Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters

Kevin J. Bowers, Edmond Chow, Huafeng Xu, Ron O. Dror, Michael P. Eastwood,
Brent A. Gregersen, John L. Klepeis, Istvan Kolossvary, Mark A. Moraes, Federico D. Sacerdoti,
John K. Salmon, Yibing Shan, David E. Shaw
D. E. Shaw Research, LLC, New York, NY 10036, USA

Although molecular dynamics (MD) simulations of biomolecular systems often run for days to months, many events of great scientific interest and pharmaceutical relevance occur on long time scales that remain beyond reach. We present several new algorithms and implementation techniques that significantly accelerate parallel MD simulations compared with current state-of-the-art codes. These include a novel parallel decomposition method and message-passing techniques that reduce communication requirements, as well as novel communication primitives that further reduce communication time. We have also developed numerical techniques that maintain high accuracy while using single precision computation in order to exploit processor-level vector instructions. These methods are embodied in a newly developed MD code called *Desmond* that achieves unprecedented simulation throughput and parallel scalability on commodity clusters. Our results suggest that Desmond's parallel performance substantially surpasses that of any previously described code. For example, on a standard benchmark, Desmond's performance on a conventional Opteron cluster with 2K processors slightly exceeded the reported performance of IBM's Blue Gene/L machine with 32K processors running its Blue Matter MD code.

1. INTRODUCTION

By modeling the motions of atoms within a molecular system, molecular dynamics (MD) simulations can serve as a computational "microscope" onto phenomena that are difficult to observe experimentally. Such simulations hold great promise in biochemistry and molecular biology, where they allow functional observation of proteins, nucleic acids, membranes, and other building blocks of the cell. Unfortunately, many of the events of greatest biological and pharmaceutical interest take place on time scales that are still beyond the reach of MD simulations on modern computers. Such events as functionally important changes in protein structures, "folding" of proteins to their native three-dimensional structures, and various types of interactions between two proteins or between a protein and a candidate drug molecule often take place on a microsecond to millisecond time scale. An MD simulation of such an event might involve tens of thousands of atoms, representing one or more biological macromolecules surrounded by a *solvent* environment consisting of solvated ions (in some cases) and a large number of water molecules.

Because the vibrational frequencies of these atoms typically limit each simulation time step to a few femtoseconds, simulations of more than a microsecond have thus far proven infeasible on systems with more than about ten thousand atoms [13, 40]. It seems clear that longer simulations will require the application of massive computational parallelism. The scalability of MD codes, however, has historically been limited by formidable inter-processor communication requirements associated with the exchange of atomic positions and inter-atomic force data.

A number of established MD codes, including CHARMM [7], Amber [10], GROMACS [45], and NAMD [24, 34, 35], are widely used in the research community, each supporting somewhat different features and targeting somewhat different goals. Of these, NAMD is regarded as the most scalable and the most efficient on highly parallel runs, although GROMACS typically achieves superior performance on single-processor runs. IBM's recently developed Blue Matter MD code was designed to scale up to the full 128K-processor size of Blue Gene/L [15-17, 19]. Further advances in the parallel execution of MD simulations, however, could have important implications for both scientific research and the development of novel pharmaceutical compounds.

This paper describes a new MD code, named *Desmond*, that achieves unusually high parallel scalability and overall simulation throughput on commodity clusters by using new distributed-memory parallel algorithms. Desmond's efficiency and scalability are due to 1) a novel parallel decomposition method that greatly reduces the requirement for inter-processor communication, 2) an implementation that reduces the number of inter-processor messages, 3) new, highly efficient communication primitives, and 4) the use of short-vector SIMD (single-instruction multiple data) capabilities of the sort included in most modern commodity processors. In addition, Desmond employs new numerical

David E. Shaw is also with the Center for Computational Biology and Bioinformatics, Columbia University, New York, NY 10032. E-mail correspondence: david@deshaw.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 ©2006 IEEE

techniques that allow it to use single precision computation while maintaining high accuracy.

Using Desmond, simulations of several microseconds are now feasible for reasonably large chemical systems running on commodity hardware. Our results suggest that Desmond's performance on highly parallel systems substantially surpasses that of any previously described code. Indeed, using a standard benchmark, we found that Desmond's performance on a conventional Opteron cluster with 2048 processors slightly exceeded that of IBM's Blue Gene/L machine with 32,768 processors running either Blue Matter or NAMD. When we tuned simulation parameters specifically for Desmond, its simulation rate increased by an additional factor of 2.8 without compromising common accuracy measures.

To provide a concrete example of Desmond's performance in an actual biomolecular simulation, on a system with 23,558 atoms (a protein targeted by various cancer drugs called dihydrofolate reductase, surrounded by water), Desmond achieved a simulation rate of over 173 ns/day on 256 dual-processor Opteron nodes (512 processors total) with an InfiniBand interconnect. On a relatively large system with 92,224 atoms (apolipoprotein A1, the main component of what is commonly referred to as "good cholesterol," again surrounded by water), Desmond ran at a speed of 120 ns/day on 1024 nodes of the same cluster.

Several other research groups have deployed significant computational resources either to run a large number of separate MD simulations, each of limited duration [31], or to parallelize the simulation of extremely large biochemical systems [29, 46]. While such simulations may be of considerable value in certain contexts, our own research focuses on the many applications that require the simulation of a single, very long molecular trajectory for a biomolecular system with tens or hundreds of thousands of atoms. The ability to perform these computational experiments could lead to important advances in the biomedical sciences, but the massively parallel execution of a single, very long MD simulation on a system in this size range requires far more inter-processor communication, and thus poses far greater scalability challenges, than either the execution of many short simulations or the simulation of an extremely large biochemical system.

In the remainder of this section, we briefly review the essential aspects of molecular dynamics simulation. In Section 2, we describe parallelization strategies, including a new parallel decomposition method called the *midpoint method* that is used to parallelize range-limited interactions among both covalently bonded and nonbonded atoms, and a method for parallelizing the three-dimensional Fast Fourier Transform (FFT) used as part of the procedure that handles long-range electrostatic interactions. Section 3 describes techniques for shared memory parallelism within each node using multiple threads and for parallelism within each processor using SIMD instructions. Section 4 discusses new numerical techniques that achieve high accuracy and increased simulation throughput. Section 5 discusses novel communication primitives that exploit the communication patterns used in Desmond to deliver better performance than standard message-passing protocols. In Section 6, we present performance measurements for Desmond on two biochemical systems that have been used to benchmark other MD codes, including strong scaling results for up to 2048 processors.

Molecular Dynamics Simulations. In an MD simulation, the positions and velocities of particles corresponding to atoms evolve according to the laws of classical physics. In this paper, we will assume a one-to-one correspondence between particles and atoms for expository simplicity, although Desmond is also capable of representing a group of atoms by one particle or a single atom by several particles. Desmond is designed for *explicit solvent* simulations, where water molecules (along with any ions that may be present within this solvent environment) are represented at the atomic level rather than through a less accurate but potentially cheaper continuum electrostatics model.

Each time step of an MD simulation involves 1) computing forces on each particle (*force computation*) and 2) using these forces to compute updated positions and velocities on each particle by numerically integrating Newton's laws of motion (*integration*). Most of the computational load lies in the force computation.

The force computation uses a model called a *molecular mechanics force field* (or simply, *force field*), which specifies the potential energy of the system as a function of the atomic coordinates. (The force on a particle is the derivative of this potential energy with respect to the position of that particle.) Although classical MD simulation is inherently an approximation, it is dramatically faster than a direct solution to the full set of quantum mechanical equations. Several decades of work have gone into the development of biomolecular force fields through fitting models to experimental and quantum data.

Desmond is compatible with common biomolecular force fields, including CHARMM [26], AMBER [23], OPLS-AA [22], GROMOS [39], and MMFF [20]. All of these express the total potential energy (E) of a chemical system as a sum of the form

$$E = E_{bonded} + E_{es} + E_{vdW}.$$

E_{bonded} is the sum of several *bonded terms* which depend on the covalent bond structure of the molecules. These include bond length terms, involving two particles connected by a bond; bond angle terms, involving three particles, two of which are bonded to a third; and dihedral angle (torsional) terms, involving four particles connected by three bonds.

E_{es} and E_{vdW}—the *electrostatic* and *van der Waals terms*, respectively—are known as *nonbonded terms* because they include interactions between essentially all pairs of particles in the system. They therefore represent a much larger computational burden than the bonded terms. Van der Waals forces fall off sufficiently quickly with distance that they can typically be neglected for pairs of particles separated by more than some cutoff radius R, typically chosen between 9 and 12 Å. An ever-mounting body of evidence shows that neglecting electrostatic interactions beyond a cutoff is inadequate for explicit solvent simulations [8, 27, 30, 33]; electrostatic forces thus are typically computed by one of several efficient, approximate methods that account for long-range interactions without requiring the explicit interaction of all pairs of particles. Desmond supports particle mesh Ewald (PME) [12]

and *k*-space Gaussian split Ewald (*k*-GSE) [41], both of which use the fast Fourier transform (FFT) to compute electrostatic potential on a mesh given a charge distribution on that mesh. Both these methods require that modified pairwise electrostatic interactions be computed explicitly within the cutoff radius. They also require that charge be mapped from particles to nearby mesh points before the FFT computation (*charge spreading*) and that forces on particles be calculated from potentials at nearby mesh points after the FFT computation (*force interpolation*). PME and *k*-GSE assume periodic boundary conditions, which are used by default in Desmond.

One can lengthen the simulation time step somewhat by applying *constraints* that eliminate the fastest vibrational motions. When applying constraints, we typically fix the lengths of bonds to all hydrogen atoms, as well as angles between the bonds in water atoms, by computing a correction to atom positions and velocities after integration.

2. Parallelization of Molecular Dynamics

A distributed memory parallel program consists of several processes using a number of processors that can communicate via a network. Each process usually runs on one processor, but it is also common for multiple processes to be run on one processor, or for each process to be run on multiple processors that share memory. In this section, we discuss parallelization from the point of view of processes, independent of their relation to processors. In Section 3.1, we discuss how we actually run Desmond on an Opteron cluster that has two processors per node.

In Desmond, as in many other MD codes designed for scalability [15-17, 19, 34-36], each process takes responsibility for updating positions of particles that fall in a certain region of space. Desmond assumes that the region to be simulated (the *global cell*) is a parallelepiped, divided into a regular grid of small parallelepipeds called *boxes*. For expository simplicity, we will assume in this paper that both the global cell and the boxes are rectangular parallelepipeds, although Desmond also supports non-rectangular global cells. Each process updates the coordinates of the particles in one box, referred to as the *home box* of that process and of those particles. In the interest of simplicity, we will refer interchangeably to a process and its home box.

The FFT operation used to evaluate long-range electrostatics requires communication between boxes that are distant from one another. All other Desmond operations that require inter-process communication—explicit pairwise computation of nonbonded forces, computation of bonded forces, charge spreading, force interpolation, computation of constraints, and particle migration—involve only local communication between nearby boxes. Section 2.1 describes our parallelization strategy for computations involving local communication and Section 2.2 describes our parallelization strategy for the FFT.

2.1 The Midpoint Method

We first describe Desmond's approach to parallelizing the evaluation of nonbonded (electrostatic and van der Waals) forces between all pairs of atoms separated by less than the cutoff radius *R*. Several papers survey traditional methods for parallelizing the explicit evaluation of interactions between pairs of particles separated by less than some maximum distance [21, 36-38]. Plimpton [36] categorized these methods as atom, force, and spatial decomposition methods. Unlike atom and force decomposition methods, spatial decomposition methods offer the desirable property that the amount of data to be transferred into and out of each process (the method's *communication bandwidth*) decreases as the interaction radius decreases. In traditional spatial decomposition methods, as in Desmond's parallelization strategy, each process takes responsibility for updating positions of particles falling in one region of space. In traditional spatial decomposition methods, the process that computes the interaction between two particles is always a process on which one or both particles reside.

A number of recently introduced methods for parallelizing range-limited particle interactions require significantly less communication bandwidth than traditional parallelization methods [4-6, 43, 44]. These novel methods also employ a spatial assignment of particles to processes, but unlike traditional spatial decomposition methods, they sometimes compute an interaction between two particles in a process on which neither particle resides [6]. We refer to such techniques as *neutral territory methods* [6, 43].

Two distinct neutral territory methods were developed independently by Snir [44] and Shaw [43], who point out that these methods might be viewed as hybrids between traditional spatial decompositions and the force decompositions introduced by Plimpton and Hendrickson [38]. We later generalized these methods [6] and introduced several new neutral territory methods, including the *midpoint method* [5]. The midpoint method requires less communication bandwidth than traditional spatial decomposition methods, particularly at moderate or high levels of parallelism. While certain other neutral territory methods require less communication bandwidth than the midpoint method for pairwise interactions parallelized over a sufficiently large number of processes, the midpoint method offers several significant advantages for an MD code [5]. It applies not only to pairwise interactions, but also to interactions involving sets of three or more particles, and it can therefore be used to parallelize the evaluation of bonded force terms. Moreover, it allows nearly all the computations that require local communication to rely on the same data that is communicated for the evaluation of pairwise nonbonded forces. It also typically incurs a smaller penalty due to communication latency than other methods. A related method was recently developed independently for Blue Matter [17, 19].

When applied to pairwise interactions, the midpoint method specifies that two particles interact on a particular box if and only if the midpoint of the segment connecting them falls within the region of space associated with that box. We refer

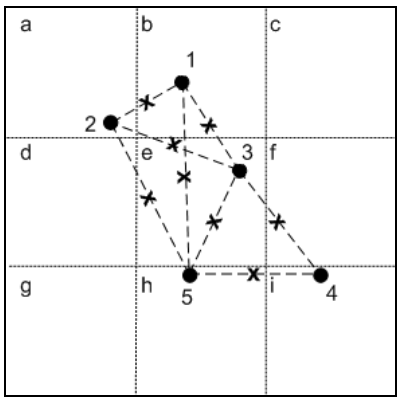

Figure 1. Assignment of particle pairs to interaction boxes in the midpoint method. In this figure, the boxes are square with side length b, and $R = 1.5b$. Each pair of particles separated by a distance less than R is connected by a dashed line segment, with the "x" at its center indicating the box which will compute the interaction of that pair.

to the box in which a set of particles interact as their *interaction box*. Figure 1 illustrates the assignment of particle pairs to interaction boxes implied by the midpoint method. Two particles that lie in the same box necessarily interact in that box, but particles that lie in different boxes may interact either in the box in which one of them resides (e.g., particles 2 and 3) or in a box in which neither resides (e.g., particles 1 and 5 or particles 3 and 4).

In the midpoint method, the volume of space from which a given process must "import" particle data that ordinarily resides within other processes (its *import region*) includes only points within a distance $R/2$ of its home box, because if the distance between two particles is less than R and one of them lies more than a distance $R/2$ from the home box, their midpoint must lie outside the home box. This import region is shown in Figure 2(a) for a two-dimensional system.

For comparison, Figure 2(b) shows the import region of a particular traditional spatial decomposition method in which the box that interacts two particles is always the home box of one or both particles. In this method, the particles interact within the home box of the particle with the smaller x-coordinate, unless the home boxes of the two particles are in the same vertical column, in which case the particles interact within the home box of the particle with the smaller y-coordinate. The import region includes half the space within a distance R of the home box. This method is the 2-D analog of the *HS method* defined in our recent publications [6, 43].

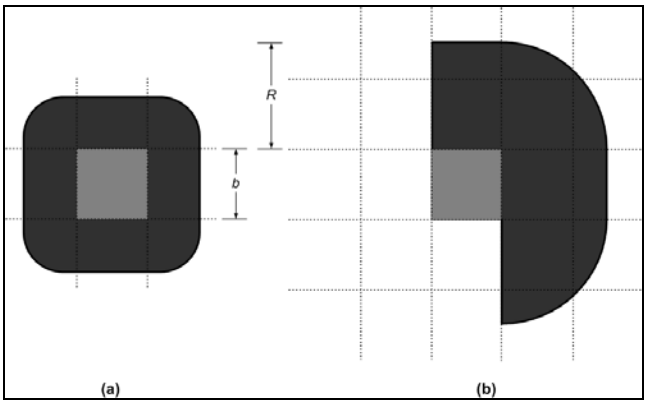

Figure 2. Import regions of (a) the 2-D midpoint method and (b) the 2-D analog of the HS method, where $R = 1.5b$. In each case, the interaction box is light gray and the import region is dark gray.

When the midpoint or HS methods are used for an MD simulation, or any other application that requires computation of the total force on each particle, each interaction box must "export" a force contribution to each of the particles in its import region after it has computed the interactions assigned to it. A method in which each process computes all force contributions for each particle in its home box would avoid the need for such force export, but it would have twice the import volume of the HS method and would require that each interaction between particles in different boxes be computed twice.

Communication Volume. Assuming uniform particle density, the amount of particle data that must be transferred into each process during particle import and out of each process during force export is proportional to the volume of the import region. We therefore use the volume of the import region (the *import volume*) as a measure of communication bandwidth requirements of a parallelization method. Assuming cubical boxes of side length b, we can express the import volumes of the 3-D HS and midpoint methods ($V_{import,HS}$ and $V_{import,midpoint}$, respectively) in terms of $\alpha_R = R/b$ as:

$$V_{import,HS} = b^3 (2/3\ \pi\alpha_R^3 + 3/2\ \pi\alpha_R^2 + 3\alpha_R)$$
$$V_{import,midpoint} = b^3 (1/6\ \pi\alpha_R^3 + 3/4\ \pi\alpha_R^2 + 3\alpha_R).$$

A large value of α_R implies a high degree of parallelism, as b is determined by the number of processes used for simulation as well as the size of the global cell. The import volume of the midpoint method is always smaller than that of the HS method, with the difference growing in both relative and absolute terms as α_R grows. For a more detailed comparison of the import volumes of various parallelization methods, see [5] and [6].

Parallelization of Other Calculations. The midpoint method also applies to interactions that involve sets of three or more particles: the interaction between a set of m particles is computed on the box that contains their midpoint, defined as the center of the smallest sphere that contains all m particles. Desmond uses the midpoint method to parallelize computation of the bonded terms, which typically involve two, three, or four particles. Each of these terms is evaluated on the box containing an easily computed, approximate midpoint of the particles involved. For parameters associated with typical biomolecular force fields, this requires no additional communication because all particle positions needed for the computation of each bonded term are already included in the midpoint method import region associated with the pairwise nonbonded computations [5].

Similarly, no additional communication is typically required for the charge spreading or force interpolation operations associated with PME and k-GSE because the particles to be communicated already lie in the midpoint method's import region [5]. The same holds for the constraint calculations performed in Desmond. Under the HS method, these operations would require additional communication.

Particle Migration. In an MD simulation parallelized via the midpoint method, particles migrate from one box to another as they move. In principle, particle migration could be combined with the particle import step of the midpoint method. In Desmond, we have chosen to perform particle migration in a separate round of communication following particle position import, to make communication protocols simpler and more efficient. Desmond also exploits this latter round of communication to transfer responsibility for computation of bonded terms and constraint terms between processors. Desmond always performs calculations with current position and velocity data. However, because the distance by which a particle typically moves in a time step is several orders of magnitude smaller than the cutoff radius, Desmond can avoid performing migration at every time step by slightly expanding the import region. In particular, expanding the import region by a few tenths of an Angstrom in each direction ensures that migration need be performed only every few time steps, without affecting the accuracy of the force terms computed. While this increases the communication bandwidth required for particle import and force export, it typically reduces overall communication requirements.

Number of Messages. The current version of Desmond requires that all box side lengths are greater than $R/2$, implying that each box only communicates with its 26 nearest neighbors under the midpoint method. The HS method, on the other hand, requires communication between more distant boxes whenever any box side length is less than R [5].

Instead of sending 26 individual messages from each process simultaneously, Desmond sends only 6 messages, one in each cardinal direction, using a staged approach similar to that introduced by Plimpton in the context of a traditional spatial decomposition method [36]. All boxes first send messages in the $+x$ and $-x$ directions, followed by messages in the $+y$ and $-y$ directions, followed by messages in the $+z$ and $-z$ directions. To send data to a $(+x, +y, +z)$ direction neighbor, that data is sent first in the $+x$ direction (along with all other data being sent in that direction), then in the $+y$ direction (along with other data), and finally in the $+z$ direction (along with other data). Other neighbors are handled similarly. The aggregated messages generally remain short: due to spatial locality, much of the data sent in the $+x$ direction to the $(+x, +y, +z)$ direction neighbor, for example, will also be needed by the $+x$ direction neighbor.

The question remains as to whether sending 6 messages in 3 stages is faster than sending 26 individual messages simultaneously. In many communication networks, the latency of communication calls may be somewhat overlapped, that is, it is possible to send k messages with nonblocking communications in time less than the product of k with the time to send a single message. In tests using 4096-byte messages on an InfiniBand network, we found that sending and receiving 2 simultaneous messages took 33.7 μs. Hence the staged approach would take approximately $3 \times 33.7 = 101.1$ μs, significantly less than the 328.8 μs required to send and receive 26 simultaneous messages (for comparison, sending and receiving a single message took 22.4 μs).

Load Balance. The density of atoms in biomolecular systems is highly uniform, at approximately 0.1 atoms/Å^3. Load balance is therefore not a significant issue for explicit solvent MD simulations at moderate levels of parallelism, as illustrated by our performance measurements in Section 6. At very high levels of parallelism, statistical fluctuations, differences between the bonded terms in solvent molecules (e.g., water) and in solute molecules (e.g., proteins and lipids), and the slightly lower density of solute molecules cause some load imbalance. Variants of the midpoint method [5, 17, 19] allow for improved load balance, but we have found these to have only a minor performance impact in Desmond due to additional communication and bookkeeping overhead.

Parallelization Engine. To maximize the efficiency of communication and computation associated with the midpoint method, an implementation should ensure that:

- The messages to be sent are as small as possible, including only data that needs to be updated in other processes, and no more messages are sent than necessary.
- Assembly of these messages requires minimal computation.
- Each process stores data in such a way that it can be accessed efficiently when needed for computation.

To achieve these goals, we developed a portable library called the *particle simulation parallelization engine* that manages ordering of data in memory and assembly of messages sent to other processes. This engine handles not only particle records, but also group records that specify properties of sets of particles (for example, each bond and constraint term has an associated group record that specifies which particles participate in that term, along with the corresponding force field parameters). Desmond's parallelization engine ensures that:

- The six messages sent by each process, one in each of the cardinal directions, include only those records required by the midpoint method, with no overhead. Records are sent only to processes that need a copy but do not already have one. Only particles that fall strictly within a box's import or export region are communicated. In particular, the corners and edges of this region are properly rounded, as we have found the rounding computation to be cheaper than the additional overheads implicit in not rounding. Only a single copy of a record is communicated across any communication link, even if that record needs to be sent to multiple destinations. Between migrations, the engine performs distributed memory update operations on subsets of the particle properties rather than communicating the entire particle record. The update messages themselves have no formatting overhead; the sender and receiver both know how to pack and unpack these messages.
- Each process aggregates the six outgoing messages in no more than a single pass through its particle memory.
- A process can find any particle record present in an amount of time independent of the number of particle records stored in that process. Such particle lookups can be indexed by spatial location as well as local or global particle identifier.

2.2 FFT Parallelization

The PME and *k*-GSE methods for efficient long-range electrostatics both require computation of a 3-D FFT on a mesh, followed by a multiplication of the transformed data with an *influence function* that is independent of the data, followed by an inverse 3-D FFT. The computational structure of the inverse FFT is similar to that of the forward FFT. We introduce a parallelization technique for the multidimensional FFTs that minimizes the required number of inter-process messages, assuming that the mesh is distributed spatially across boxes and their corresponding processes.

Let $N_x \times N_y \times N_z$ denote the dimensions of the 3-D mesh superimposed on the global cell; to simplify the calculation of the FFT, we assume that N_x, N_y, and N_z are all powers of 2. We assume that these mesh points are distributed spatially across $p_x \times p_y \times p_z$ boxes, where p_x, p_y, and p_z are also powers of 2. We denote by $n_x \times n_y \times n_z$ the size of the mesh on each box. For biomolecular simulations, N_x, N_y, and N_z are typically between 32 and 128. Parallel FFT computation would not normally be necessary for this small amount of data, except for the fact that the data is distributed. The cost for redistributing the data turns out to govern how best to parallelize small distributed FFT calculations.

A 3-D FFT is computed by successively calculating sets of 1-D FFTs in each of the three dimensions. Each column of boxes in the *x*-dimension contains $N_x n_y n_z$ data points, requiring $n_y n_z$ separate 1-D transforms. The data for each 1-D transform is distributed across all p_x processes in the column. One parallelization approach, suitable for torus networks such as that of Blue Gene/L, is to use all-to-all communication among these p_x processes to redistribute the data such that each process computes $n_y n_z / p_x$ transforms [14].

For commodity networks, however, all-to-all communication among the p_x processes is slow and generally requires every process to send a message directly to every other process. Our approach is not to redistribute the data and instead to perform directly the butterfly communications required in the FFT calculations, as shown in Figure 3. In the *x*-dimension transform considered above, the FFT will require $log2(N_x)$ stages, of which $log2(p_x)$ will require inter-process data transfer. (The communication pattern is greatly simplified because N_x and p_x are powers of 2.) In each stage that requires inter-process communication, each process exchanges data with one other process.

FFT algorithms can be categorized as *decimation in time* or *decimation in frequency*. In a decimation in frequency transform, the first $log2(p_x)$ stages (of a 1-D transform) require inter-process communication and the remaining $log2(n_x)$ stages use completely local data. In a decimation in time transform, the first $log2(n_x)$ stages are local while the remaining $log2(p_x)$ stages require communication. Our forward transforms use decimation in frequency. After three sets of forward transforms, one along each dimension, the ordering of the transformed data across processes is bit-reversed, but computations in the Fourier domain (multiplication by the influence function) can be applied with the data in this order at no extra expense. Our inverse transforms use decimation in time, which brings the result back into the original spatially distributed ordering.

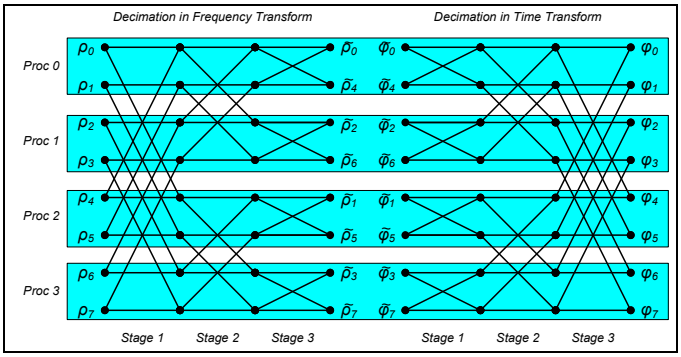

Figure 3. Computation pattern for the 1-D forward decimation in frequency FFT and 1-D inverse decimation in time FFT for 8 points distributed across 4 processes. The ρ variables represent charge and the φ variables represent potential. The computation of φ from ρ involves multiplication by an influence function, performed in bit-reversed order. The links that cross process boundaries imply inter-process communication. In decimation in frequency, the first two stages require communication; the last stage is local. In decimation in time, the first stage is local; the last two stages require communication.

When parallelizing across 512 processes in an 8×8×8 configuration, the forward and inverse 3-D FFTs each require 9 messages to be sent by each process: 3 for each of 3 dimensions. For relatively small meshes on commodity networks, the number of messages to be sent and their associated latency is the primary determinant of FFT communication time. FFTs involving large amounts of data are typically parallelized in other ways (e.g., FFTW [18]), because in those cases, message bandwidth begins to dominate latency.

3. Intra-Node Parallelization

3.1 Shared Memory Parallelism

Many modern distributed parallel systems have multiple processors per node, typically in a shared memory configuration. This leads to a choice of parallelization methods at the node level. If a node has *k* processors, one option is to split the box assigned to that node into *k* smaller boxes, with *k* corresponding processes. Each processor takes responsibility for one of the smaller boxes and the corresponding process, using the same protocols to communicate with other processors on the same node as with processors on other nodes. An alternative is to run a single process per node with *k* threads, called a *hybrid* approach, which may be implemented using OpenMP or POSIX threads (pthreads). The various threads may be running the same parts of the code simultaneously, or they may be running different parts of the code.

In various studies involving other applications [9, 11], assigning a separate process to each processor has been found to be more efficient than the hybrid approach, due to poorer cache behavior and serialization of communication and other "critical sections" in the hybrid approach. However, implementations of message-passing protocols often limit the maximum number of processes they can support, which forces

the use of a hybrid approach in order to utilize very large numbers of processors. Desmond's requirement that box side lengths be no less than $R/2$ has a similar effect, as it limits the number of boxes that can be used for small chemical systems. To allow high degrees of parallelism for small chemical systems, Desmond uses pthreads to implement a capability for each process to divide its work among multiple threads and processors. This implementation is highly portable across platforms. For a fixed number of processors, we find that the use of multiple threads per process leads to a small degradation in performance (see Section 6).

3.2 Data Parallelism

Many microprocessors provide short-vector SIMD (single instruction, multiple data) extensions to accelerate multimedia tasks. For example, some Intel and AMD processors provide SSE (streaming SIMD extensions) while some PowerPC-based processors provide AltiVec extensions. In both cases, these extensions include the ability to simultaneously perform four 32-bit single precision floating point operations. Unfortunately, these operations require that memory accesses are 16-byte aligned, making it difficult or impossible for a compiler to generate efficient SIMD code automatically.

To exploit short vector SIMD accelerations without resorting to hand-coding SIMD instructions, we developed a programming interface that allows users to write code that will compile anywhere but that will be converted to near optimal inline assembly on platforms that support short-vector SIMD. Desmond uses this interface in nearly all its compute-intensive loops. The interface is a set of C++ classes utilizing operator overloading, making SIMD code look very much like plain C code. At present, we support conversion of code written under this programming model to SSE-accelerated data parallel code as well as portable scalar code; one can easily add support for the short-vector SIMD instructions available in other architectures. Our approach might be viewed as an elaboration of the V4 programming model used in the relativistic 3-D particle-in-cell simulation code V-PIC [3].

SIMD parallelism is used either "vertically" or "horizontally." In vertical SIMD, four similar quantities are computed simultaneously; for example, these might comprise forces due to four particles on a given particle, or four local 1-D FFTs. In horizontal SIMD, four arithmetic operations are performed simultaneously on a single particle; for example, one might compute its energy and the x-, y-, and z-components of its force.

The use of SIMD gives Desmond very high single-processor performance. We estimate that the kernel that performs the cutoff-limited nonbonded force calculations is operating at 1.7 single precision Gflops/s per processor (peak is 9.6 single precision Gflops/s, assuming four per cycle). This estimate is based on a count of 56 (single precision) flops per pair of interacting particles (there are also 19 memory operations and 35 other operations per pair of particles, including integer arithmetic and address generation).

4. NUMERICAL TECHNIQUES

Desmond can be configured to execute the great majority of its arithmetic instructions in either single precision or double precision. Single precision operation reduces memory and network bandwidth requirements by about a factor of two and allows SIMD extensions to be used, giving much higher performance than double precision. In this section, we briefly describe some numerical techniques that allow Desmond to operate accurately and efficiently in single precision.

An exact MD simulation would conserve energy exactly. Errors in the simulation generally lead to an increase in the overall energy of the simulated system with time, a phenomenon known as *energy drift*. The rate of energy drift is often used as one measure of the accuracy of an MD simulation, with a lower energy drift rate corresponding to a more accurate simulation. We express energy drift in terms of that rate at which the temperature would change if all the excess energy were converted to heat. Desmond can be configured such that its single precision simulations achieve energy drift levels below 1 K per microsecond of simulated time—better than most double precision codes.

Bitwise Time Reversibility. In principle, an MD simulation should be time reversible, because the classical equations of motion are time reversible. For example, if one runs a simulation for a thousand time steps, negates the instantaneous velocities for all particles, and runs for another thousand time steps, one should exactly recover the initial particle positions. Most particle simulation codes fail to achieve reversibility for two reasons. First, roundoff error during integration leads to a loss of state information; particle positions and velocities at one time step cannot be exactly reconstructed from those at the next time step. Second, lack of associativity of floating point summation leads to a situation where computed forces can depend on various factors that affect order of summation, such that these forces are not uniquely defined by particle positions and force field parameters.

Desmond, on the other hand, preserves exact bitwise reversibility for simulations that do not use constraints and in which particle migration is performed at every time step, under some MD integration schemes. Desmond avoids loss of information during integration by performing updates to particle coordinates using fixed point arithmetic (the details of how this is done without significant performance impact are beyond the scope of this paper). Desmond avoids problems due to non-associativity by maintaining a consistent ordering of the particles and computations such that computed forces are unique functions of the particle positions and force field parameters. We are not aware of any other large floating point code, scalar or parallel, that ensures exact reversibility for non-trivial runs. Reversibility helps ensure that Desmond will accurately model thermodynamic relationships that depend on detailed balance and also results in very low energy drift. While Desmond simulations that employ constraints or that perform particle migration infrequently are not strictly bitwise reversible, we have found that the numerical techniques mentioned here are still beneficial in those cases, resulting, for example, in significantly lower energy drift. In

addition, we note that Desmond simulations are always deterministic.

Constraint Stabilization. Constraint calculations can introduce correlations between roundoff errors in a particle's position and velocity. These correlations can cause severe energy drift in single precision MD codes. We have developed new position constraint algorithms that eliminate these correlations, to be described in a future paper.

Normalized Local Coordinate Representation. Desmond represents the positions of particles in each box in a local coordinate system that is normalized to that box, i.e., each coordinate ranges from –0.5 to +0.5. This gives extra precision compared to using absolute global coordinates, particularly in parallel runs. Another benefit of using a normalized coordinate representation is that pressure control, which alters the dimensions of the boxes, can be implemented to be time-reversible. Further, local coordinate representation simplifies the handling of periodic boundary conditions.

Interpolation Schemes. Desmond uses a piecewise polynomial approximation to compute explicit pairwise electrostatic interactions between nearby atoms, as the modified functional form required by PME and *k*-GSE is expensive to compute directly. Instead of computing the force between two particles as a function of the distance r between them, however, Desmond computes it as a function of a transformed variable of the form ar^2+b. This has the benefit that energy and force can be computed simultaneously from the same set of polynomial coefficients without using square roots or reciprocal square roots. This ensures that forces are strictly the gradient of some potential and eliminates expensive operations in Desmond's nonbonded inner loop.

5. Communication Primitives Using RDMA

Modern high-performance interconnects use some form of remote direct memory access (RDMA) to transfer data in a distributed memory system without interrupting the CPU. To use RDMA, memory regions participating in data transfer must be registered to prevent these memory regions from being swapped out and to provide the network interface card with the virtual to physical address mapping. Memory registrations, however, require costly OS operations. To avoid this overhead for short messages, a message-passing library would typically pre-allocate and pre-register several small buffers; a short message is sent by first copying it to a pre-registered buffer, then sending it via RDMA, and finally copying it from the pre-registered buffer on the receiver side. To send a long message, a three-step protocol is used which avoids memory copies at the cost of memory registrations: 1) the sender registers the memory of the data to be sent and first sends a short request message, 2) the receiver registers memory and replies with a destination address, 3) the sender sends the data directly via RDMA to the destination address.

We have developed a novel set of RDMA communication primitives for all message lengths that avoids the memory copies, memory registrations, and multiple-step protocols of the above techniques. This results in communication primitives that have much lower overhead for sending a message. The primitives use a one-step protocol, sending RDMA messages without attempting to first synchronize with the receiver. For this to work, one must guarantee that for each message being sent, there is a buffer available on the receive side. This condition is easily satisfied for common communication patterns used in many parallel codes, including Desmond, which we call *iterative exchange patterns*. These patterns are composed of iterations where a process exchanges messages with other processes. When a process receives a message, this implies that a message sent by that process in an earlier iteration must have been received; thus the receive buffer used in that earlier iteration is available. Figure 4 shows this in detail.

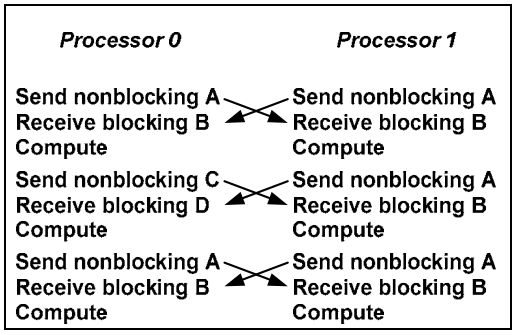

Figure 4. Iterative exchange pattern for two processes and three communication steps. On processor 0, two send buffers are labeled A and C; two receive buffers are labeled B and D. On processor 1, the corresponding buffers are labeled A', C', B', and D'. When processor 0 receives a message into buffer D, it implies that the communication from buffer A to buffer B' has completed and that the send buffer A and the receive buffer B' are available. The next communication step then uses these buffers without needing to synchronize with the receiver. This communication pattern can be regarded as two simultaneous ping-pongs.

At initialization time, the application specifies a connection between each pair of processes that will communicate, as well as the maximum buffer sizes required. Two sets of buffers (two receive and two send buffers) for each connection are allocated and registered at this time. The application uses these buffers directly, alternating between the two sets of buffers as shown in the figure.

Our implementation of these communication primitives is for InfiniBand; it uses the Verbs interface provided by Mellanox Technologies [28]. We use RDMA write operations, which are faster than RDMA read operations on current hardware [42]. We must poll on the message explicitly to know when a message has arrived, since it is inefficient for RDMA writes to generate a completion signal on the receive side. We use the two-stage polling technique described in [25]. The technique requires that send and receive buffers are not modified except for writing data to be sent into the send buffer.

We compare these primitives to an implementation of MPI called MVAPICH [25] which supports InfiniBand and its RDMA features. We tuned MVAPICH to optimize application performance. In particular, we use 12 KB pre-registered buffers (called VBUFs) and 8 VBUFs per connection (there is a connection between every pair of MPI processes in MVAPICH 0.9.5, which limits VBUF memory). Many messages in Desmond, however, are over 12 KB, which

means the slower three-step protocol is used for these messages. A slower protocol is also used for short messages when MVAPICH's credit system detects that VBUFs on the receiver side have been exhausted.

We benchmarked our new communication primitives on an iterative exchange between processes arranged in a ring topology, where processes exchange messages with their left and right neighbors. For a ring of 1024 processes (on 1024 nodes of an InfiniBand cluster) the average time for one iteration was 111.5 microseconds on MVAPICH, and 12.5 microseconds with our new primitives—almost an order of magnitude improvement. The performance of the new primitives is almost insensitive to number of processes, while the performance of MVAPICH degrades with number of processes. Compared to MPI, the new primitives trade generality for higher performance.

6. Performance Tests

Performance tests with Desmond were carried out on an in-house 1056 node (2112 processor) InfiniBand cluster. Each node is a Sun Fire V20z server with two 2.4 GHz AMD Opteron Model 250 (single-core) processors. The operating system is 64-bit Linux (kernel 2.6.9 from the Rocks 4.0 cluster distribution [32]). The InfiniBand network is constructed with Cisco SFS 7000 leaf switches, Cisco SFS 7008 core switches, and Cisco InfiniBand PCI-X Host adapters. Timings were collected by accessing the Opteron processor cycle counter, which gives very fine-grained timing results.

We present performance results for two chemical systems that are among the most common benchmark systems used for MD codes. These are the ApoA1 (apolipoprotein A1) system with 92,224 atoms (in a global cell of approximately 109×109×78 $Å^3$) and the DHFR (dihydrofolate reductase) system, also known as the Joint Amber-CHARMM benchmark, with 23,558 atoms (approximately 62×62×62 $Å^3$). For each system, we use two sets of simulation parameters. First, we use *benchmark parameters*, which are the parameters specified by the benchmarks. (For ApoA1, Desmond uses a PME mesh that is finer than specified by the benchmark, due to the powers-of-2 restriction in Desmond's FFT calculation.) We also use *production parameters*, which were tuned for Desmond to achieve a higher simulation rate without compromising common accuracy measures, as discussed below. Table 1 lists the benchmark and production parameters for both systems. All simulations were run without temperature or pressure control.

We first show performance results using benchmark parameters. Table 2 shows elapsed time per time step for Desmond running on our cluster on the two benchmarks for 8 to 2048 processors. Figures 5 and 6 show this data graphically for ApoA1 and DHFR, respectively. Our default run for each system uses two processes per node and our new communication primitives. For ApoA1, we also show results using a single two-threaded process on each node ("tpn=2");

Benchmark Parameters

System	*Time step*	*Con-straints*	*PME frequency*	*Cutoff*	*PME mesh*	*PME order*
ApoA1	1fs	No	4 steps	12 Å	128×128×128	4
DHFR	1fs	No	1 step	9 Å	64×64×64	4

Production Parameters

System	*Time step*	*Con-straints*	*PME frequency*	*Cutoff*	*PME mesh*	*PME order*
ApoA1	2.5fs	Yes	2 steps	12 Å	64×64×64	6
DHFR	2.5fs	Yes	2 steps	9 Å	64×64×64	4

Table 1. Benchmark and production parameters. The cutoff is the cutoff radius used for explicit evaluation of pairwise electrostatic and van der Waals interactions; the PME order is the order of the B-splines used in the smooth PME algorithm. For ApoA1, NAMD used a coarser PME mesh of 108×108×80; Desmond and Blue Matter used 128×128×128 which, although more costly, is necessary due to the powers-of-2 restriction in Desmond's and Blue Matter's FFT calculation.

Processor Count	***ApoA1***	***ApoA1 tpn=2***	***DHFR***	***DHFR with MPI***
8	0.2568		0.0414	0.0429
16	0.1268	0.1415	0.0210	0.0219
32	0.0643	0.0704	0.0115	0.0121
64	0.0335	0.0374	0.0063	0.0069
128	0.0182	0.0206	0.0037	0.0044
256	0.0094	0.0105	0.0020	0.0026
512	0.0052	0.0062	0.0014	0.0023
1024	0.0030	0.0035		
2048	0.0020	0.0025		

Table 2. Desmond performance: Average elapsed time per time step (in seconds) using benchmark parameters.

Processor Count	***ApoA1***	***DHFR***
8	0.3542	0.0569
16	0.1730	0.0335
32	0.0898	0.0190
64	0.0479	0.0124
128	0.0285	0.0097
256	0.0234	0.0070
512	0.0186	0.0159
1024	0.0260	

Table 3. NAMD performance: Average elapsed time per time step (in seconds) using benchmark parameters, running on the Opteron/InfiniBand cluster. Both processors on each node were used for computation.

this mode is slightly slower at any given level of parallelism. For DHFR, we also show results using an MPI library (MVAPICH) instead of our communication primitives, illustrating that the performance gain due to these primitives increases as the number of processors increases.

To provide additional context for these results, we compare them to performance measurements on the same benchmarks for the NAMD and Blue Matter codes, both of which were designed specifically for high parallel performance [15-17, 19, 35]. Table 3 and Figures 5 and 6 show performance results for NAMD on our cluster. Desmond is faster than NAMD at all levels of parallelism examined, with the performance difference growing as parallelism increases. On ApoA1, Desmond scales to at least 2048 processors, at which point it

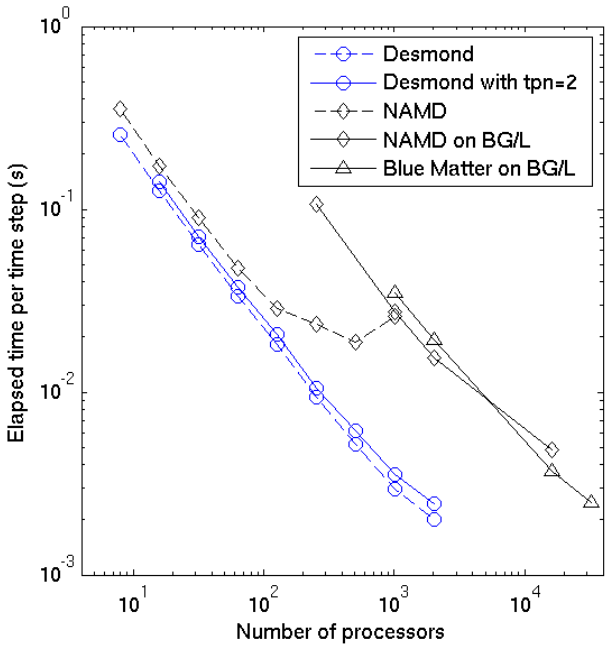

Figure 5. ApoA1: Average elapsed time per time step using benchmark parameters.

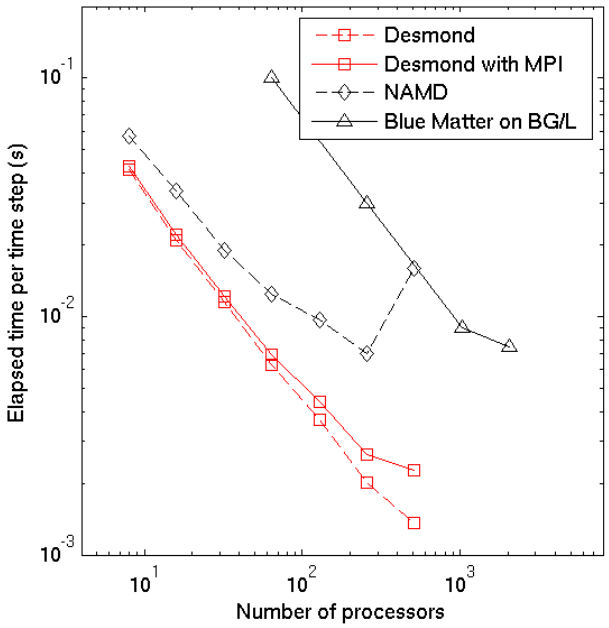

Figure 6. DHFR: Average elapsed time per time step using benchmark parameters.

is 9.3 times faster than NAMD running on 512 processors; NAMD's simulation rate decreases beyond 512 processors.

Table 4 shows performance measurements for the ApoA1 and DHFR benchmarks on Blue Gene/L using NAMD and Blue Matter. This data is also plotted in Figures 5 and 6 for comparison purposes. In the tests reported for Blue Matter [14, 15], both processors (cores) on each node were used for computation. In the tests reported for NAMD [24], one core was used for computation while the other was used for communication (co-processor mode).

Processor Count	*ApoA1 with NAMD [24]*	*ApoA1 with Blue Matter [16]*	*DHFR with Blue Matter [17]*
64			0.1
256	0.1063		0.03
1024	0.0276	0.0384	0.009
2048	0.0154	0.0190	0.007
16384	0.0048	0.0031	
32768		0.0021	

Table 4. Comparable results for NAMD and Blue Matter running on Blue Gene/L as reported in the literature: Average elapsed time per time step (in seconds) using benchmark parameters. For NAMD, separate timings for cutoff-based and PME calculations were given, and we have combined the results into a single number assuming one PME calculation every four steps.

Time step	*Constraints*	*Desmond*	*NAMD*
1 fs	No	2.0 K/ns	2.8 K/ns
1 fs	Yes	0.07 K/ns	0.4 K/ns
2.5 fs	Yes	0.07 K/ns	0.9 K/ns

Table 5. Energy drift comparison between Desmond and NAMD for ApoA1 at 300 K. The first row corresponds to the benchmark parameters, the second row corresponds to the benchmark parameters except for the use of constraints, and the third row corresponds to production parameters.

We point out in particular that, for ApoA1, Desmond's performance on 2048 processors of our cluster slightly exceeds that of Blue Matter on 32,768 processors of Blue Gene/L. Desmond on 2048 Opteron processors is 9.5 times faster than Blue Matter on the same number of Blue Gene/L processors. Desmond's performance on 1024 processors of our cluster exceeds NAMD's performance on 16,384 processors of Blue Gene/L, the highest level of parallelism reported in the NAMD tests [24]. The Opteron processors in our cluster are somewhat faster than the PowerPC 440 processors of Blue Gene/L; our cluster processors have a theoretical peak of 4.8 billion double precision floating point operations per second, as compared to 2.8 billion for the Blue Gene/L processors. On the other hand, Blue Gene/L's communication network is faster than our InfiniBand network; the Blue Gene/L torus network provides an aggregate raw bandwidth per node of 16.8 Gbps and a nearest-neighbor latency of 2–3 μs [1], while our network provides an aggregate raw bandwidth per node of 8.5 Gbps and a nearest-neighbor latency of approximately 6 μs.

Table 5 shows energy drift rates for Desmond and NAMD on ApoA1 for several sets of simulation parameters. Because the Desmond runs used the OPLS-AA 2005 force field while the NAMD runs used the CHARMM22 force field, the exact energy drift figures may not be compared strictly. However, the fact that Desmond's energy drift is lower than that of NAMD, which performs most of its computation in double precision, confirms that Desmond achieves acceptable energy drift despite its use of single precision computation. Thanks to the numerical stability of Desmond's constraint algorithms, its energy drift drops dramatically, both in absolute terms and relative to NAMD, when constraints are applied to fix the lengths of bonds to all hydrogen atoms and the bond angles of water molecules. In addition, increasing the time step from 1 fs to 2.5 fs in a constrained simulation has a negligible impact on energy drift in Desmond but a significant impact in

NAMD, suggesting that Desmond is able to use a longer time step while maintaining low energy drift.

We also quantified simulation accuracy using *relative rms force error*, a measure of the error in the calculated nonbonded forces defined as the root mean squared error in the force divided by the root mean squared force [41]. For ApoA1, Desmond's relative rms force error is lower using production parameters than using benchmark parameters. In particular, the relative rms force error using production parameters is consistently below 10^{-4}; a relative rms force error below 10^{-3} is considered sufficiently accurate for biomolecular MD simulations [47, 48]. For DHFR, the relative rms force error is identical for benchmark and production parameters.

The use of production parameters allowed us to more than double the simulation rates obtained using the benchmark parameters without increasing energy drift or relative rms force error. Figure 7 plots simulation rates for ApoA1 and DHFR using production parameters for various processor counts in terms of simulated nanoseconds per elapsed day. The average simulation rate of ApoA1 increased by a factor of 2.5 relative to the rate using benchmark parameters, mostly due to use of a larger time step and a coarser PME mesh. On 2048 processors, the simulation rate increased by a factor of 2.8. The average simulation rate for DHFR increased by a factor of 2.7, mostly due to use of a larger time step and less frequent PME calculation.

For ApoA1, a 14 ns simulation was sustained over 2.8 hours on 2048 processors at a rate of 120 ns/day. For DHFR, a simulation on 512 processors (256 nodes) maintained a rate of 173 ns/day. To our knowledge, these ApoA1 and DHFR simulation rates are the fastest reported for these systems.

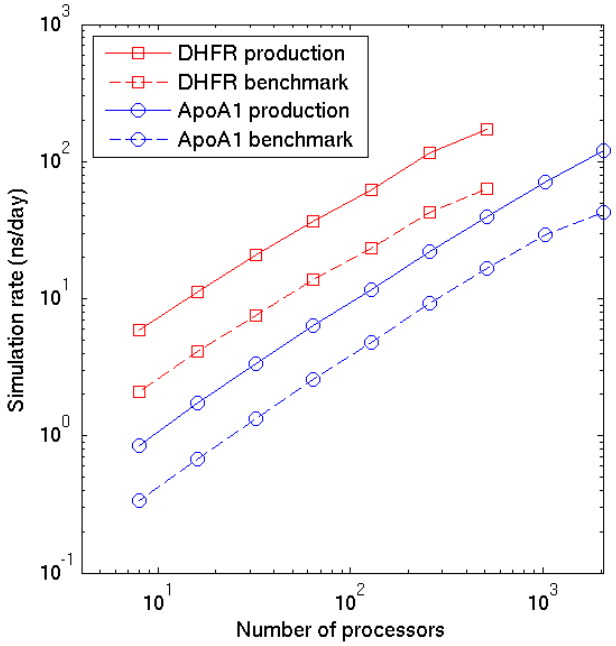

Figure 7. Desmond simulation rate for DHFR and ApoA1 using benchmark and production parameters, using two processes per node and new communication primitives.

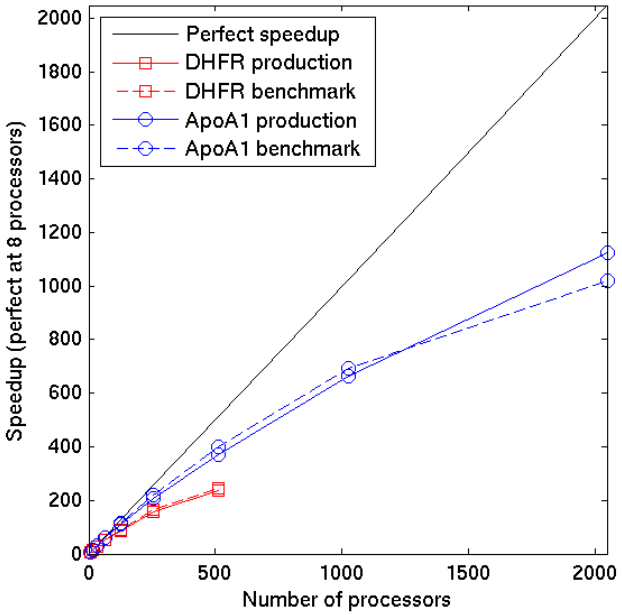

Figure 8. Strong scalability for DHFR and ApoA1, corresponding to the data in Figure 7.

Figure 8 plots speedup figures for the simulations relative to performance on 8 processors (i.e., assuming a speedup of 8 for 8 processors). The parallel efficiency is approximately 55 percent for ApoA1 for 2048 processors. The DHFR system has fewer atoms and less computational work per time step; its parallel efficiency is approximately 40 percent for 512 processors.

Applications. The proteins used for the performance tests reported in this paper were chosen to facilitate comparison with other benchmarks in the literature. Most of our Desmond simulations, however, have focused on other biomolecular systems of interest at a basic scientific level and/or in the context of drug development.

For example, we have studied the sodium-proton antiporter protein NhaA from *E. coli* in an effort to describe how this protein harnesses the electrochemical gradient of protons across the cell membrane to pump sodium ions out of the cell [2]. NhaA is critical for the maintenance of intracellular pH and sodium levels, but from the crystal structure alone it is not clear how it performs this essential function; molecular dynamic studies may play a key role in elucidating the mechanism.

We have investigated several protein kinases, proteins which play key roles in cellular signaling pathways. Most protein kinases can exist in both an active and an inactive conformation; defects which cause some protein kinases to become stuck in their active conformation are implicated in certain cancers. We have used molecular dynamics simulations to investigate elements of the conformational changes involved in the activation process (i.e., transition between inactive and active states) of one of these, c-Abl kinase, the target of the cancer drug Gleevec.

Also of interest to us, and a central practical problem in drug discovery, is the determination of binding affinities between small drug-like molecules and drug targets. High

performance molecular dynamics simulations may have the potential to provide an accurate solution to this problem, supplementing the faster but highly error-prone technique currently in use. We have started to investigate the interactions of a particular membrane-bound receptor protein implicated in cancer with small molecule inhibitors using molecular dynamics, as an important practical problem on which to test various approaches.

We have also used Desmond to simulate fast folding proteins. The fastest of these fold in a few microseconds or less, opening up the possibility of directly simulating the entire folding process. Although the details of folding are fascinating in themselves, one important reason to study such systems is that they provide a stern test of modern day molecular force fields. Hopefully, detailed study of the failures as well as the successes of current force fields can lead to a new generation of improved force fields that can be used to solve further intriguing problems in biology.

7. Concluding Remarks

We are undertaking a concerted effort to make molecular dynamics a useful tool in biology, chemistry, and medicine, particularly by reducing the turn-around time for computational experiments and enabling simulation of biochemical events on long time scales. Desmond represents an effort where we have developed new algorithms and implementation techniques, as well as MD software of unprecedented performance that is already being used in biochemistry research. In this paper we have outlined the novel methods underlying Desmond's efficiency and scalability, and shown that Desmond's parallel performance significantly exceeds that of other state-of-the-art codes. Other papers will present in more detail the specific methods we have developed, many of which are applicable to particle simulations in general.

Like established MD codes, Desmond contains a variety of features that are useful for research in chemistry and molecular biology. For example, it contains algorithms for simulations with various types of ensembles, sampling methods, force fields, and global box shapes. To handle the large amounts of particle trajectory data, Desmond uses a system of distributed output files. Desmond also contains a checkpoint mechanism that allows a bitwise perfect restart of a simulation that was stopped due to hardware failures or other reasons. Our continuing efforts with Desmond involve adding more features without compromising performance. We plan to release the Desmond software for use by universities and non-commercial research organizations at no cost.

Acknowledgements

We are grateful to Anne Weber, Christine McLeavey, Ross Lippert and Brian Towles for their comments and assistance with this paper. We also wish to thank the referees for their helpful suggestions.

References

[1] G. Almasi, C. Archer, J. G. Castanos, et al., Design and Implementation of Message-Passing Services for the Blue Gene/L Supercomputer, *IBM J. Res. & Dev.*, 49(2-3): 393-406, 2005.

[2] I. T. Arkin, H. Xu, K. J. Bowers, et al., Mechanism of a Na^+/H^+ Antiporter, *submitted*, 2006.

[3] K. J. Bowers, Speed Optimal Implementation of a Fully Relativistic 3D Particle Push with a Charge Conserving Current Accumulate on Modern Processors, presented at 18th International Conference on the Numerical Simulation of Plasmas, Cape Cod, MA, 2003.

[4] K. J. Bowers, R. O. Dror, and D. E. Shaw, Overview of Neutral Territory Methods for the Parallel Evaluation of Pairwise Particle Interactions, *J. Phys. Conf. Ser.*, 16: 300-304, 2005.

[5] K. J. Bowers, R. O. Dror, and D. E. Shaw, The Midpoint Method for Parallelization of Particle Simulations, *J. Chem. Phys.*, 124: 184109, 2006.

[6] K. J. Bowers, R. O. Dror, and D. E. Shaw, Zonal Methods for the Parallel Execution of Range-Limited *N*-Body Problems, *in press, J. Comput. Phys.*, 2006.

[7] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, et al., CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations, *J. Comput. Chem.*, 4: 187-217, 1983.

[8] C. L. Brooks, B. M. Pettit, and M. Karplus, Structural and Energetic Effects of Truncating Long Ranged Interactions in Ionic and Polar Fluids, *J. Chem. Phys.*, 83(11): 5897-5908, 1985.

[9] F. Cappello and D. Etiemble, MPI Versus MPI+OpenMP on the IBM SP for the NAS Benchmarks, presented at ACM/IEEE SC2000 Conference, Dallas, TX, 2000.

[10] D. A. Case, T. E. Cheatham, III, T. Darden, et al., The Amber Biomolecular Simulation Programs, *J. Comput. Chem.*, 26(16): 1668-1688, 2005.

[11] E. Chow and D. Hysom, Assessing Performance of Hybrid MPI/OpenMP Programs on SMP Clusters, Lawrence Livermore National Laboratory UCRL-JC-143957, 2001.

[12] T. Darden, D. York, and L. Pedersen, Particle Mesh Ewald: An N Log(N) Method for Ewald Sums in Large Systems, *J. Chem. Phys.*, 98(12): 10089-10092, 1993.

[13] Y. Duan and P. A. Kollman, Pathways to a Protein Folding Intermediate Observed in a 1-Microsecond Simulation in Aqueous Solution, *Science*, 282(5389): 740-744, 1998.

[14] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, et al., Scalable Framework for 3D FFTs on the Blue Gene/L Supercomputer: Implementation and Early Performance Measurements, *IBM J. Res. & Dev.*, 49(2-3): 457-464, 2005.

[15] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, et al., Blue Matter: Strong Scaling of Molecular Dynamics on Blue Gene/L, IBM RC23888, February 22, 2006.

[16] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, et al., Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics, IBM RC23956, May 12, 2006.

[17] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, et al., Blue Matter: Strong Scaling of Molecular Dynamics on Blue Gene/L, IBM RC23688, August 5, 2005.

[18] M. Frigo and S. G. Johnson, The Design and Implementation of FFTW3, *Proceedings of the IEEE*, 93(2): 216-231, 2005.

[19] R. S. Germain, B. Fitch, A. Rayshubskiy, et al., Blue Matter on Blue Gene/L: Massively Parallel Computation for Biomolecular Simulation, presented at 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '05), New York, NY, 2005.

[20] T. A. Halgren, MMFF VII. Characterization of MMFF94, MMFF94s, and Other Widely Available Force Fields for Conformational Energies and for Intermolecular-Interaction Energies and Geometries, *J. Comput. Chem.*, 20(7): 730-748, 1999.

[21] G. S. Heffelfinger, Parallel Atomistic Simulations, *Comput. Phys. Commun.*, 128(1-2): 219-237, 2000.

[22] W. L. Jorgensen, D. S. Maxwell, and J. Tirado-Rives, Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids, *J. Am. Chem. Soc.*, 118(45): 11225-11236, 1996.

[23] P. A. Kollman, R. W. Dixon, W. D. Cornell, et al., "The Development/Application of a "Minimalist" Organic/ Biomolecular Mechanic Forcefield Using a Combination of Ab Initio Calculations and Experimental Data," in *Computer Simulation of Biomolecular Systems: Theoretical and Experimental Applications*, W. F. van Gunsteren and P. K. Weiner, Eds. Dordrecht, Netherlands: ESCOM, 1997, 83-96.

[24] S. Kumar, G. Almasi, C. Huang, et al., Achieving Strong Scaling with NAMD on Blue Gene/L, presented at IEEE International Parallel & Distributed Processing Symposium, Rhodes Island, Greece, 2006.

[25] J. Liu, J. Wu, and D. K. Panda, High Performance RDMA-Based MPI Implementation over InfiniBand, presented at 17th International Conference on Supercomputing, San Francisco, CA, 2003.

[26] J. MacKerell, A. D., D. Bashford, M. Bellott, et al., All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins, *J. Phys. Chem. B*, 102(18): 3586-3616, 1998.

[27] P. Mark and L. Nilsson, Structure and Dynamics of Liquid Water with Different Long-Range Interaction Truncation and Temperature Control Methods in Molecular Dynamics Simulations, *J. Comput. Chem.*, 23(13): 1211-1219, 2002.

[28] Mellanox Technologies, Mellanox IB-Verbs API (VAPI): Mellanox Software Programmer's Interface for InfiniBand Verbs, 2001.

[29] T. Narumi, A. Kawai, and T. Koishi, An 8.61 Tflop/s Molecular Dynamics Simulation for NaCl with a Special-Purpose Computer: MDM, presented at ACM/IEEE SC2001 Conference, Denver, Colorado, 2001.

[30] J. Norberg and L. Nilsson, On the Truncation of Long-Range Electrostatic Interactions in DNA, *Biophys. J.*, 79(3): 1537-1553, 2000.

[31] V. S. Pande, I. Baker, J. Chapman, et al., Atomistic Protein Folding Simulations on the Submillisecond Time Scale Using Worldwide Distributed Computing, *Biopolymers*, 68(1): 91-109, 2003.

[32] P. M. Papadopoulos, M. J. Katz, and G. Bruno, NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters, *Concurrency Comput. Pract. Ex.*, 15(7-8): 707-725, 2003.

[33] M. Patra, M. Karttunen, T. Hyvönen, et al., Molecular Dynamics Simulations of Lipid Bilayers: Major Artifacts Due to Truncating Electrostatic Interactions, *Biophys. J.*, 84: 3636-3645, 2003.

[34] J. C. Phillips, R. Braun, W. Wang, et al., Scalable Molecular Dynamics with NAMD, *J. Comput. Chem.*, 26(16): 1781-1802, 2005.

[35] J. C. Phillips, G. Zheng, S. Kumar, et al., NAMD: Biomolecular Simulation on Thousands of Processors, presented at ACM/IEEE SC2002 Conference, Baltimore, 2002.

[36] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular-Dynamics, *J. Comput. Phys.*, 117(1): 1-19, 1995.

[37] S. Plimpton and B. Hendrickson, Parallel Molecular-Dynamics Simulations of Organic Materials, *Int. J. Mod. Phys. C.*, 5(2): 295-298, 1994.

[38] S. Plimpton and B. Hendrickson, A New Parallel Method for Molecular Dynamics Simulation of Macromolecular Systems, *J. Comput. Chem.*, 17(3): 326-337, 1996.

[39] W. R. P. Scott, P. H. Hünenberger, I. G. Tironi, et al., The GROMOS Biomolecular Simulation Program Package, *J. Phys. Chem. A*, 103(19): 3596-3607, 1999.

[40] M. M. Seibert, A. Patriksson, B. Hess, et al., Reproducible Polypeptide Folding and Structure Prediction Using Molecular Dynamics Simulations, *J. Mol. Biol.*, 354(1): 173-183, 2005.

[41] Y. Shan, J. L. Klepeis, M. P. Eastwood, et al., Gaussian Split Ewald: A Fast Ewald Mesh Method for Molecular Simulation, *J. Chem. Phys.*, 122: 054101, 2005.

[42] T. Shanley, *InfiniBand Network Architecture*. Boston: Addison-Wesley, 2003.

[43] D. E. Shaw, A Fast, Scalable Method for the Parallel Evaluation of Distance-Limited Pairwise Particle Interactions, *J. Comput. Chem.*, 26(13): 1318-1328, 2005.

[44] M. Snir, A Note on N-Body Computations with Cutoffs, *Theor. Comput. Syst.*, 37: 295-318, 2004.

[45] D. van der Spoel, E. Lindahl, B. Hess, et al., GROMACS: Fast, Flexible, and Free, *Journal of Computational Chemistry*, 26(16): 1701-1718, 2005.

[46] M. Taiji, T. Narumi, Y. Ohno, et al., Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations, presented at ACM/IEEE SC2003 Conference, Phoenix, Arizona, 2003.

[47] R. Zhou and B. J. Berne, A New Molecular Dynamics Method Combining the Reference System Propagator Algorithm with a Fast Multipole Method for Simulating Proteins and Other Complex Systems, *J. Chem. Phys.*, 103(21): 9444-9459, 1995.

[48] R. Zhou, E. Harder, H. Xu, et al., Efficient Multiple Time Step Method for Use with Ewald and Particle Mesh Ewald for Large Biomolecular Systems, *J. Chem. Phys.*, 115(5): 2348-2358, 2001.