From the Proceedings of High Performance Computing '94
Proceedings of the 1994 SCS Simulation Multiconference, April 1994

MD – A Flexible Framework for High-Speed Parallel Molecular Dynamics

Greg Eisenhauer and Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

This paper introduces MD, a complete high-performance parallel molecular dynamics simulation system. MD efficiently manages the control and data flow required for the basic molecular dynamics calculations as well as for global statistics and scientific visualization. We briefly examine the structure of MD, present an analysis of its performance for several physical systems, and examine ways in which it can extended.

1 Introduction

Creating efficient high-speed parallel programs is a difficult endeavor. Past efforts have failed to produce a generally accepted method for writing such programs that can be readily employed by the community of scientific programmers. Therefore, one important contribution to this community is the creation of a flexible parallel framework that can be employed for writing an entire class of scientific applications. This paper presents a domain-specific library for high performance parallel programming for the domain of molecular dynamics.

MD is a complete high-performance parallel molecular dynamics simulation system. It efficiently manages the control and data flow required for the basic molecular dynamics calculations as well as for global statistics and scientific visualization. MD is written using Cthreads, a multi-processor threads package that allows easy control of thread-based parallelism and provides portability between a variety of shared-memory multiprocessor and uniprocessor platforms. Although MD is currently implemented on shared-memory multiprocessors, its basic structure is easily adaptable to distributed-memory machines as well.

We call MD a *framework* because it was designed to be portable and sufficiently flexible for a variety of uses, while still maintaining efficiency and performance. MD provides for the management of parallelism achieved by domain decomposition, where the unit of decomposition is called a *molecule*. A *molecule* is a cluster of particles that exhibit some spatial locality, much like a physical molecule. While simpler programs might limit the spatial decomposition to slices over one or two dimensions, MD extends the notion of *domain* to any describable region of space. In our implementation, entire molecules, not individual particles, are assigned to a domain. Each domain is assigned to a processor and and that processor is responsible for computing the forces on the particles belonging to the molecules in its domain.

This work has led to useful contributions both in the domain of molecular dynamics simulations and in programming technology for parallel machines. Of particular importance is ability of the MD framework to support complete applications, from constructs for programming basic MD computations – called *basic* MD – to provision of fast, portable binary I/O and the support of scientific data visualization – called *extended* MD. This distinguishes MD from existing parallel libraries like LAPack or code kernels like the NASA kernels. Examining a complete high-performance scientific application means that the inferences we make from its demands should have more general applicability than those drawn from examining a kernel.

The following sections will examine our contributions in more detail. The specific molecular dynamics application used in our work is a simulation of the behavior of hydrocarbon chains on a fixed substrate. Its dominant computational requirement is calculating the long-range forces between particles, but other required computations with different characteristics also affect the application's structure and behavior. These computations include finding the bond forces within the hydrocarbon chains, determining system-wide characteristics such as atomic temperature, and performing analysis and online visualization. The implementation of this application provides a vehicle for refining and evaluating MD.

2 The Structure of Basic MD

2.1 Problem Types Supported by MD

To understand the structure of MD it is necessary to explain the nature of the class of problems it targets and the required types of computations. Both are described next.

The problem class. The solution technique employed in the type of molecular dynamics simulations supported by MD is as follows: given the locations of all particles in the system, calculate the forces on each particle, calculate movement of the particles in some time interval Δt, update the locations and repeat.

We are specifically interested in physical systems where most or all of the particles are bound into the larger association of a physical molecule. It is most natural to map a physical molecule to the data structure called a *molecule* in MD. We assume a one-to-one mapping in this paper because it simplifies the description, but it is by no means required. The only requirement for a MD molecule is that it be a set of particles with some degree of physical locality.

The simulation itself takes place in a two or three dimensional space. MD provides for the decomposition of this space into multiple *spatial domains*. These domains form the basis for the parallelization.

Nature of the abstract computation. Discounting global statistics, analysis, and visualization, the most computationally intensive aspect of this class of application is the calculation of the long-range pair interactions between particles. This calculation is $O(n^2)$ in complexity, where n is the total number of particles in the worst case. In comparison, calculating the forces resulting from intra-molecular bond interactions is an $O(n)$ computation. In order to reduce computational demand, it is common to employ a *cutoff radius*. Particles outside the cutoff radius are assumed to have a net zero contribution to the calculation. One common implementation of cutoff radius is to maintain, for each particle in the system, an *interaction list* of all other particles within this cutoff radius. This list does not have to be recomputed for every timestep; it can generally be used for several iterations. However, the problem of computing and updating the interaction list is still of $O(n^2)$ complexity, and the interaction list's memory requirements can be quite large*.

For a sample application to exercise MD we choose a physical simulation used by physicists at Georgia Tech(Xia et al. 1992). The simulation is designed to investigate evolutionary trends of the structure and dynamics of n-hexadecane (C_{16}-H_{34}) films on a crystalline substrate modeling $Au(001)$. In the simulation, the alkane system is described via intramolecular and intermolecular interactions between pseudoatoms (CH_2 and terminal CH_3 segments) and the substrate atoms. The calculational cell is a square cylinder which is periodically repeated in the x-y directions. Temperature is controlled via infrequent scaling of the particles' velocities. For this sample system, the pseudoatoms are represented as MD *particles*. The alkanes are represented as MD *molecules*. For purposes of implementation, the substrate is also organized into "molecules", although it is in fact a particle lattice. This results in a more regular code structure and allows MD to take advantage of the spatial locality of these substrate "molecules". The alkanes remain associated in a chain with very predictable bond lengths throughout the simulation. This provides an implicit *locality* that can be exploited. The next section provides the details of our implementation.

*One implementation of the interaction list technique which we have studied used reasonably complex techniques to try to pack the interaction list as tightly as possible. The interaction list still accounted for over 80% of the memory used.

2.2 Parallelization Scheme

As mentioned previously, MD employs a spatial-decomposition approach to parallelization. The physical system is divided into spatial domains and each domain is allocated to a processor. Molecules are assigned to domains by their centers of mass, and the processor on which a domain is placed is responsible for calculating the forces on the domain's alkanes. In order to calculate the forces on its alkanes due to the presence of alkanes in other domains, each processor must obtain information from all spatial domains whose particles might affect its own, called *neighbor domains*. The number of neighboring spatial domains is determined by the geometry of the domains, the sizes of the domains, and the size of the cutoff radius. The information flow between domains represents the main communication requirement for this class of MD applications, and it must take place once for each simulation timestep. Other communication requirements include the occasional need to calculate a global temperature value to control the temperature of the simulation. This is done infrequently for stable systems and has little impact on overall communications.

Given this approach, the basic process of simulation involves the following steps from each processor:

1. obtain needed particle location information from neighbor domains,
2. calculate all the forces on local particles,
3. apply the calculated forces to yield new particle positions, and
4. publish the new particle locations.

2.3 Summary of Basic MD

The abstractions presented by *Basic* MD and used in the sample molecular dynamics simulation presented above are sufficient for implementation of the computations and data structures required for a variety of complex

MD simulations. Next, we summarize the constructs offered by *Basic* MD, and we describe their performance evaluation using the sample MD simulation. The facilities offered by *Extended* MD are presented in Section 4.2 below.

The first abstraction offered in MD is that of a *particle*, which is the basic dynamic element the simulation. Particles are represented as data structure containing the elements' type, location, derivatives of motion and fields for the accumulation of forces on the element. All particles in MD are members of *molecules*, which are defined as sets of particles that exhibit some spatial locality throughout the simulation. Molecules and represented as data structures which contain contain an array of particles as well as additional information about the molecule itself, such as its center of mass, its spatial extent (or *bounding box*) and the number of particles it contains.

The system simulation space is divided into one or more *domains*. The number and configuration of the domains is a *decomposition*. A decomposition is characterized by the number of domains it contains, the neighbor lists of those domains, and a function that maps coordinates in simulation space to domains. Entire molecules are assigned to the domain which contains their center of mass. The domain assignment of a molecule may change as the molecule moves in space during the course of the simulation.

Domains are the unit of allocation of data and of execution threads to processors, where with each domain is associated one thread responsible for performing both the intra-molecular computations for the domain's molecules and the inter-molecular computations for its and its neighbor domains' molecules. However, additional execution threads may be created within each domain in order to overlap inter-domain communications with computations internal to each domain. Toward this end, MD offers access to the functionality of the underlying Cthreads package(Mukherjee 1991). In this case, programmers would use the `cthread_fork` facility in Cthreads, which creates and schedules a new thread for execution on a designated processor. Global synchronization, stopping all processors for some span of time to perform a system-wide function, is also accomplished using a combination of the `conditions` and `mutexes` available in Cthreads.

3 Performance Evaluation

Evaluating MD is a relatively complex task. Here, we ignore issues associated with how easy it is to create new simulations within the MD framework. Instead, we evaluate the advantages and disadvantages of the abstractions discussed in the previous section by comparing the performance of MD on the KSR-1 to that of an interaction list based molecular dynamics application running on a Cray YMP. Additionally, we concentrate on examining different ways of employing the facilities offered by MD.

In the cases presented below, MD was run on a Kendall Square Research KSR-1. The KSR supercomputer is a NUMA (non-uniform memory access) shared memory, cache-only architecture with an interconnection network that consists of hierarchically interconnected rings, each of which can support up to 32 nodes. Each node consists of a 64-bit processor, 32 MBytes of main memory used as a local cache, a higher performance 0.5 Mbyte sub-cache, and a ring interface. CPU clock speed is 20 MHz, with peak performance of 20 Mflops per node (on the KSR-2 to be released early 1994, these speeds will be doubled). Access to non-local memory results in the corresponding cache line being migrated to the local cache, so that future accesses to that memory element are relatively cheap. The parallel programming model implemented by the KSR's OSF Unix operating system is one of kernel-level threads which offer constructs for thread fork, thread synchronization and shared memory between threads. This kernel-level thread facility is called Pthreads. The Cthreads library mentioned previously is a user-level threads facility that is built on top of Pthreads. The advantages of Cthreads include increased performance compared to Pthreads for thread creation, context switches, and synchronization. Cthreads also gives users the flexibility to design and use threads primitives customized to their applications.

3.1 Overall Results

Of particular interest are the performance results for the base system mentioned in Section 2.1. This system consists of 300 *n*-hexadecane molecules (16 particles each) on a substrate of 1350 gold particles. The substrate is organized into 90 "molecules".

Normative comparison. Our base point of comparison is a Fortran program that simulates this system using the commonly used interaction list technique. It has been used in previous experimentation by the physicists with whom we are working. For this program, a Cray Y-MP requires an average of .65 seconds to compute a single timestep of the simulation. The majority of the CPU time is spent in updating the interaction list, which is done only every eighth timestep. The same Fortran code running on a single KSR-1 processor averages 15.4 seconds per timestep, which roughly compares with the 18.6 seconds required by MD using a single processor. However, it is not useful to speculate on the utility of interaction list vs. molecular representations of particles with this comparison, in part because the specific solution algorithms used in the Fortran code are slightly different

	Seconds per Timestep
Cray YMP	.65
KSR-1 using 49 processors	.81
KSR-1 using 60 processors	.74

Table 1: Comparison of MD performance on KSR-1 vs. interaction lists on Cray YMP

from those used in MD. Furthermore, the Fortran code has not been optimized or parallelized for the KSR.

The primary result of interest to end users of parallel systems is shown in Table 1. These measurements demonstrate that an application implemented with MD on the 64-node* KSR-1 can achieve roughly the same performance as a highly-tuned interaction list based implementation on a Cray YMP. The version of MD measured here has not been optimized heavily. Specifically, we have not added machine-specific optimizations like KSR's prefetch or poststore instructions, we have not optimized the assembly code generated by the KSR's C compiler, and we have not customized the underlying parallel programming library (Cthreads) to remove any specific operating system inefficiencies. Given these results, it is fair to say that parallel MD on the KSR-1 can attain supercomputer performance. This is especially satisfying given the slow KSR-1 processors (20 MHz!) compared to the Cray YMP machine. Furthermore, as shown next, performance improvements scale well with machine size, so that larger scale KSR machines should be able to outperform Cray YMP's and other comparable machines.

Speedup results. While the performance achieved above is satisfying, it is more important in the long term to understand how MD will scale to larger physical systems and to larger parallel machines. Of particular importance is how *computational costs* change when using the MD framework. For an interaction list based application, we would expect the time required to compute a timestep to rise with with the square of the number of particles involved in the simulation. For MD, the function expressing computational cost is considerably more complex. In order to explain MD computational costs, let us first examine the sample system. Figure 1 compares (1) the actual iteration times for MD and for the sample system with (2) the iteration times we would expect assuming perfect speedup (i.e., the time for one processor divided by the number of processors). Note that while the iteration time decreases with an increasing number of processors, speedup is not linear. This behavior can be explained with certain characteristics of the sample physical system being simulated. In this system, the cutoff radius amounts to 13.4% of the x-y extent of the system and a "stretched out" alkane can span up to 43% of the same extent. One consequence of these numbers is that the system is very highly coupled. At least in the x and y dimensions, no domain can be so far from another domain that none of molecules could interact. This means that regardless of the way we decompose the system, each domain must have all other domains as members of its neighbor list, or some potential interactions might be missed. Therefore, independent of system decomposition, every processor must communicate with every other processor on every timestep.

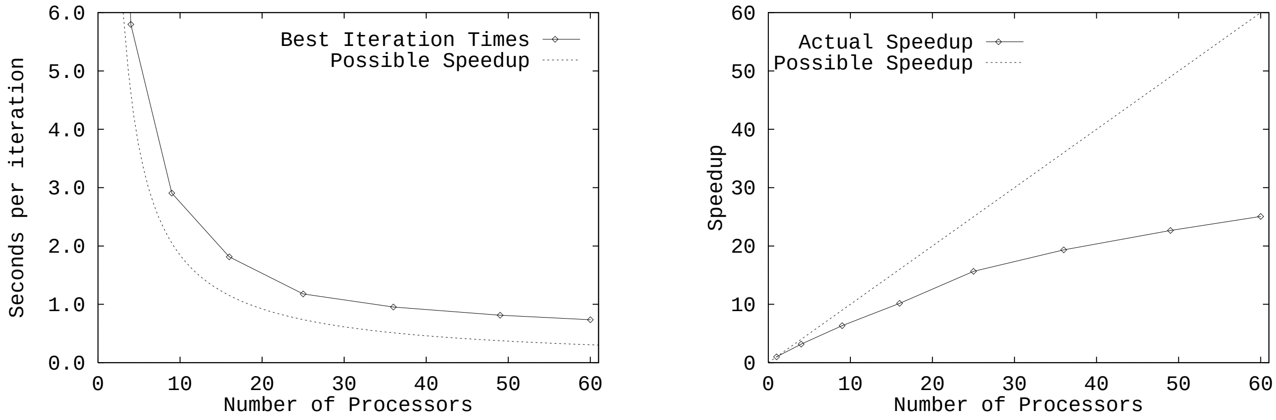
*At most 60 of the 64 nodes were employed because several nodes are reserved for operating system functions and cannot be used by applications.

In general, global communication on every timestep is a major performance problem for physical simulations on parallel machines, but it cannot be avoided for the sample system because we must maintain the integrity of the calculation. In general, other parameters will also affect the speed at which a timestep can be calculated. These parameters include (1) characteristics of the simulated system (such as density and cutoff radius) as well as (2) parameters which can be specified to MD (such as the number of processors employed and the decomposition scheme). In general, it is important to understand how the time required to calculate a timestep varies with these parameters. The changing costs of simulation may mean that some simulations which were previously impractical are now within the realm of possibility. We cannot possibly explore all the corners of performance space, but we examine the consequences of changing the physical parameters in a single system.

3.2 Different Physical Systems

In order to further examine the behavior of MD, we have created an artificial system that differ from the base sample system in various characteristics. This system has been derived from the original by mechanical processes such as particle trimming and replication. While simulations so created do not necessarily have significance to the physics community, they should give an indication of the behaviors of actual simulations with similar characteristics.

The Thin64 system. The artificial system we will examine is called "Thin64". It was created by "thinning" the base system by a factor of 64 (removing all but every 64^{th} molecule and substrate) and placing 64 copies of the trimmed system in an 8×8 grid. The resulting system has roughly the same number of alkanes and substrate elements as the base system, but it is considerably less dense and should have fewer interactions than the



(a) Time

(b) Speedup

Figure 1: Best attained MD performance results for the sample system as compared to the best possible times assuming perfect linear speedup.

base system. More important from the point of view of MD's performance, however, is that the ratio of cutoff radius and molecular extent to the dimension of the system have been diminished by 8. The result is a substantially less coupled system.

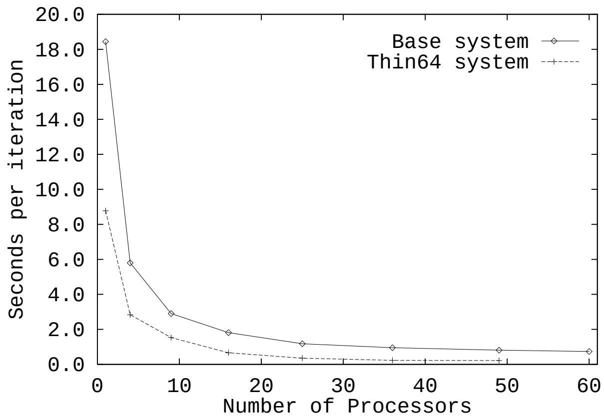


Figure 2: MD performance results comparing the base system and the Thin64 system.

Figure 2 compares the iteration time and speedups for the base system and the Thin64 system. Despite the fact that Thin64 has 7% more alkanes and 40% more substrate particles*, MD is able to accomplish an iteration in only .22 seconds using 49 processors. This is substantially less than the completion time of an iteration on the base system. While we have not done any experiments on the Cray YMP with the Thin64 system, the Cray version of this code spends most of its time recomputing interaction lists, a task which is not reduced in Thin64. Therefore we believe that MD is also substantially faster than the Cray code for this system.

*The fact that the Thin64 system has more elements than the base system is an artifact of the method used to produce it.

The numbers reported for the Thin64 system in Figure 2 were achieved using a cylindrical decomposition. That is, each domain is a square cylinder of simulation space. To this point, we have presented only the "best" decompositions of the systems we have examined. Other decompositions are of course possible, though the analysis of these alternative decompositions is beyond the scope of this paper.

4 Towards Extended MD

As stated earlier, *Basic*MD addresses the computational kernels of molecular dynamics applications, while *Extended*MD addresses entire MD applications. Future work on both is discussed next.

4.1 Future Work on Basic MD

Part of our future work with MD will be continued evaluation of the suitability of creating MD codes with our framework. Toward this end, we are working with physicists on solving additional real MD problems, and we are continuing to evaluate alternative implementations of the current system being simulated. Of particular importance is the evaluation of complex decompositions that might be required to match odd or asymmetric molecular arrangements, such as might result from the inclusion of physical probes in the system being simulated. Further study of dynamic reconfiguration of domain decompositions is also quite important.

The goals of this research are to demonstrate the suitability of MD for a large variety of MD simulations and to evaluate the performance issues in building large scale parallel MD codes.

4.2 The Functionality of Extended MD

Much of the functionality of any large-scale MD simulation is not supported by basic MD. For example, such simulations require fast portable binary I/O, support for on-line or off-line visualization and performance evaluation, and support for construction of multi-machine MD codes. While space prohibits a full discussion of the facilities of Extended MD, we do present our work and plans for visualization and program steering.

4.2.1 Visualization Library

Scientific visualization and data analysis are often thought of as after-the-simulation activities to be performed on high performance workstations. However, as analysis and visualization are becoming more complex, they are increasingly requiring parallelization themselves. This fact, coupled with the desire to allow greater scientific insight during long simulation runs, implies that the supercomputers employed for high-speed simulations will soon be called upon to perform these visualization calculations as well as core simulation computations. Accordingly, extended MD includes a basic visualization capability, which can run concurrently with the core simulation.

Several interesting issues are raised when considering on-line MD visualization. One such issue addresses the basic tradeoffs between computational costs and visual accuracy when producing visualizations. Complex visualization techniques may provide accurate rendition of a set of particles with appropriate lighting models, shadows, etc. However, such techniques are computationally costly, must be internally parallelized if they are to result in online visualizations, and are often too costly for their on-line use. For example, journal-quality color graphics are not suitable if the main purpose of online visualization is to get a general idea of the progress of the computation.

In response to this issue, we have initially chosen to provide online visualizations with a minimalistic approach, thereby avoiding complex graphic rendering techniques. Specifically, we utilize sorting and simple z-buffer style drawing techniques to create representations of the physical system. Rather than to attempt to parallelize this method, the production of frames is pipelined. Therefore, if computation of a single frame requires the same amount of time as three timesteps, we will be computing three frames concurrently. While the first frame will not appear until after the third timestep, subsequent frames will appear on every timestep, lagging the computation by three frames. We have had considerable success with this minimalistic approach, and we are now constructing higher quality visualization support based on the Silicon Graphics GL library, in part to evaluate the viability of such quality for online visualization.

A second issue addressed by our research concerns the available network bandwidth, which currently limits the speed of online visualization. A variety of improvements are underway. First, we are experimenting with data compression protocols and other means to overcome the bandwidth limitations. Second, we would like to enhance the visualization capabilities. Allowing interactive control over the visualization (to rotate, zoom, remove layers, etc.) and allowing different characteristics of the molecule (such as temperature, stress, etc.) to affect how it is visualized are two promising possibilities. Third, we would like to exploit the capabilities of the parallel machine to produce a more visually pleasing image. The current rendering technique can provide only primitive depth cues. Ray tracing or some other technique would produce a much more comprehensible image.

4.2.2 Interactive Visualization and Program Steering

Another intriguing possibility is to incorporate other work we have been doing on interactive *steering* of parallel applications. Steering is defined as configuring a program during execution via an algorithm or by human interaction, where the purpose of such configuration is to affect the program's execution behavior. Coupled with enhanced visualization capabilities, steering could make this type of scientific exploration an interactive activity. Rather than concentrating on batch jobs and offline data reduction, the scientist could watch the system evolve online, modifying temperature, external pressure, timestep size or other characteristics to observe their effect on the developing system. This is an ambitious project, but we already have much of the infrastructure required to make this kind of system a reality.

5 Acknowledgments

We would like to thank Bill Ribarsky and Uzi Landman for their help in this getting this project started, Jian Ouyang for his work on the force calculations, and Tingkang Xia and Charles Cleveland for their help in debugging the physics of the sample system.

References

Mukherjee, B. 1991. A portable and reconfigurable threads package. In *Proceedings of Sun User Group Technical Conference*, pp. 101–112.

Xia, T. K., J. Ouyang, M. W. Ribarsky, and U. Landman 1992. Interfacial alkane films. *Physical Review Letters 69*(13), 1967–1970.