tildesley [1] point out, such a method will eventually mean that angles of the simulation system will become extremely acute. Hansen et al. remedy this problem by resetting the shape of the system when it reaches 45^o. In this deforming cell method, the system boundaries deform with the shear flow such that the shearing boundaries always align with the image cells.

Implementations of the spatial decomposition method for use with charged system are much more numerous then those for replicated data. Brown et al. [41] employed the Ewald summation method with spatial decomposition by having each processor compute the sum over reciprocal space for each atom in its domain. The real space portion was simply incorporated into the nonbonded force calculation normally employed in the spatial decomposition. This method achieved a near linear speedup of 7.7 on 8 processors for a benchmark system of one molecule of echistatin, a small peptide with 713 atoms, one chloride ion and 6903 water molecules [41]. Two other similar algorithms appear in the literature, both for single instruction multiple data (SIMD) architectures, one for the Connection Machine CM-200 [74] and another for a MasPar MP-2 [75] although few details are provided in either case.

Other work on the spatial decomposition with Ewald sums includes the use of the two dimensional Ewald approximation developed by Hautman and Klein [76] in what is called "a hybrid spatial decomposition and systolic loop algorithm" [77]. This scheme was designed to enable the SHAKE algorithm to be performed locally thus restricting the assignment of whole molecules to single processors. Although only an broad outline of the method is presented, each processor carries out the calculation of the intra-molecular forces for its molecules and then participates in a global communications step to obtain the positions of all other charges in the system. The modified Ewald summation is then carried out in parallel with each processor calculating the real-space contribution due to the charged sites in its molecules and a share of the reciprocal space calculation decomposed over the k-vectors. Finally, the intermolecular interactions are "distributed by distributing the neighbor-cells amongst the processors" on "an arbitrary basis, rather than on a positional basis as would be the case for the spatial decomposition." Parallel efficiencies of 40% were obtained for a 128 molecule bilayer system and a total of 5,120 atoms [77].

Particle-mesh Ewald methods, discussed above in the context of the replicated data algorithm have also been implemented for the spatial decomposition. The real-space part is decomposed by domain, thus each processor calculates not only the van der Waals interactions for the atoms in its domain but also the real-space part of the Ewald sum. The reciprocal space portion is calculated in parallel by moving grid data from the spatial decomposition to an FFT decomposition where each processor owns part of a 3-dimensional mesh. Efficiencies of 46% were obtained for 512 processors as compared with 32 processors of an Intel Paragon on a 7,134 atom membrane system [78].

Another class of methods for treating long-ranged forces is multipole methods [79]. The basic idea behind these methods is that an atom interacts with a distant group of atoms as if it were interacting with a single particle at the center of mass of the group. Thus, in the multipole method, a hierarchy of grids is superposed on the simulation domain and atoms are grouped into cells at the finest grid level with multipole coefficients calculated for each cell. By collecting and combining these coefficients at coarser grid levels, the interaction between an individual atom and a group of atoms at a distance (in a cell at the finest grid level) can be approximated. The method can be implemented in parallel [80-89] in a spatial decomposition framework by distributing the various grids across the processors. Communication is required between processors owning adjacent grid cells and as the algorithm sequences up and down through the grid hierarchy. Using a multilevel preconditioned conjugate gradient method for the fast multipole algorithm in the context of spatial decomposition, Nakano obtained a scaled efficiency with 414,720 atoms per processor of over 90% on 64 processors [87]. Kalé et al. [82] employed a combination of the spatial and force decompositions in a sophisticated object oriented code called **NAMD2** which can treat electrostatics with cut-off's, fast multipole, or particle mesh Ewald, achieving a speedup of 143 on 192 processors of the ASCI Red Intel Teraflop machine at Sandia National Laboratories for a system with 12Å cutoffs. Finally, Lim et al. [88] demonstrated a spatial decomposition implementation of the cell multipole method [90] in a code called MPSIM. Linear scaling with system size and near linear scaling with number of processors was obtained for up to 10 million atoms and 500 processors.

Load-balancing a spatial decomposition molecular dynamics code is an issue for systems with inhomogeneous densities and thus several schemes have been developed to address this problem. This can be done locally [91-93] with neighboring processors redistributing the work to achieve balance or globally [70, 94], with a master processor periodically re-decomposing the problem across the available processors. On simulations of a system of highly irregular α-quartz nanoclusters, Nakano et al. [93] employed adaptive curvilinear coordinates and simulated annealing to determine the optimal coordinate system with minimal load imbalance and communication costs. On 32 processors the execution time was reduced by a factor of 4.2 at a cost of 3.7% extra elapsed time due to the load balancer. As Srinivasan et al. [94] point out in a summary of load balancing strategies, re-distributing the workload at each time-step is expensive, schemes designed for specific processor topologies limit portability, restricting the domain size to the cut-off distance of the interaction potential limits the number of processors which can be used, and iterative methods may be limited by convergence difficulties if the needed load balancing changes faster than the rate of the convergence method. Thus developing an appropriate load balancing strategy for spatial molecular dynamics is highly dependent on the systems to be simulated and the available computer hardware.

Other Parallel Molecular Dynamics Algorithms

Several other parallel molecular dynamics algorithms have also been developed. The "embarrassingly parallel" method in which *P* simulations are carried out on *P* processors to achieve better statistics is normally reserved for Monte Carlo simulations but has been applied to molecular dynamics as well [95] and was found to produce better precision per processor than the spatial decomposition method [68]. Systolic loop methods [28] for decomposing the force matrix over multiple processors and cycling atom data around a ring of processors fall into a general category of "force decompositions." However, systolic loop methods are designed for multi-processor machines with limited inter connectivity and have faded from the scene for use with molecular dynamics [96-101, 4] as massively parallel computer hardware has evolved to very high degrees of interprocessor connectivity. The reader interested in Coulombic interactions should note that several of these efforts included a direct calculation of these long-ranged forces [98, 100, 102] while another approach [103] includes only the Coulombic contribution due to nearby atoms in every time-step while calculating the Coulombic contribution from distant atoms less frequently [101]. Work has progressed on related topics, however. For example, Trobec et al. employed a ring topology on a MIMD computer to investigate improved integrating algorithms for solving Newton's equations of motion [102, 104, 105].

The force matrix containing the *ij* interactions of the pair potential has also been decomposed over parallel processors. A key feature of such decompositions of the force matrix is that double counting the forces to minimize interprocessor communication is unnecessary because processors are not responsible for specific atoms. Thus, the fact that the forces between two atoms (and due to each other) are equal and opposite (Newton's 3rd law) can be used as with serial MD codes.

Such "force decompositions" have been carried out by sub-blocks [106] and element by element [107-109]. These approaches are not general in that they were designed for long-range force systems and therefore require the calculation of the forces for all pairs of atoms meaning that neighbor lists cannot be used. Boyer and Pawley [106] employed a direct calculation of the Coulombic interactions while Nguyen et al. [107] employed the Ewald sum method, calculating the real-space terms with the van der Waals interactions and distributing the reciprocal-space terms over the available processors. Other more general decompositions of the force matrix have been developed. Schreiber et al. [110] employ a master-slave approach for a solvated peptide system. In their approach, the master processor constructs the neighbor list, enabled by using cut-offs for the Coulombic contributions, and computes the interactions within the peptide while the slaves compute the interactions between peptide and the solvent and between solvent molecules. The full neighbor list is then gathered onto the master which can then distribute the work over the slave processors with a staircase decomposition. This approach yielded overall efficiencies of 82% on 19 processors.

For short-ranged forces, Plimpton et al. [13, 33, 111-113] employ a block decomposition of the force matrix over P processors yielding $O(N/\sqrt{P})$ scaling communication cost, an improvement over the $O(N)$ scaling of replicated data algorithm which is essentially a row-wise decomposition of the force matrix. Plimpton concludes that while the strength of the force decomposition is that it works well for varying molecular topographies, it requires some preprocessing to ensure load balance and is not as efficient as the spatial decomposition for *atomic* systems larger than 10,000 atoms and load balanced macromolecular systems larger than tens of thousands of atoms [13]. Efficiencies of almost 90% were obtained with this force decomposition algorithm on up to 1024 processors for a 6,750 atom liquid crystal system in a regular 3-dimensional box [113] and 61% on 1024 processors for a 14,026 solvated myoglobin molecule in a spherical geometry [33]. The communication requirements to achieve good performance with Plimpton's force decomposition for $N \approx 50,000$ systems have been comprehensively analyzed by Taylor et al. [114].

In related work, Skeel implemented **GROMOS** on an Alliant FX/8 by decomposing the force calculation over the available processors and treating Coulombic interactions with cut-off's [115] while Murty and Okunbor [116] developed two force decomposition algorithms which do not double compute the *ij* interactions. Their algorithms, called force-row interleaving (FRI) and force-stripped row (FSR) have been tested on 32,000 atom systems with up to 16 processors of an Intel iPSC/860. Briefly, the FRI algorithm treats one row the force matrix at a time, multiple rows per processor which alternate to achieve load balancing while the FSR determines a priori the block of rows that balances the workload across the available processors.

Finally, we note that it is the femtosecond time-step which limits the time-scales achievable with molecular dynamics thus other approaches to speeding up a MD simulation have recently been investigated. For example, multiscale methods such as rRESPA [117] which extend the time-step by factors of 2-5 have been parallelized [78]. Other approaches include the recently developed hyper-MD [118] and parallel replica method [119] which extend the time-scale that can be simulated for single event transitions in small systems. Hyper-MD uses a bias potential to raise the energy away from transition states bias while the parallel replica method involves carrying out parallel MD simulations on each processor until the first transition occurs then providing this configuration to all processors and proceeding as before.

B. Parallel Algorithms for Classical Monte Carlo Simulations

Molecular dynamics provides both equilibrium and dynamic thermodynamic data (e.g. diffusivities). Monte Carlo methods [1], on the other hand, are not well-suited to simulating dynamic phenomena, but they are often faster than molecular dynamics or the only way available to obtain equilibrium data for some systems. Parallel Monte Carlo (MC) simulations have been difficult to implement due to the fact that atoms in a normal MC simulation are moved one at a time. While the bulk of the computational work in a MC simulation, the energy calculation of a proposed move, can be shared across processors, given that many MC simulations employ short-ranged potentials, this approach is inefficient because only a small subset of the system's atoms interact with the proposed move at any given point in the simulation. Therefore, any efficient massively parallel Monte Carlo algorithm for systems with short-ranged interactions must involve simultaneous moves on different processors. This can be accomplished by two approaches, depending on whether or not the parallel algorithm relies on modifying the simulation's Markov chain so as to make it parallelizable. In the text which follows, examples of both approaches are briefly discussed. (Note: While lattice systems are usually included in any general discussion of Monte Carlo methods in statistical physics and several parallel algorithms for such systems have been developed [120-123], these methods are not discussed below).

Parallel Monte Carlo Algorithms With Unmodified Markov Chains: In the "embarrassingly parallel" approach [124], the most obvious way to parallelize Monte Carlo methods, simultaneous, independent, simulations are carried out on parallel processors. Ensemble data from parallel MC simulations, identical in every respect except for the initial value of the random number generator seed, are averaged over all processors yielding better statistics than a single simulation of the same duration. This decomposition is easiest to program, but if the non-equilibrium "warm-up" phase is not short relative to the "production phase" (where the desired equilibrium data is accumulated), this decomposition cannot compete with a standard Monte Carlo program running for an extended time on a serial supercomputer. A second limitation of this approach is that the memory available on a single processor must be large enough to accommodate the entire system. The embarrassingly parallel approach has been carefully analyzed and found to be superior to a spatial parallel decomposition (discussed below) for short-ranged fluids for systems not hindered by the restrictions discussed above [125]. Interestingly, it has also been applied to molecular dynamics [95] as well.

A second approach to parallelizing Monte Carlo simulations which also does not employ a modified Markov chain is systolic loop methods [28, 126], an approach efficient only for systems with complex interaction potentials (so that the calculation time greatly exceeds the communication time) and for machines with small numbers of processors and limited memory per processor. Another approach, the "farm" algorithm [127], involves distributing the calculation of the energy of proposed moves over P available processors, where each processor is responsible for calculating the contribution to the energy of the proposed move due to some subset of the system's members. Jones and Goodfellow [128] demonstrated that the efficiency of the farm algorithm falls off dramatically with increasing numbers of processors P for atomic MC simulations, dropping below 40% for more than 10 processors. However, Ulberg and Gubbins [129] employed this method on a Connection Machine-2 for Monte Carlo simulations of TIP4P water in graphite pores. They achieve speed-ups of approximately 3-5 over a Cray Y/MP for 512 water molecules but provide only a limited discussion of their parallel algorithm and its performance. Systolic loop methods have also been employed to parallelize the calculation of the potential energy of a protein, including a direct calculation of the long-ranged Coulombic forces [130].

Parallel Monte Carlo Algorithms With Modified Markov Chains: Modifying the Monte Carlo simulation algorithm itself so as to make it parallelizable has been accomplished in several ways. However, because modifying the Markov chain of a Monte Carlo simulation changes the amount of work needed to achieve a desired accuracy of a statistical result, the comparison of the efficiency of MC algorithms employing different Markov chains, whether parallel or not, requires careful analysis. For the parallel Monte Carlo algorithms discussed below, differing methods of assessing their efficiency were often employed and therefore are discussed in addition to the methods themselves.

"Multiparticle Monte Carlo [131]" simply involves proposing new positions for some subset *K* of the system's *N* atoms, calculating the total energy change due to this composite move in parallel, and accepting/rejecting the composite move. The efficiency of the method was obtained by calculating the correlation length, *l*, of the energy for a systems of 32 and 256 Lennard-Jones atoms as a function of *K*, the number of atoms in a composite move. As *K* increases, the maximum displacement which can be used while achieving a reasonable acceptance rate (37%) decreases. Thus this algorithm is doing more work, albeit faster due to multiple processors, while sampling phase space more slowly. For the 256 atom case, the correlation length with *K* equal to 256 was 83 times greater than that with *K* equal to 1.

"Hybrid Monte Carlo" is a method similar to multiparticle Monte Carlo. In this method [132], composite Monte Carlo moves are postulated by proposing new positions for all atoms in the systems by drawing velocities for each atom from a Gaussian distribution rather than randomly. This algorithm can be parallelized with any of the parallel molecular dynamics algorithms discussed previously. However, like multiparticle Monte Carlo, the performance of parallel hybrid Monte Carlo depends more on the statistical efficiency of the Monte Carlo algorithm itself (e.g. how far one can displace the system's molecules in the composite move and still obtain a reasonable acceptance rate) rather than just the parallel efficiency of the energy calculation itself. Loyens et al. [133] employed the spatial decomposition algorithm with hybrid Monte Carlo as part of a parallel implementation of Gibbs ensemble Monte Carlo (discussed further below). These authors found that in a gas phase, a time step much larger than that of molecular dynamics can be achieved with hybrid Monte Carlo. Conversely, in a liquid phase, the achievable time step of hybrid Monte Carlo is approximately equal to that of molecular dynamics. However, Brodz and de Pablo [134] found that a hybrid Monte Carlo time step for a silica system could be twice as large as that for molecular dynamics. Thus one must conclude that the statistical efficiency of hybrid Monte Carlo, and thus its parallel efficiency as well, is system dependent.

Another approach to developing a parallelizable Monte Carlo algorithm, the spatial decomposition method [125, 135, 136], relies on the fact that in simulations of short-ranged potentials, atoms separated by a distance greater than the range of the potential are energetically independent and can therefore be moved simultaneously without violating the constraint of detailed balance. This was accomplished by Heffelfinger and Lewitt [125], by taking advantage of the fact that for short-ranged interaction potentials, processors can work simultaneously so long as the geometric regions in which they operate are separated by a distance further than the cutoff length of the potential (Figure 4). The efficiency of this spatial decomposition method can be determined by defining a speedup, *S*, as,

$$S = \frac{C_{SE}\big|_{P=8}}{C_{SE}\big|_{P}} \tag{3}$$

where C_{SE} is CPU time *to reach a given value of the standard error* for the system potential energy on 8 processors (numerator) and P processors (denominator). Eight processors were chosen due to the fact that was the minimum number of processors the algorithm could employ. While the speedup obtained for one state point simulation carried out on 1024 processors was about 128, (near perfect speedup from 8 to 1024 processors), the embarrassingly parallel approach (*EP*) was found to be generally twice as efficient [125]. Thus, while the spatial parallel MC algorithm is well-suited for very large systems which would not fit within the memory of a single processor, (thus restricting the use of the embarrassingly parallel algorithm as discussed above,) it is otherwise not as efficient as the embarrassingly parallel approach. This is especially true for denser liquids. In the case of a Lennard-Jones system with a liquid density of $\rho\sigma^3 = 0.73$, the spatial decomposition did not reach the standard error achieved by the embarrassingly parallel approach in a reasonable number of CPU hours (Figure 5).

Another Monte Carlo algorithm, developed specifically for the grand canonical ensemble (constant chemical potential, μ, volume, V, and temperature, T), which is inherently parallelizable involves utilizing multiple canonical ensemble simulations (constant number of atoms, N, V, and T), each with a different N, which are sampled as they progress [137]. A parallelization strategy would involve running the independent *NVT* simulations on individual processors and using a host program to "hop" between the parallel MC simulations, accumulating statistics. This approach would have the same limitations as the embarrassingly parallel approach: each independent MC simulation would need to be warmed-up individually and small enough to fit within the memory limitations of the individual processors.

Chen and Hirtzel [138] also developed a parallel algorithm for grand canonical Monte Carlo simulations (GCMC) of a rather specific zeolite gas adsorption problem. Their algorithm, implemented for a Connection Machine CM2, involves using groups of processors to carry out simultaneous "microstate" GCMC simulations for varying numbers of gas molecules, up to the limited number able to be accommodated in a zeolite molecular sieve. Defining a "macro state" as a set of microstates with the same number of adsorbed gas molecules and determining the transition rates between these macro states yields a "macro state Markov Chain Model" (hence the name MSMCM [139-141]). They reduced the simulation time for a single phase diagram data point from tens of hours on a DEC Micro VAX II to a few minutes on some 16,000 processors of the CM-2, a respectable achievement although with an algorithm constructed for a very specific problem.

The issue of inserting and deleting molecules in a Monte Carlo simulation is not limited to GCMC. Rather, one may think of a Monte Carlo move of a single molecule from one position to another as an insertion followed by a deletion. While this generally presents little problem for atomic systems, such Monte Carlo moves can suffer from low acceptance rates for molecular systems, especially at liquid densities. This problem has been solved by carrying out molecular insertions by "growing" a molecule (inserting a molecule one atom at a time) at a new site and accounting for the bias the method introduces. Such "configurational bias Monte Carlo" or CMBC methods [142, 143] thus enable Monte Carlo simulations with insertion of macromolecules. Esselink et al. [144] have used a similar idea to construct a parallelizable Monte Carlo algorithm. Briefly, several trial positions are generated simultaneously (and if implemented on a parallel computer, on different processors). The trial position with the highest probability of being accepted is selected and the bias introduced is removed by adjusting the acceptance rules. When this Monte Carlo algorithm was implemented on a *serial* machine, gains of up to 40% were achieved [144]. This is because in the original CBMC, not only is the optimal configuration tested for acceptance but also configurations which have a low probability of acceptance. The strength of the Esselink et al. algorithm is that it eliminates expensive calculations for the configurations which have a very low probability of acceptance. To our knowledge, this algorithm has not been implemented on a parallel machine, however Esselink et al. do provide an estimate of parallel performance, speedups of about 5 on 20 processors for pentane and 13 on 20 processors for dodecane. The method was also employed to develop a parallel algorithm for Gibbs ensemble Monte Carlo, as discussed below.

Parallel Algorithms for Other Monte Carlo Methods

Parallel algorithms for other macromolecular Monte Carlo simulations have also been developed. Such MC atomistic simulations are generally aimed at elucidating the minimum energy configuration of a macromolecular system. In particular, the electrostatically driven Monte Carlo (EDMC) method [145-147] has been parallelized. Briefly, the goal of the EDMC method is to find the global minima, or lowest energy configuration, by carrying out an iterative search of the conformational hyperspace of small polypeptide molecules. Test conformations, generated with electrostatic predictions or with random sampling, are subjected to further energy minimization and then compared to the current configuration. Acceptance or rejection of the test configuration is then conducted via the standard Metropolis Monte Carlo criterion. The parallel algorithm developed for EDMC employs a master processor to carry out the conformational search and generate new conformations [145, 148]. The energy of a local minimum for the proposed conformations is computed by multiple slave processors, with one conformation per processor. The master processor periodically accepts a new conformation (which has been energy minimized by a slave processor) and then halts the program, loading the slave processors with new starting conformations. This adds a synchronization cost, but ensures that all processors are working with the best starting point for the global minimization.

After discussing the difficulty of accessing the efficiency of parallel algorithms for Monte Carlo calculations which employ different Markov chains, Ripoll and Thomas [148] settle on an "apparent speedup," $S(k,p)$,

$$S(k,p) = \frac{C(k,1)}{C(k,p)} \times \frac{f_p}{f_1} \qquad (4)$$

where $C(k,1)$ is the time for k iterations on a single processor, $C(k,p)$ the time for k iterations on p processors, f_p the number of required energy calculations required for the parallel program running on p processors, and f_1 the number of required energy calculations required for the program running on a single processor. The apparent speedup value obtained for the algorithm running on an Intel iPSC/2-SX is approximately 14 on 16 processors. However, as the authors point out, the method is best for small numbers of processors (less than 5) because of the EDMC method's underlying conformation acceptance rate which limits the attainable parallelism.

A second effort to develop a parallel EDMC algorithm was carried out for a Kendall Square Research KSR1 shared memory computer [147]. In this effort, a coarse-grained approach was used to distribute a set of conformations across a set of processors. These "coarse-grained, or CG" processors then distributed the task of computing the interaction potentials between the atoms among a set of "fine-grained, or FG" processors. The primary criteria for good performance for this algorithm was found to be an appropriate number of fine-grained processors: two for molecules up to 100 atoms, three for 400 atom molecules and four to five for 1000 atom molecules and nearly linear scaling behavior with number of coarse grained processors. In one example, for a BPTI molecule (886 atoms and 325 dihedral angles), speed-ups of about 6 to 20 were obtained with 10 coarse-grained processors, depending on the number of fine-grained processors.

Finally, parallel algorithms have also been developed for Monte Carlo simulations used to model polymer growth [149-152]. In these methods, the possible conformational angles of a new monomer, added to the end of a polymer chain are sampled with a Monte Carlo method. One effort to parallelize such a technique, briefly discussed in [151], claims nearly 100% efficiency.

C. Parallel Algorithms for Related Classical Simulation Methods

Molecular simulations of open systems (in the grand canonical ensemble) have until recently been confined to Monte Carlo methods as discussed above, which yield only equilibrium properties. Recent efforts, however, have yielded grand canonical molecular dynamics (GCMD) methods (see [153] for a recent discussion of such methods). To date, however, only one GCMD simulation algorithm has been completely parallelized [154, 155]. In this work, a parallel algorithm was developed for dual control grand canonical molecular dynamics or DCV-GCMD. DCV-GCMD method was developed [156, 157] to provide dynamical simulations of systems with chemical potential gradients and basically consists of two grand canonical Monte Carlo "control volumes" inserted in a molecular dynamics simulation. GCMC insertions and deletions are carried out within these two control volumes enabling local chemical potential control while molecular dynamics moves are carried out across the entire simulation domain. Because this approach is essentially a hybridization of Monte Carlo and molecular dynamics simulation methods, the parallel algorithm developed for the method combined spatial decomposition molecular dynamics and spatial decomposition Monte Carlo [154]. A schematic of a parallel DCV-GCMD simulation volume is shown in Figure 6. The method and its parallel algorithm have been extended to bonded molecular systems as well [155].

The Gibbs ensemble Monte Carlo simulation method [158], designed to efficiently determine the phase diagram of a model fluid, has also been parallelized. In this atomistic Monte Carlo simulation, two simulation volumes are employed, one for each phase of a two phase (e.g. gas-liquid or liquid-liquid) system. Three types of moves are employed: molecule displacement (within a simulation volume), volume changes of a simulation volume, and exchanges between the two simulation volumes. In their parallel implementation of the Gibbs ensemble method, Loyens et al. [133], developed parallel algorithms for each of the Gibbs ensemble Monte Carlo moves and tested their algorithm on 1 to 10 processors for systems of 1024 and 14,364 Lennard-Jones atoms. Because hybrid Monte Carlo [132] uses velocities drawn from a Gaussian distribution to project new trial positions for a composite move of the system's molecules, the energy calculation of the proposed new configuration can be carried out as in a of molecular dynamics simulation (and multiparticle Monte Carlo [131], discussed above). Loyens et al. [133] performed a parallel displacement move by combining spatial decomposition MD techniques with the hybrid Monte Carlo approach. The spatial MD algorithm can also be used to determine the energies due to the Monte Carlo volume change, with each processor calculating its contribution to the total system energy change due to the molecules in its simulation domain and then appropriately scaling their positions when a volume change has been accepted. Parallelizing the third Gibbs ensemble Monte Carlo move, exchange between the two simulation volumes, raises the same issues as parallelizing GCMC simulations due to the fact that molecules must be inserted and deleted. Here, Loyens et al. used the idea of Esselink et al. [144], attempting one molecule exchange (insertion in one simulation volume and deletion from another) at a time and using multiple processors to increase the acceptance ratio of the exchange. We can define a speed-up, S, for this operation as

$$S = \frac{C_1/A_1}{C_P/A_P}, \tag{5}$$

where C_1 is the wall clock time for the simulation on a single processor, C_P the wall clock time on P processors, A_1 the number of accepted exchange moves on a single processor, and A_P the number of accepted exchange moves on P processors. Using this definition, the parallel algorithm for the exchange move in the Loyens et al. parallel Gibbs ensemble method achieved speedups of 3 to 7 on 10 processors.

IV. CONCLUSIONS

Over the last 10 years or so, an enormous amount of effort has gone into developing parallel algorithms for atomistic simulation methods, a tribute not only to their computational demands but also to their usefulness. While many of these efforts focused on computers which are no longer marketed, an ever increasing body of knowledge about parallel atomistic simulations has emerged. With the advent of commodity-based cluster computing, which will provide economic computing resources at the terascale in the coming years, the methods reviewed in this paper will be essential to employ these resources to solve important and exciting problems governed by molecular-scale phenomena.

V. ACKNOWLEGEMENTS

The author thanks Steve Plimpton for allowing the inclusion of his unpublished results for spatial molecular dynamics on the CPlant cluster at Sandia National Laboratories as well as a critical reading of the manuscript.

VI. REFERENCES

1. M.P. Allen and D.J. Tildesley, Computer Simulations of Liquids (Clarendon, Oxford, 1989).

2. S. Gupta, Computer Physics Communications **70** (1992) 243.

3. L. Nyland, J. Prins, R.H. Yun, J. Hermans, H-C. Kum, and L. Wang, Journal of Parallel and Distributed Computing **47** (1997) 125.

4. W. Smith, Computer Physics Communications **62** (1991) 229.

5. D. Frenkel and B. Smit, Understanding Molecular Simulation, From Algorithms to Applications (Academic Press, San Diego, CA, 1996) and references therein.

6. G.C. Fox and S.W. Otto, Physics Today **39** (1984) 50.

7. D. Fincham, Molecular Simulation **1** (1987) 1.

8. D.C. Rappaport, Computer Physics Reports **9** (1988) 1.

9. R. Knirsch and K. Hofmann, Lecture Notes in Computer Science **634** (1992) 829.

10. W. Smith, Theoretica Chimica Acta, **84** (1993) 385.

11. W. Scott, A. Gunzinger, B. Baümle, P. Kohler, U.A. Müller, H-R. Vonder Mühll, A. Eichenberger, W. Guggenbühl, N. Ironmonger, F. Müller-Plathe, and W.F. van Gunsteren, Computer Physics Communications **75** (1993) 65.

12. M.B.Woods and C.O. Algord, Computers & Electrical Engineering **21** (1995) 159.

13. S.J. Plimpton, Journal of Computational Physics **117** (1995) 1.

14. R. K. Kalia, A. Nakano, D. Greenwell, and P. Vashishta, Supercomputer **54** (1993) 10.

15. J.B. Drake, A.K. Hudson, E. Johnson, D.W. Noid, G.A. Pfeffer, and S. Thompson, Computers & Chemistry **12** (1988) 15.

16. F. Müller-Plathe, W. Scott, W.F. van Gunsteren, Computer Physics Communications **84** (1994) 102.

17. T.W. Clark and J.A. McCammon, Computers & Chemistry **14** (1990) 219.

18. T.W. Clark, J.A. McCammon, and L.R. Scott, Proceedings of the 5th SIAM Conference on Parallel Processing for Scientific Computing, (SIAM, Philadelphia, PA, 1992) 338.

19. D.G. Green, K.E. Meacham, and F. van Hoesel, Lecture Notes in Computer Science **919** (1995) 875.

20. J.P. Ryckaert, G. Ciccotti, and H.J.C. Berendsen, Journal of Computational Physics 23 (1977) 327.

21. B.R. Brooks and M. Hodoscek, Chemical Design Automation News **7** (1992) 16.

22. S.L. Lin, J. Mellor-Crummey, B.M. Pettit, and G.N. Phillips Jr., Journal of Computational Chemistry **13** (1992) 1022.

23. J.E. Mertz, D.J. Tobias, C.L. Brooks, and U.C. Singh, Journal of Computational Chemistry, 12 (1991) 1270.

24. S. Fleischman, Molecular Simulation **14** (1995) 209.

25. T.G. Mattson and G. Ravishanker, ACS Symposium Series **592** (1995) 133.

26. S.E. DeBolt and P.A. Kollman, Journal of Computational Chemistry **14** (1993) 312.

27. J.J. Vincent and K.M. Merz, Journal of Computational Chemistry **16** (1995) 1420.

28. N.S. Ostland and R.A. Whiteside, Annals of the NY Academy of Science **439** (1985) 195.

29. J.F. Janak and P.C. Pattnaik, Journal of Computational Chemistry **13** (1992) 1098.

30. G. La Penna, S. Letardi, V. Minicozzi, S. Morante, G.C. Rossi, and G. Salina, Nuclear Physics B **63A-C** (1998) 985.

31. P.T. Cummings, Fluid Phase Equilibria **144** (1998) 331, and references therein.

32. H.D. Cochran, P.T. Cummings, S.T. Cui, S.A. Gupta, R.K. Bhupathiraju, and P.F. LoCascio, Computers & Mathematics with Applications 35 (1998) 73.

33. S. Plimpton and B. Hendrickson, ACS Symposium Series **592** (1995) 114.

34. P.P. Ewald, Ann. Phys., 64 (1921) 253.

35. W. Smith, Computer Physics Communications 67 (1992) 392.

36. W. Smith and T.R. Forester, Journal of Molecular Graphics 14 (1996) 136.

37. Z. Fang, A.D.J. Haymet, W. Shinoda, S. Okazaki, Computer Physics Communications **116** (1999) 295.

38. T. Darden, D. York, and L. Pedersen Journal of Chemical Physics 98 (1993) 10089.

39. M.F. Crowley, T.A. Darden, T.E. Cheatham III, D.W. Deerfield II, The Journal of Supercomputing 11 (1997) 255.

40. E.L. Pollock, J. Glosli, Computer Physics Communications 95 (1996) 93.

41. D. Brown, H. Minoux, B. Maigret, Computer Physics Communications 103 (1997) 170.

42. S. Chynoweth, U.C. Klomp, and L.E. Scales, Computer Physics Communications 62 (1991) 297.

43. F. Brugè, Computer Physics Communications **90** (1995) 59.

44. A. McDonough, A. Panijkov, S. Russo, and I.K. Snook, Computer Physics Communications **103** (1997) 157.

45. D.C. Rapaport and E. Clementi, Physical Review Letters **57** (1986) 695.

46. D.C. Rapaport, Computer Physics Communications **9** (1988) 1.

47. F. Brugè and S.L. Fornili, Computer Physics Communications **60** (1990) 31.

48. M.R.S. Pinches, D.J. Tildesley, and W. Smith, Molecular Simulation **6** (1991) 51.

49. S.Y. Liem, D. Brown, and J.H.R.Clarke, Computer Physics Communications 67 (1991) 261.

50. F. Hedman and A. Laaksonen, International Journal of Modern Physics C **4** (1993) 41.

51. D. Brown, J.H.R. Clarke, M. Okuda, and T. Yamazaki, Computer Physics Communications **74** (1993) 67.

52. K. Esselink, B. Smit, and P.A. Hilbers, Journal of Computational Physics **106** (1993) 101.

53. R.A. McCoy and Y. Deng, The International Journal of High Performance Computing Applications 13 (1999) 16.

54. F.F. Abraham, IEEE Computational Science and Engineering, April-June (1997) 66.

55. P.S. Lomdahl, P. Tamayo, N. Gronbech-Jensen, and D.M. Beazley, Proceedings of Supercomputing 93 (IEEE Computer Society Press, Los Alamitos, CA, 1993) 520.

56. D.M. Beazley and P.S. Lomdahl, Parallel Computing **20** (1994) 173.

57. J. Stadler, R. Mikulla, and H-R. Trebin, International Journal of Modern Physics C **8** (1997) 1131.

58. S.G. Srinivasan, I. Ashok, H. Jônsson, G. Kalonji, and J.Zahorjan, Computer Physics Communications 102 (1997) 28.

59. M. Pütz and A. Kolb, Computer Physics Communications **113** (1998) 145.

60. W.B. Street, D.J. Tildesley, and G. Saville, Molecular Physics **35** (1978) 639.

61. R. Kalia, S.W. de Leeuw, A. Nakano, P. Vashishta, Computer Physics Communications **74** (1993) 316.

62. A. Nakano, P. Vashishta, R. Kalia, Computer Physics Communications **77** (1993) 303.

63. A. Nakano, L. Bi, R. Kalia, and P. Vashishta, Physical Review Letters **71** (1993) 1.

64. A. Nakano, L. Bi, R. Kalia, and P. Vashishta, Physical Review B **49** (1994) 9441.

65. Information about the Cplant machine at Sandia National Laboratories can be found at http://www.cs.sandia.gov/cplant/

66. S.J. Plimpton, unpublished data, http://www.cs.sandia.gov/cplant/apps/LAMMPS/

67. K. Esselink and P.A. Hilbers, Journal of Computational Physics **106** (1993) 108.

68. D. Brown, J.H.R. Clarke, M. Okuda, T. Yamazaki, Computer Physics Communications **83** (1994) 1.

69. G. Eisenhauer and K. Schwan, Journal of Parallel and Distributed Computing **35** (1996) 76.

70. T.W. Clark, R.V. Hanxleden, J.A. McCammon, and L.R. Scott, Proceedings of the Scalable High-Performance Computing Conference (IEEE Computer Society Press, Los Alamitos, CA, 1994) 95.

71. D.P. Hansen and D.J. Evans, Molecular Simulation 13 (1994) 375.

72. R. Bhupathiraju, P.T. Cummings, H.D. Cochran, Molecular Physics 88 (1996) 1665.

73. A. Jabbarzadeh, J.D. Atkinson, and R.I. Tanner, Computer Physics Communications 107 (1997) 123.

74. F. Hedman and A. Laaksonen, Molecular Simulation 14 (1995) 235.

75. P. Ballestrero, P. Baglietto, and C. Ruggiero, Journal of Computational Chemistry **17** (1996) 469.

76. J. Hautman and M.L. Klein, Molecular Physics 75 (1992) 379.

77. M. Surridge, D.J. Tildesley, Y.C. Kong, and D.B. Adolf, Parallel Computing 22 (1996) 1053.

78. Plimpton, S.J., Pollock, R., and Stevens, M., Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing (SIAM, Philadelphia, PA, 1997).

79. L. Greengard and V. Rokhlin, Journal of Computational Physics 73 (1987) 325.

80. J.A. Board Jr., J.W. Causey, J.F. Leathrum Jr., A. Windemuth, and K. Schulten, Chemical Physics Letters **198** (1992) 89.

81. M.T. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L.V. Kalé, R.D. Skeel, and K. Schulten, The International Journal of Supercomputer Applications and High Performance Computing **10** (1996) 251.

82. L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, Journal of Computational Physics **151** (1999) 283.

83. A. Nakano, R.K. Kalia, and P. Vashishta, Computer Physics Communications **83** (1994) 197.

84. P. Vashishta, A. Nakano, R.K. Kalia, and I. Ebbsjö, Journal of Non-Crystalline Solids **182** (1995) 59.

85. P. Vashishta, A. Nakano, R.K. Kalia, and I. Ebbsjö, Materials Science and Engineering **B37** (1996) 56.

86. P. Vashishta, R.K. Kalia, A. Nakano, and W. Jin, International Journal of Thermophysics **17** (1996) 169.

87. A. Nakano, Computer Physics Communications **104** (1997) 59.

88. K. Lim, S. Brunett, M. Iotov, R.B. McClurg, N. Vaidehi, S. Dasgupta, S. Taylor, and W.A. Goddard III, Journal of Computational Chemistry **18** (1997) 501.

89. A. Fijany, T. Çagin, A. Jaramillo-Botero, and W. Goddard III, Advances in Engineering Software **29** (1998) 441.

90. H-Q. Ding, N. Karasawa, and W.A. Goddard III, Journal of Chemical Physics 97 (1992) 4309.

91. J.E. Boillat, F. Brugè, and P.G. Kropf, Journal of Computational Physics **96** (1991) 1.

92. Y. Deng, R.A. McCoy, R.B. Marr, and R.F. Peierls, Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing (SIAM, Philadelphia, PA, 1995) 605.

93. A. Nakano and T. Campbell, Parallel Computing 23 (1997) 1461.

94. S.G. Srinivasasn, I. Ashok, H. Jônsson, G. Kalonji, and J. Zahorjan, Computer Physics Communications **102** (1997) 44.

95. D. Brown, J.H.R. Clarke, M. Okuda, T. Yamazaki, Journal of Chemical Physics **100** (1994) 1684.

96. A.R.C. Raine, D. Fincham, and W. Smith, Computer Physics Communications **55** (1989) 13.

97. J.P. Brunet, J.P. Mesirov, and A. Edelman, Proceedings of Supercomputing 90 (IEEE Computer Society Press, Washington, DC, 1990) 748.

98. H. Heller, H. Grubmuller, and K. Schulten, Molecular Simulation **5** (1990) 133.

99. J. Li, A. Brass, D.J. Ward, and B. Robson, Parallel Computing **14** (1990) 211.

100. A. Windemuth and K. Schulten, Molecular Simulation **5** (1991) 353.

101. A.B. Sinha and K. Schulten, Computer Physics Communications **78** (1994) 265.

102. R. Trobec, F. Merzel, and D. Janezic, Journal of Chemical Information and Computer Sciences **37** (1997) 1055.

103. H. Grubmüller, H. Heller, A. Windemuth, and K. Schulten, Molecular Simulation **6** (1991) 121.

104. R. Trobec, I. Jerebic, and D. Janezic, Parallel Computing **19** (1993) 1029.

105. R. Trobec and D. Janezic, Journal of Chemical Information and Computer Sciences **35** (1995) 100.

106. L.L. Boyer and G.S. Pawley, Journal of Computational Physics 78 (1988) 405.

107. H.L. Nguyen, H. Khanmohammadbaigi, and E. Clementi, Journal of Computational Chemistry **6** (1985) 634.

108. M.A. Shifman, A. Windemuth, K. Schulten, and P.L. Miller, Computers and Biomedical Research **25** (1992) 168.

109. J.P. Brunet, J.P. Mesirov, and A. Edelman, SIAM Journal On Scientific Computing **14** (1993) 1143.

110. H. Schreiber, O. Stenhauser, and P. Schuster, Parallel Computing **18** (1992) 557.

111. S.J. Plimpton, B.A. Hendrickson, and G.S. Heffelfinger, Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing (SIAM, Philadelphia, PA, 1993) 178.

112. S. Plimpton and B. Hendrickson, International Journal of Modern Physics C **5** (1994) 295.

113. S. Plimpton and B. Hendrickson, Journal of Computational Chemistry **17** (1996) 326.

114. V.E. Taylor, R.L. Stevens, and K.E. Arnold, Journal of Parallel and Distributed Computing **45** (1997) 166.

115. R.D. Skeel, Journal of Computational Chemistry **12** (1991) 175.

116. R. Murty and D. Okunbor, Parallel Computing **25** (1999) 217.

117. M.E. Tuckerman, B.J. Berne, and G.J. Martyna, Journal of Chemical Physics 97 (1992) 1990.

118. A.F. Voter, Physical Review Letters, **78** (1997) 3908.

119. A.F. Voter, Physical Review B 57 (1998) 13985.

120. J. Saarinen, K. Kaski, and J. Viitanen, Review of Scientific Instrumentation **60** (1989) 2981.

121. T.J.P. Penna and P.M.C. de Oliveira, Journal of Statistical Physics **61** (1990) 933.

122. W. Schleier, G. Besold, and K. Heinz, Journal of Statistical Physics **66** (1992) 1101.

123. M. Flanigan and P. Tamayo, Physica A **215** (1995) 461.

124. D. J. Adams, Journal of Computational Physics **75** (1988) 138.

125. G.S. Heffelfinger and M.E. Lewitt, Journal Computational Chemistry **17** (1995) 250.

126. R.A. Whiteside, P.G. Hibbard, and N.S. Ostland, Proceedings of the Third International Conference on Distributed Systems (IEEE Computer Society, New York, NY, 1982) 800.

127. S.J. Zara and D. Nicholson, Molecular Simulation **5** (1990) 245.

128. D.M. Jones and J.M. Goodfellow, Journal of Computational Chemistry **14** (1993) 127.

129. D.E. Ulberg and K.E. Gubbins, Molecular Simulation **13** (1994) 205.

130. J.F. Janak and P.C. Pattnaik, Journal of Computational Chemistry 13 (1992) 533.

131. G. Heffelfinger, Proc. 1992 DAGS/PC Symposium, (Dartmouth Institute for Advanced Graduate Studies, Hanover, NH, 1993) 229.

132. B. Mehlig, D.W. Heermann, and B.M. Forrest, Physical Review B **45** (1992) 679.

133. L.D.J.C. Loyens, B. Smit, and K. Esselink, Molecular Physics **86** (1995) 171.

134. F.A. Brodz and J.J. de Pablo, Chemical Engineering Science **49** (1994) 3015.

135. G. Pawley, K. Bowler, R. Kenway, and D. Wallace, Computer Physics Communications **37** (1985) 251.

136. M.A. Johnson, Concurrent Computation and Its Application to the Study of Melting in Two Dimensions (PhD Thesis, California Institute of Technology, 1986).

137. K.S. Shing and S.R. Azadipour, Chemical Physics Letters **190** (1992) 386.

138. A. Chen and C.S. Hirtzel, Supercomputer Applications and High Performance Computing **8** (1994) 54.

139. A. Chen, Computer Simulation for Adsorption and Desorption of Molecules on Solid Surfaces (PhD Thesis, Syracuse University, 1993).

140. A. Chen and C.S. Hirtzel, Molecular Physics **82** (1994) 263.

141. A. Chen and C.S. Hirtzel, Separations Technology **4** (1994) 167.

142. J.J. de Pablo, M. Laso, U.W. Suter, Journal of Chemical Physics **96** (1992) 2395.

143. J.I. Siepmann and D. Frenkel, Molecular Physics **75** (1992) 59.

144. K. Esselink, L.D.J.C. Loyens, and B. Smit, Physical Review E **51** (1995) 1560.

145. D.R. Ripoll and H.A. Scheraga, Biopolymers **27** (1988) 1283.

146. D.R. Ripoll and H.A. Scheraga, Journal of Protein Chemistry **8** (1989) 263.

147. D.R. Ripoll, M.S. Pottle, K.D. Gibson, H.A. Scheraga, A. Liwo, Journal of Computational Chemistry **16** (1995) 1153.

148. D.R. Ripoll and S.J. Thomas, Journal of Supercomputing **6** (1992) 163.

149. T.H. Pierce and L.L. Lynn, Abstracts and Papers of the American Chemical Society **203** (1992) 11-COMP.

150. T.H. Pierce and L.L. Lynn, Abstracts and Papers of the American Chemical Society **203** (1992) 303-POLY.

151. T.H. Pierce and L.L. Lynn, Abstracts and Papers of the American Chemical Society **205** (1993) 13-COMP.

152. L. Lynn, T.H. Pierce, and M.R. Schure, Abstracts and Papers of the American Chemical Society **207** (1994) 108-COMP.

153. A.P. Thompson, D.M. Ford, and G.S. Heffelfinger, Journal of Chemical Physics **109** (1998) 1.

154. G.S. Heffelfinger and D.M. Ford, Molecular Physics **94** (1998) 659.

155. D.M. Ford and G.S. Heffelfinger, Molecular Physics **94** (1998) 673.

156. G.S. Heffelfinger and F. van Swol, Journal of Chemical Physics **100** (1994) 7548.

157. J.M.D. MacElroy, Journal of Chemical Physics **101** (1994) 5274.

158. A.Z. Panagiotopoulos, Molecular Physics **61** (1987) 813.

Figure Captions

Figure 1. The three communication steps of the parallel spatial molecular dynamics algorithm. In the first step, each processor exchanges the coordinates of the atoms in its domain with the processors in the *+x* and *-x* directions. In step 2, each processor exchanges the 3 sets of coordinates it now possesses (it's own, those from the *+x* processor, and those from the *-x* processor) with the processors in the *+y* and *-y* directions. Each processor now possesses 9 sets of coordinates and can carry out step 3, exchanging the 9 sets of coordinates with the processes in the *+z* and *-z* directions.

Figure 2. Plimpton's [66] data for a spatial parallel molecular dynamics algorithm [13] for a 32,000 atom Lennard-Jones system ($\rho^*=0.844$, $r_c=2.5\sigma$) on a traditional massively parallel platform, the Intel Teraflops, and Cplant, a computing system of commodity computing and networking components [65].

Figure 3. Plimpton's [66] data for the spatial parallel molecular dynamics algorithm [13] for Lennard-Jones systems of 32,000 atoms (1 processor) to 16,384,000 atoms (512 processors) on a traditional massively parallel platform, the Intel Teraflops, and Cplant, a computing system of commodity computing and networking components [65].

Figure 4. An eight domain (denoted by the large numbers, 0-7) decomposition of a GCMC simulation domain using the spatial parallel algorithm [125]. In this algorithm, each processor "owns" a domain and carries out simultaneous GCMC moves in its 8 "subdomains" (the regions denoted by the smaller numbers). Each processor works in the same subdomain (sized larger than the interaction potentials cut-off distance) at the same time, enabling simultaneous insertions, deletions, and displacements in different domains (and on different processors).

Figure 5. The standard error as a function of CPU hours for varying number of processors, *P*, of an Intel Paragon for both the spatial parallel GCMC algorithm (solid lines) and the embarrassingly parallel GCMC algorithm (dashed lines) [125].

Figure 6. A schematic of the parallel algorithm for dual control volume grand canonical molecular dynamics (DCV-GCMD) [154, 155] for 32 domains (and to be carried out on 32 processors). The control volumes (unshaded regions) are subdivided into 8 subdomains to enable the use of the spatial parallel Monte Carlo algorithm [125] during the "GCMC phase" (when insertions and deletions are carried out in the control volumes to establish the desired chemical potential in the control volumes). The spatial parallel molecular dynamics algorithm [13] is used throughout the system volume to calculate the forces and new positions during the MD phase (when all of the system's molecules are moved in a molecular dynamics time-step).

Figure 1

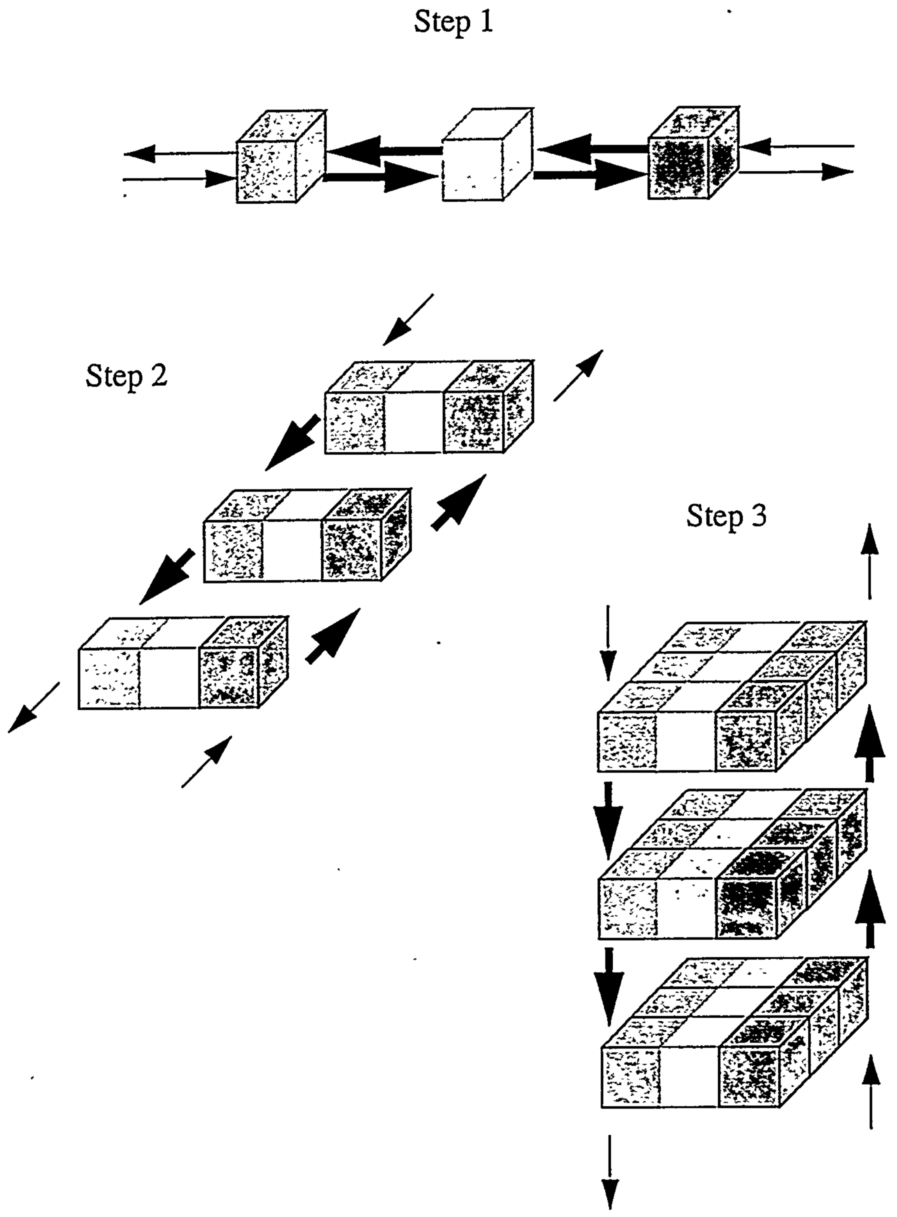


Figure 2

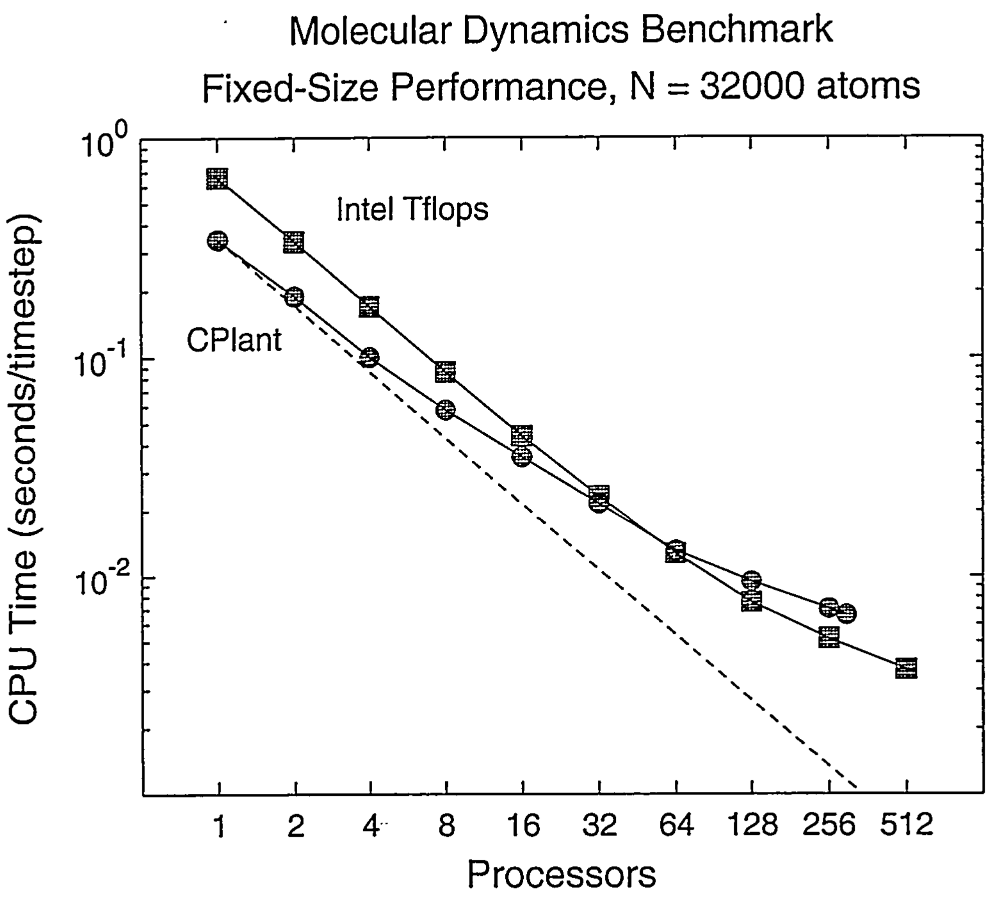
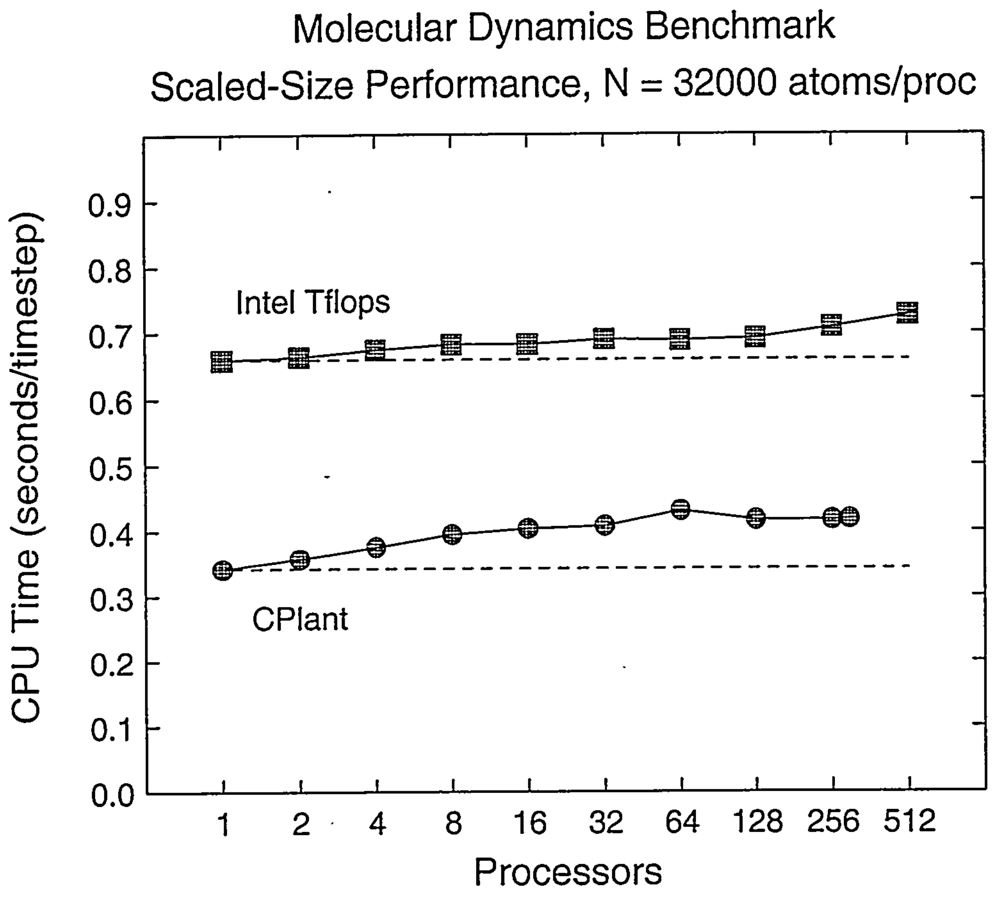
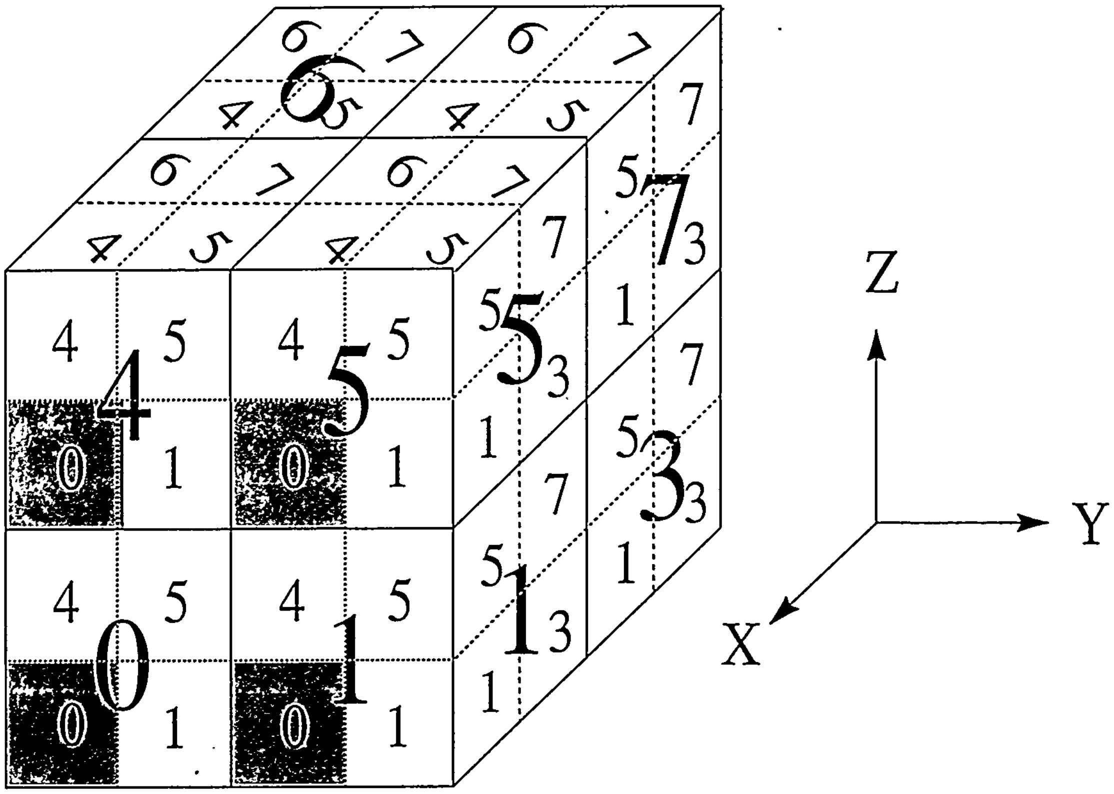

Molecular Dynamics Benchmark
Fixed-Size Performance, N = 32000 atoms
Intel Tflops
CPlant
CPU Time (seconds/timestep)
10^0
10^{-1}
10^{-2}
1
2
4
8
16
32
64
128
256
512
Processors

Figure 3



Molecular Dynamics Benchmark
Scaled-Size Performance, N = 32000 atoms/proc
CPU Time (seconds/timestep)
Intel Tflops
CPlant
Processors
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
4
8
16
32
64
128
256
512

Figure 4



6
7
6
7
6
4
5
4
5
7
6
7
6
7
4
5
4
5
5
7
3
7
4
5
4
5
4
5
5
3
1
1
0
1
0
1
1
7
5
3
3
4
5
4
5
1
5
0
1
3
1
1
0
1
0
1
Z
Y
X

Figure 5

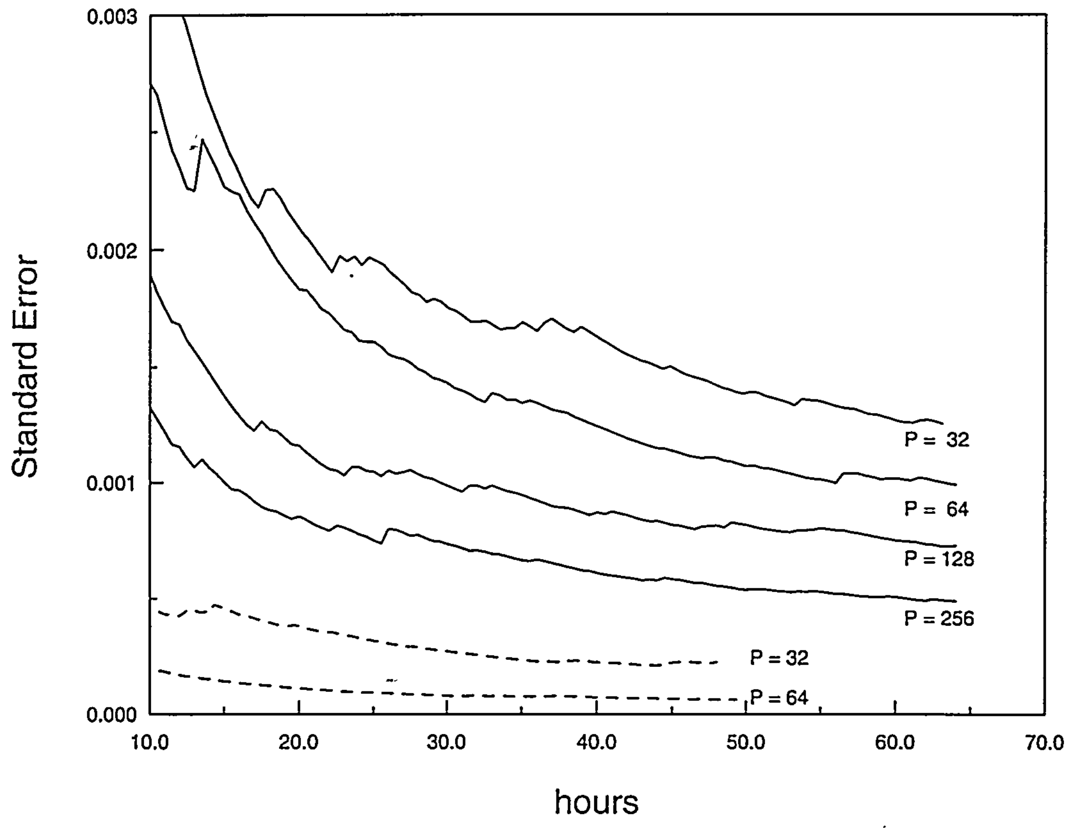
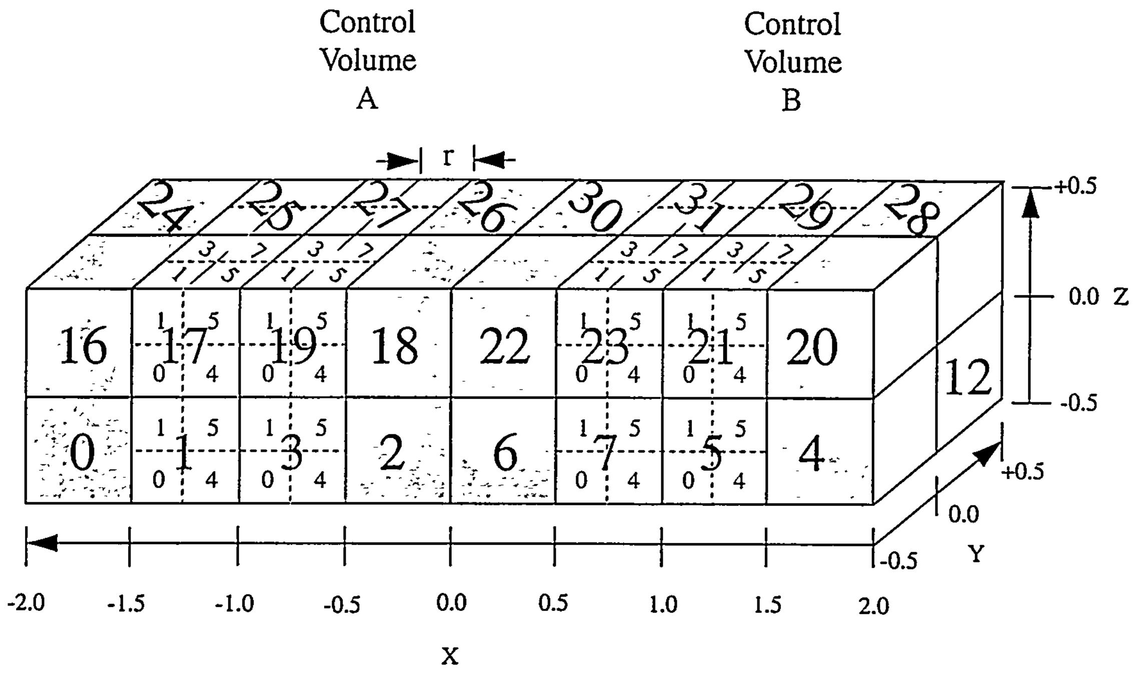

0.003
0.002
0.001
0.000
Standard Error
10.0
20.0
30.0
40.0
50.0
60.0
70.0
hours
P = 32
P = 64
P = 128
P = 256
P = 32
P = 64

Figure 6


Control Volume A
Control Volume B
r
24
25
27
26
30
31
29
28
16
17
19
18
22
23
21
20
12
0
1
3
2
6
7
5
4
+0.5
0.0 Z
-0.5
+0.5
0.0
-0.5
Y
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
X