International Journal of Modern Physics C, Vol. 4, No. 6 (1993) 1075–1084
© World Scientific Publishing Company

MULTI-MILLION PARTICLE MOLECULAR DYNAMICS ON THE CM-5*

P. S. LOMDAHL, D. M. BEAZLEY, P. TAMAYO, N. GRØNBECH-JENSEN
Theoretical Division and Advanced Computing Laboratory
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

We outline a recently developed short-range molecular dynamics (MD) algorithm for message-passing MIMD computers. Timings and performance numbers are presented for a code, SPaSM, which implements the algorithm on the Connection Machine-5 (CM-5). We demonstrate that runs with more than 10^8 particles are now possible on massively parallel MIMD computers. The speed of the code scales linearly with the number of processors and with the number of particles and shows 95% parallel efficiency in the speedup. Recent results from 2D simulations of fracture dynamics are also presented

1. Introduction

A new scalable algorithm[1] for short-range molecular dynamics[2,3] simulations on distributed memory MIMD multicomputer based on a message-passing multi-cell approach is outlined. The algorithm allows for simulation of at least 10^8 particles interacting via a relatively short range potential. We have implemented the algorithm in a code, SPaSM (Scalable Parallel Short-range Molecular dynamics), on the Connection Machine 5 (CM-5) and demonstrated that simulations with tens of millions of atoms can now be performed routinely. In addition, it is clear that simulations with 10^8 particles are now possible. To our knowledge such MD systems are at least an order of magnitude larger than what has previously been reported[4,5,6,7,8,9]. In preliminary 2D studies, we have simulated the fracture dynamics of a piece of material with 2 million atoms that is being pulled apart in a tensile experiment. This simulation with 75000 time steps ran for approximately 14 hours on a 512 node CM-5. In this paper we present recently improved timings for our algorithm in addition to preliminary results from 2D fracture simulations.

*Work performed under the auspices of the US Department of Energy.

2. The MD Problem

The MD method concerns the solution of Newton's equation of motion for N interacting particles. This general N-body problem involves the calculation of $N(N-1)/2$ pair interactions in order to compute the total force on any given particle:

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = -\sum_{j \neq i} \frac{\partial V_{ij}(|\mathbf{r}_j - \mathbf{r}_i|)}{\partial \mathbf{r}_i}, \tag{1}$$

here $\mathbf{r}_i$ indicates the instantaneous position and m_i the mass of particle i. The complexity of the force calculation is simplified considerably if the potential $V_{ij}(r)$ has a finite range of interaction. This is a reasonable approximation of the atomistic interactions in many solids and fluids. In our study here we have used the Lennard-Jones 6-12 (LJ) potential

$$V(r) = \begin{cases} 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) & 0 < r \leq r_{max} \\ 0 & r_{max} < r \end{cases} \tag{2}$$

Here σ and ϵ are the usual LJ parameters. The potential is cut-off at r_{max}, i. e. no particles interact beyond this range. Our study here was done for $r_{max} = 2.5\sigma$. We stress that while more complicated and accurate potentials which include many-body effects are available, the qualitative physics is represented well by the LJ potential. The number of interacting neighbors for each particle depends on the value of the cut-off distance r_{max} and the particle density ρ.

Our code is written in ANSI C with explicit calls to the CMMD message-passing library. The kernel of the force calculation is coded in the CM-5 vector unit (VU) assembler language, CDPEAC, and consists of approximately 60 lines of code. All our calculations were performed in *double precision.*

3. The Multi-Cell MD Algorithm

Our algorithm has been described in detail in[1]. Here we briefly outline its main features, illustrating the algorithm in 2D, but it extends naturally to 3D. Space is considered to be a rectangular region with periodic boundary conditions. This region is subdivided into large cells that are assigned to the processing nodes (PNs) on the CM-5. The region assigned to each PN is further subdivided into small cells with dimensions slightly larger than the cutoff distance. Particles are assigned to a particular cell geometrically according to the particle's coordinates. In Fig. 1, solid lines represent processor boundaries while dashed lines represent the cells created on each PN. For large simulations, many thousands of cells per PN may be created (this does not explicitly depend on the number of PNs being used).

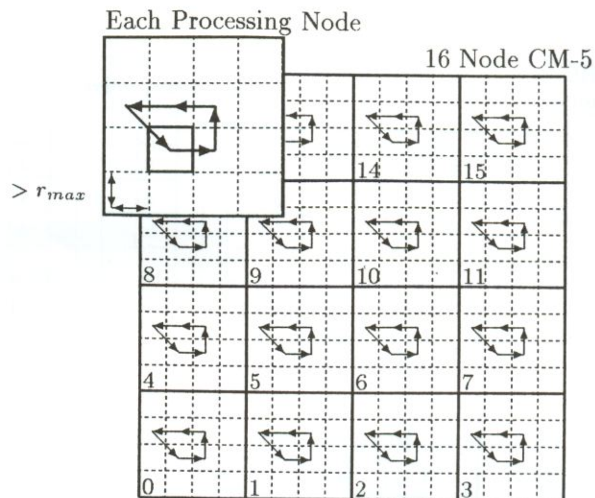

Fig. 1. Processor layout and force calculation.

Associated with each cell is a small block of memory for storing a sequential list of particles. Each particle is a C structure containing the position, velocity, force, mass, and type. The cell layout allows us to calculate the forces on all of the particles. To compute the forces for particles in a cell, we first compute all of the interactions between particles in that cell. Afterwards, forces between particles in neighboring cells are calculated by following an interaction path that visits neighboring cells. The path in 2D is shown in Fig. 1. As we follow the path, accelerations are accumulated by the original cell and any visited cells (using Newton's third law). To calculate all of the forces, this procedure is carried out on all cells on all of the PNs. Cells will accumulate accelerations from their lower neighbors when they calculate their interactions. Whenever the interaction path crosses a processor boundary, message passing is used to communicate particle data. After all forces have been calculated, the particle positions are updated. Since our algorithm is geometrically based, all of the data structures must be updated to account for positional changes. The particle coordinates are checked and if a particle is in the wrong cell it is moved to the proper cell. If the new cell is on a different PN, asynchronous message passing is used to send the particle to its new PN. Each PN checks for incoming particles and places them in the proper cell when received. Since a large number of cells may be created on each PN (even for moderately sized systems) hundreds or even thousands of message-passing calls may be required for each time step.

4. Using The Vector Units and Parallel Memory

On each processing node, the CM-5 has four vector units (VUs) that operate in SIMD mode. Each VU has a peak speed of 32 Mflops for a combined speed of 128 Mflops per node. In addition to performing fast vector operations the VUs also act as memory controllers with each VU controlling an 8 Mbyte bank of memory. The

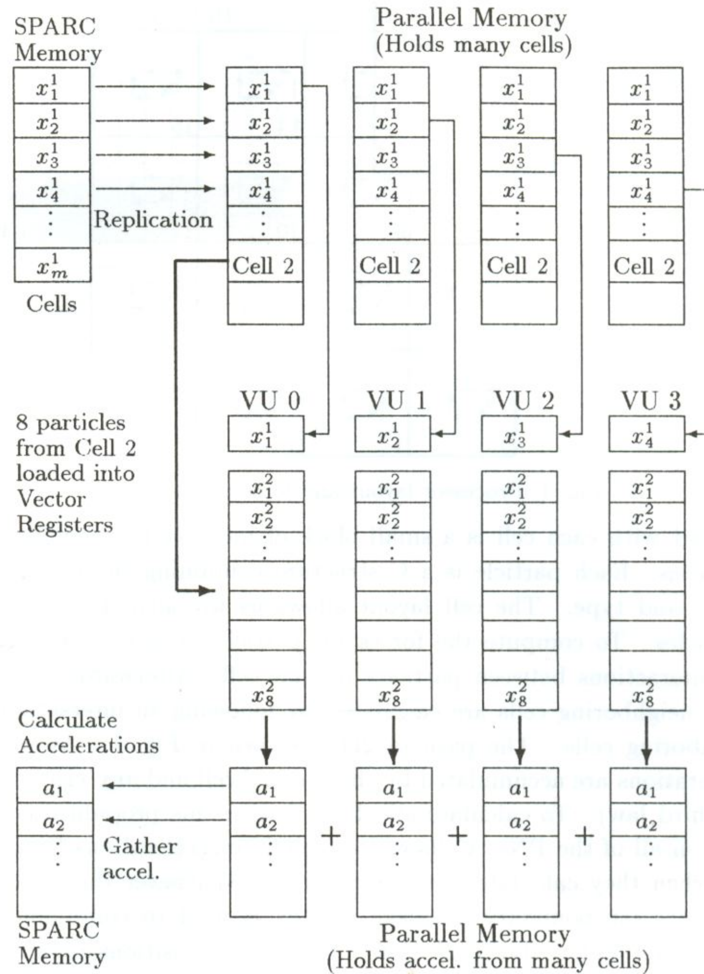

Fig. 2. Calculating forces on the VUs.

32 Mbytes of memory on each PN can be accessed by both the SPARC processor and the VUs, but it is divided into two separate areas for this purpose. SPARC memory is memory that has been allocated for use by the SPARC processor and all usual SPARC operations perform normally in this space. Parallel memory is a special memory allocation that allows all four VUs to participate in load/store operations simultaneously. Each VU allocates an identically structured memory region in the 8 Mbyte bank they control. When loads or stores are performed, each VU only accesses its particular bank. This arrangement allows the VUs to operate on four different data sets in a SIMD mode. The SPARC processor can access any particular bank of parallel memory, but access is slower than usual. The VUs can not directly access SPARC memory, but the contents of SPARC memory

can be copied to parallel memory using registers or special instructions. Accessing SPARC memory from the VUs is slow and should be avoided. As a general rule, any operations involving the VUs must use parallel memory for optimal performance.

To access the VUs, the force calculation has been implemented in CDPEAC (the assembler language for controlling the CM-5 vector units). The force calculator takes two cells of particles and calculates the resulting accelerations. The process is described in detail in Figure 1. First, particle coordinates from the two cells are copied from SPARC memory and replicated across all four VUs in parallel memory. Eight particles from cell 2 are then loaded onto all four VUs. We then loop over all of the particles in cell 1 and calculate the accelerations between these particles and the eight particles from cell 2. At each step, four different particles from cell 1 are loaded onto the VUs (a different particle on each VU). The VUs then simultaneously compute 32 interactions. Once all particles in cell 1 have been processed, the next set of 8 particles from cell 2 is loaded and the the process is repeated. The calculation continues, calculating 32 interactions per step, until all accelerations have been calculated. Afterwards, the resulting accelerations are saved back to SPARC memory.

All of the internal data structures in our code are kept in SPARC memory. While this allows the data to be easily accessible to functions that do not require the VUs, it presents a problem in our force calculation. To use the force kernel, particle coordinates must be loaded into parallel memory. One solution to this problem is to simply load particle coordinates to parallel memory whenever needed in the force calculation. Under this scheme only two cells would be loaded at any given time. Unfortunately, this approach results in a significant amount of overhead since each cell will have to be loaded to parallel memory as many as 14 times (once when calculating self interactions and 13 times when neighboring cells calculate their interactions). This effect can dramatically degrade the performance of simulations with a smaller cutoff.

To address the issue of loading parallel memory, a parallel memory caching scheme has recently been implemented. This approach attempts to eliminate excessive loading by reducing the number of times each cell is loaded to parallel memory. A buffer for holding a collection of cells is allocated in parallel memory. Each time a cell is encountered in the force calculation, this buffer is checked to see if that cell has already been loaded. If not, it is loaded to parallel memory and the previous contents saved back to SPARC memory (including accelerations). The loading process operates according to a FIFO scheme and eventually new cells will begin to replace previously loaded cells. The advantage to caching is that we only need to store a small fraction of the total number of cells in the cache. As the force calculation proceeds, previously loaded cells will no longer be necessary in the calculation (after they have remained in the cache sufficiently long) and can be saved back to the SPARC without having to be reloaded to parallel memory.

In our code, the amount of memory available for caching can be adjusted. This gives us increased flexibility since our code can be optimized for memory (by using a

Table 1: Update times per time step in sec. Cut-Off: $r_{max} = 2.5\sigma$, Cache = 25%.

	Processors					
Particles	32	64	128	256	512	1024
512000	5.98	2.88	1.57	0.91	0.54	0.34
1024000	11.18	6.02	2.95	1.70	0.94	0.54
2048000	-	11.35	6.05	3.85	1.83	0.96
4096000	-	-	12.06	7.33	3.75	1.77
8192000	-	-	-	12.79	6.68	3.03
16384000	-	-	-	24.17	14.22	5.50
32768000	-	-	-	-	23.12	9.98
65536000	-	-	-	-	-	19.14
131072000	-	-	-	-	-	38.33

Table 2: Update times per particle in μsec. Cut-Off: $r_{max} = 2.5\sigma$, Cache = 25%.

	Processors					
Particles	32	64	128	256	512	1024
512000	11.68	5.62	3.06	1.79	1.06	0.67
1024000	10.92	5.88	2.88	1.66	0.92	0.53
2048000	-	5.54	2.95	1.88	0.89	0.47
4096000	-	-	2.94	1.79	0.92	0.43
8192000	-	-	-	1.56	0.82	0.37
16384000	-	-	-	1.47	0.87	0.34
32768000	-	-	-	-	0.71	0.30
65536000	-	-	-	-	-	0.29
131072000	-	-	-	-	-	0.29

small cache) or for speed (by using a larger cache). The addition of caching has improved our code performance dramatically and even a small cache can significantly decrease the iteration time.

5. Timing and Performance

Table 1 gives the iteration time for our code with a cutoff of $r_{max} = 2.5\sigma$. All runs were performed in double precision. In all cases, the particles were arranged in a uniform 3D cubic lattice at constant density $\rho = N/\sigma^3 = 1$. This configuration is unstable and will undergo a phase change where the particles rearrange in a face-center cubic configuration. This choice of initial conditions guarantees that the particles are moving between processors and realistic inter-processor communications is involved. Except for the run with 131,072,000 particles, 25% of the cells were cached using the caching scheme described earlier.

From the Table 1 we see that the times scale linearly in terms of the number

Table 3: Timing Breakdown. 512 PNs. Cache = 30%.

	Particles	
	4096000	16384000
Computation	63%	60%
Communications	32%	35%
Parallel Memory	5%	5%

of particles and processors. The update times in Table 2 tend to improve as the number of particles increases. The run with 131,072,000 particles was a special run done with a 3% cache. With no caching, the iteration time was approximately 54 seconds, but by using the 3% cache the time drops to 38.33 seconds. This represents a speedup of nearly 30%. In this particular run, we obtain our best update time of 290 nanoseconds per particle.

To better understand the scaling properties of caching, a 3D run with 8,192,000 particles on a 1024 PN CM-5 was repeatedly run with different cache sizes. Each processing node had 512 cells arranged in an $8 \times 8 \times 8$ configuration. The iteration time versus cache size is shown in Figure 3.

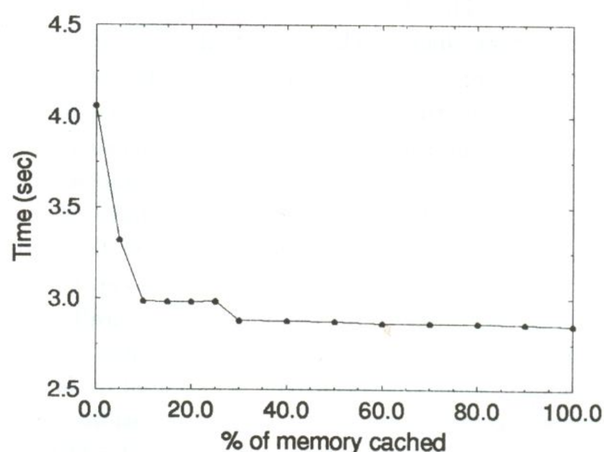

Fig. 3. Scaling of iteration time by cache size.

The iteration time drops very rapidly as the cache size is increased. Once the cache reaches approximately 30%, we obtain optimal performance. It is wasteful to increase the cache size further since the performance gained is extremely minimal. With 100% cache, all of the cells are loaded into parallel memory. This provides the best possible performance that can attained with our current force calculation scheme, but is extremely wasteful of memory since all particles must replicated across all four VUs.

In Table 3, we present a representative timing breakdown for different parts of our algorithm. Both runs were performed on a 512 PN CM-5 with 30% cached

memory. Computation time includes the force calculation and integration. Communications includes all message passing during the interaction and redistribution procedures. The parallel memory time is the time spent moving data between SPARC and parallel memory.

6. Fracture Dynamics in 2D

We have used SPaSM to perform preliminary MD simulations of fracture in a 2D LJ solid. We show the results of a tensile experiment where a piece of solid is being pulled apart at constant strain-rate. The material is a rectangular block, 2000×1000 atoms on the sides, i.e. a total of 2 million atoms. The atoms are initially at zero temperature and arranged in a hexagonal lattice configuration. The size of the block correspond a piece of material roughly $0.7\mu m \times 0.35\mu m$, which is large enough to see in a microscope. The initial velocities of the atoms were given by $(v_x, v_y) = (0.002x, 0)$ where the center of the coordinate system is at the center of the solid. The atoms thus have a linearly increasing velocity away from the center of the solid. A small notch was initially created in the material at one of the edges in order to seed a crack. In Fig. 5 we show a few snapshots from the time evolution of the simulation. The grayscale indicates the kinetic energy of the atoms, increasing from light to dark shades. The individual atoms are impossible to see in the figure, but several macroscopic phenomena are easily observed. This includes the bifurcation of the propagating crack which was initiated by the notch, shock waves released from the boundaries and dislocations generated from the crack tip and propagation into the sample generating additional sound waves. In the second picture, a substantial heating of the sample is seen to have occurred around the propagation crack. In addition, smaller cracks are seen to have opened up in the interior of the sample. At the top edge dislocations are seen to be generated and propagate into the bulk, this is due to the bending of the overall sample. In the final picture we see the interior cracks having increased in size and a substantial amount of "necking" is also seen to have have occurred, as expected in ductile materials. At this point the material has essentially failed. The dislocations generated from the initial crack tip have reached the boundary and produced additional deformation at the edges. shock waves from the dislocation propagation is also being reflected at the boundaries.

We are currently conducting a detailed study of the fracture properties of both 2D and 3D solids, including the dependency of crack length and crack speed on quantities such as applied strain rate, temperature and details of the inter-atomic potential. The fracture simulation shown in Fig. 5 is typical of what can now be performed routinely on massively parallel MIMD computers using an MD algorithm which maps closely to the underlying hardware. A more detailed account of our fracture study will be published elsewhere.

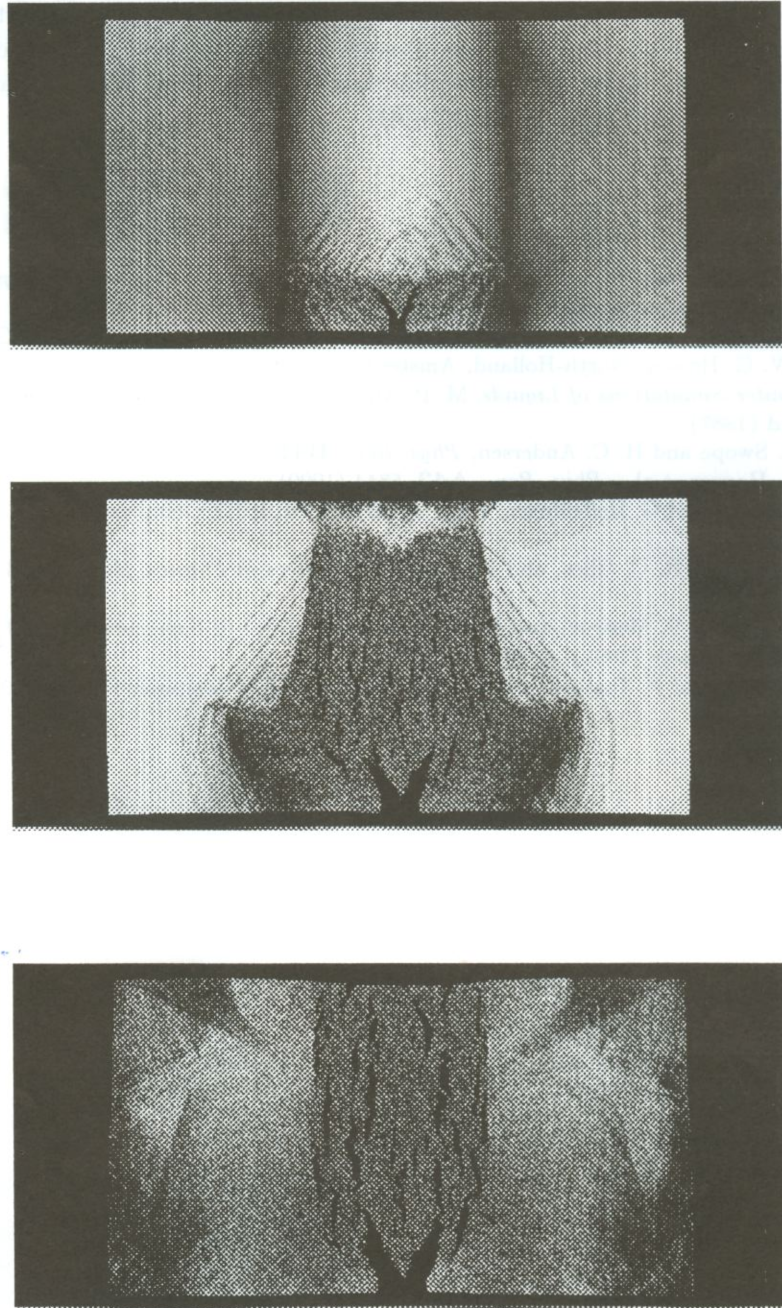

Fig. 5. Snapshots from the time evolution of 2D Fracture

Acknowledgements

We thank the Advanced Computing Laboratory for generous support, in particular M. Krogh and D. Rich provided major assistance. We also express our appreciation to D. Dahl, A. Greenberg, A. Mainwaring, and L. Tucker from TMC for valuable suggestions regarding CDPEAC and CMMD.

References

1. D. M. Beazley and P. S. Lomdahl, *Message-Passing Multi-Cell Molecular Dynamics on the Connection Machine 5*, *Parall. Comp.* (1993) (in press).
2. *Molecular Dynamics Simulations of Statistical-Mechanical Systems*, Eds. G. Ciccotti and W. G. Hoover, North-Holland, Amsterdam (1986).
3. *Computer Simulations of Liquids*, M. P. Allen and D. J. Tildesley. Clarendon Press, Oxford (1987).
4. W. C. Swope and H. C. Andersen, *Phys. Rev.* **B41**, 7042 (1990).
5. W. G. Hoover et al. , *Phys. Rev.* **A42**, 5844 (1990).
6. S. Plimpton and G. Heffelfinger, *Proc. of SHPCC'92*, IEEE Computer Society (1992), p. 246.
7. A. I. Mel'čuk, R. C. Giles, and H. Gould, *Computers in Physics*, May/June 1991, p. 311.
8. P. Tamayo, J. P. Mesirov, and B. M. Boghosian, *Proc. of Supercomputing 91*, IEEE Computer Society (1991), p. 462.
9. R. C. Giles and P. Tamayo, *Proc. of SHPCC'92*, IEEE Computer Society (1992), p. 240.