Data-parallel molecular dynamics with neighbor-lists *

Ole H. Nielsen[1,2,3]

[1] UNI•C, Technical University of Denmark, Bldg. 304, DK-2800 Lyngby, Denmark,
[2] Center for Atomic-scale Materials Physics (CAMP), Physics Dept., Technical University of Denmark, Bldg. 307, DK-2800 Lyngby, Denmark,
[3] Joint Research Center for Atom Technology (JRCAT), NAIR, 1-1-4 Higashi, Tsukuba, Ibaraki 305, Japan.

Abstract. We report on a data-parallel classical molecular dynamics algorithm and its implementation on Connection Machines CM-5 and CM-200 using CM-Fortran, a Fortran-90 dialect. A grid-based spatial decomposition of the atomic system is used for parallelization. Our previous algorithm on the CM's calculated all forces in the nearby region. A different algorithm using classical Verlet neighbor lists is more efficient, when implemented on the CM-5 with its indirect addressing hardware. The code has been used for extensive simulations on a 128-node CM-5E, and performance measurements are reported.

1 Introduction to parallel molecular–dynamics

Molecular Dynamics (MD) simulations of atomic or molecular microscopic systems are very important in several fields of science. The present project at Center for Atomic–scale Materials Physics of the Technical University of Denmark, in collaboration with Joint Research Center for Atom Technology in Tsukuba deals with the materials–physics properties of "large" metallic solid–state systems, consisting of typically 10.000 to 1 million atoms.

The basis of all MD studies is a theoretical model of the interactions between the constituent atoms. In the present work a theoretical framework called the *Effective Medium Theory*[1] (EMT) for atomic interactions has been used. In this theory, charge density "tails" from neighboring atoms are accumulated by every atom, and a non-linear functional is used to derive the interatomic energies and forces. Electrostatic energies are added to the density term, and one-electron energies may be included as well if required. Owing to the physical properties of metals, these interactions have a short range of 4–5 atomic radii. Some forms of the EMT theory resemble semi-empirical methods such as the *Embedded Atom Method*[2].

Short ranged interactions dictate the types of algorithms chosen for MD simulations: Only well localized interactions need to be taken into account, in

* To appear in proceedings of *Workshop on Applied Parallel Computing in Physics, Chemistry and Engineering Science* (PARA95), August 21-24, 1995, ed. J. Wasniewski, Springer Lecture Notes in Computer Science.

contrast to other types of systems requiring global interactions. Hence MD algorithms must be considered which allow for efficient access to atomic data in localized regions of space. A review of current research in parallel short-range MD algorithms is presented in ref. [3].

The parallel algorithm chosen in the present work is based upon a decomposition of the physical system into identical cells laid out on a regular 3–dimensional grid. Thus the algorithm only needs to consider data that reside in nearby cells on the grid. This is naturally reflected in the implementation on distributed–memory parallel computers such as the Connection Machines. This was briefly described in our previous work[4], which also reported performance measurements for CM-200 and CM-5E supercomputers.

2 The parallel Verlet neighbor-list algorithm

Even if the implementation cited above is using the Connection Machine hardware with extremely high efficiency, there is an algorithmic problem: When forces are computed between atom pairs in neighboring grid-cells, many of the forces are in fact computed for pairs whose interatomic distance is greater than the cutoff-radius of the interactions.

A kind of "algorithmic efficiency" can be estimated as the number of "relevant" non-zero interactions compared to the *total* number of calculated interactions as follows: If the cutoff-radius (or rather the radius at which the smooth cutoff-function employed in the EMT is considered to be effectively zero) is equal to the grid-cell side length D, the volume of the sphere of non-zero interactions around a given atom is $\frac{4}{3}\pi D^3$, whereas the interactions are computed inside a volume $(3D)^3$ for local $3 \times 3 \times 3$ interaction subgrids. The ratio of these volumes, which is an estimate of the algorithmic efficiency of the force computation, is approximately 0.155. If the code uses local $5 \times 5 \times 5$ subgrids and a cutoff-radius of $2 \times D$, the algorithmic efficiency is a much better 0.268. We implemented[4] a skipping of some cells in the $5 \times 5 \times 5$ subgrid, leading to about 20% improvement of this algorithmic efficiency. Larger local subgrids could be used, at the cost of more communication and a decreased number of atoms per cell. This would lead to decreased load-balance, however, so there is an upper limit to proceeding along this line.

Thus we conclude that a fairly low upper bound to the algorithmic efficiency of using local $n \times n \times n$ subgrids exists, when all interactions are computed. One possible resolution of this low efficiency is to use the classical Verlet neighbor-list method[5], but in the context of a parallel algorithm. One implementation of such an algorithm by Tamayo is discussed in ref. [3] (section 5).

We have implemented a standard Verlet neighbor-list algorithm in the context of our previously described parallel MD algorithm[4], with the aim of achieving the fastest possible execution time at the expense of using large amounts of RAM memory for storing neighbor-list related data in each parallel node.

Thus we are explicitly *not* aiming for simulating the largest possible systems, but rather on the fastest solution of more modest-sized systems (of the order of

1 million atoms). This tradeoff was decided in view of the physics that we want to investigate. Another aspect that influences algorithmic choices is the fact that our EMT interactions, which are more appropriate for simulating metallic systems, require more than 10 times the number of floating-point operations per pair-interaction, than for the case of Lennard-Jones interactions which are usually employed in MD studies. Hence communications overheads in our algorithm will be intrinsically much less significant than for the simpler Lennard-Jones interactions.

The implementation of a Verlet neighbor-list algorithm in a data-parallel CM-Fortran MD code is relatively straightforward. The task that distinguishes the data-parallel implementation from the usual serial one is the access to data in non-local processors. Our choice is to simply *copy* the data (atomic (x, y, z) coordinates, plus some interaction data) from the nearby $3 \times 3 \times 3$ local subgrid into the memory of the local processor. The `CSHIFT` Fortran-90 intrinsic is used for copying the data. This task is sketched in Fig. 1, and it is carried out whenever the atoms are moved.

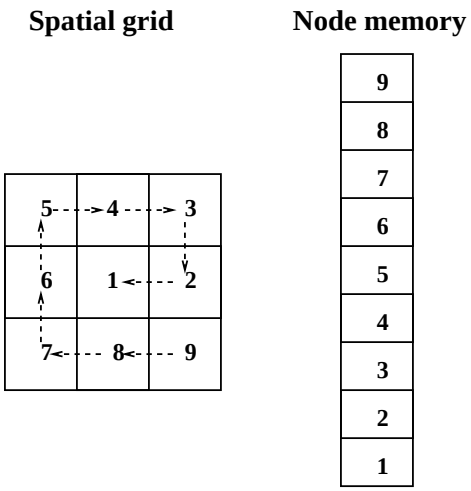

Figure 1. Copying of of 2-dimensional spatial grid data (left) onto the node's local memory (right). Data from cells 2-9 are replicated in the local memory of the node representing cell 1. In 3-D there will be a total of 27 cells in a $3 \times 3 \times 3$ subgrid.

Of course, a smaller part of the data will not represent neighbors of any atom in the central cell of Fig. 1, and hence this data could in principle be omitted. Also, it can be argued that it is a waste of memory to replicate atomic data from 26 neighboring cells. This is a deliberate choice that we have made: We want the most regular (and hence the fastest) communication pattern of data between the "vector-unit" memory of the CM-5E nodes. We also want all atomic data in a linear list in memory so that a Verlet neighbor-list algorithm can be implemented simply and efficiently. Our aim is to get the fastest possible execution at the cost of using extra memory. Since the 128-node CM-5E machine at our disposal has

128 MB of memory per node, and we want to simulate systems of only up to 1 million atoms, we can easily afford to use memory so freely.

With all relevant data available in the local memory of each node, there remains to construct the neighbor lists for each atom. Owing to all the atomic data from the central and neighboring cells being arranged in a simple linear list in local memory, it is a simple matter to traverse the list in a data-parallel fashion for each atom in the central cell and add any neighbors to a list of pointers (the *neighbor-list*). In CM-Fortran the atom-atom distance is calculated by a `FORALL` statement, and the decision whether to include the atom in the neighbor-list is implemented by a `MERGE` Fortran-90 intrinsic function. The pointers to atoms are added to the list by a "scatter" operation, which was first coded as a `FORALL` statement, but since the present CM-Fortran compiler (version CM5 VecUnit 2.2.11) hasn't implemented indirect memory access efficiently for 3-D arrays, we decided to use the faster CM-Fortran library `CMF_ASET_1D` routine in stead.

The performance-critical part of the above construction is the "scatter" operation, which requires local indirect-memory addressing that is emulated in software on the CM-200, but which has direct hardware support on the CM-5, so that the CM-5 can execute the neighbor-list construction exclusively with efficient vector operations on local memory. For the CM-200 the indirect-memory addressing was found to be too slow for a production code, so the present algorithm wasn't pursued any further on this hardware.

Since the CM-Fortran compiler generated sub-optimal code for local indirect-memory addressing, the CM-Fortran neighbor-list loop (a total of 7 Fortran statements) was translated into the CM-5 vector-unit assembler "CDPEAC", taking optimal advantage of indirect-memory addressing instructions and overlapping of computation and load/store operations. This resulted in a speedup of the neighbor-list loop by approximately a factor of 4. We also experimented with CM-Fortran "array aliasing" in order to aid the compiler's code generation, and while improvements were seen, the CDPEAC code was still more than twice as fast as the compiler's code.

After this optimization, an update of the neighbor-lists consumes a time roughly equal to the time for a single MD timestep. This means that the classical trick of increasing the cutoff-radius by adding a *skin* radius must be used in order to re-use the neighbor-list during several MD timesteps. We monitor exactly the atoms' deviations from their positions at the time of the neighbor-list construction, and when the maximum deviation exceeds half of the skin radius, an update of the neighbor-list is carried out. We believe that it is important not to postpone updates beyond this point, since the MD simulations would then not conserve energy. Atoms will migrate between cells during the simulation, and such migration-updates are performed prior to the neighbor-list updates.

When atomic forces are calculated and summed in a loop over the atoms in the neighbor-list, a simple parallel "gather" operation of the neighbor-atoms' data is performed using `FORALL` on the data structure sketched in Fig. 1. When the atomic data are in CM-5 vector-registers, the forces can be calculated by the EMT method as in ref. [4]. Again, the efficiency of the force computation depends

partially on the compiler-generated code for indirect-memory access, but less so than for the neighbor-list generation because many floating-point operations are executed for each "gather" operation. We have considered CDPEAC coding of the "gather" operation, but so far this hasn't been done.

3 Performance measurements on CM-5E

The code described above, being a relatively limited extension of the code in ref. [4], was used for large-scale production on the 128-node CM-5E at the JRCAT computer system in Tsukuba. Scientific results of using the code for studies of kinematic generation of dislocations in multi-grained copper crystals under strain have been reported in ref. [6].

Here we present performance results for this code on a typical dataset from ref. [7], which can be compared to the results of our previous paper[4]. A copper cluster with 178.219 atoms was simulated using a spatial grid of $16 \times 16 \times 16$ cells. A near-melting-point temperature of about 1400 K was used as a rather difficult test-case, giving rise to significant diffusion of atoms and hence frequent updates of the neighbor-list and migration of atoms between cells.

The performance of the new algorithm compared to the previous one[4] is shown in Table 1:

Part of the code	Present work	Old work (ref. [4])
Atomic forces kernel	0.91	6.52
Kernel: communication	0.15	1.46
Atomic rearrangement	0.35*	0.50
Neighbor-list update	1.39*	
Parallel disk I/O	0.10	0.11
Total	≈1.8*	8.62

Table 1. Elapsed time (seconds) for a single MD time-step, as measured on a 128-node CM-5E.
(* updates only performed every ≈3 timesteps, see the text below)

Since the algorithms with/without neighbor-lists differ significantly, one cannot compare all items in the table directly. However, the **atomic forces kernel** is now faster by a factor of more than 7 owing to the introduction of neighbor-lists. The kernel's computational speed is 5.1 GFLOPS, compared to the 11.7 GFLOPS for the "old" code[4], the difference being mainly due to ineffective code for indirect memory addressing through `FORALL` statements. Thus, the old code was more efficient, but due to the algorithm used, it still performed much less useful work per unit of time than the code implementing the present algorithm. The 5.1 GFLOPS corresponds to 25% of the CM-5E vector-units' theoretical peak performance, and this a very good absolute performance number.

The parallel load-balance due to fluctuations in atomic density is about the same for the new and old algorithms, and presents no problem for the present

metallic solids. The system simulated in this work contained a significant fraction of cells containing vacuum (no atoms), whereas the present grid-based algorithms are more efficient for "full" systems such as bulk materials. There are imbalances in the present system's neighbor-lists: A neighbor-list contains up to about 83 atoms with an average of 68 atoms, when a skin-radius of 10% of the cutoff-radius is used. In a data-parallel code the longest neighbor-list determines the loop-length, so we have only about 80% load-balance here.

The **kernel: communication** item in Table 1 accounts for the data copying sketched in Fig. 1, constituting only 14% of the total force computation time. The **parallel disk I/O** using the CM-5E parallel SDA disks is a negligible part of the MD timestep.

The combined **atomic rearrangement** and **neighbor-list** updates require a time of about 1.6 times that of the the force computation. Fortunately, the neighbor-list is usually performed every 5-20 MD timesteps. In the present system at near-melting, however, updates are done about every 3 MD timesteps. This leads to an average MD timestep of about 1.8 seconds, or almost 5 times faster than the "old" algorithm for the same problem. For lower temperatures the neighbor-list updates would be less frequent, and the algorithmic speedup would be larger.

4 Conclusions

The present work reports on a data-parallel molecular dynamics algorithm using Verlet neighbor-lists for the local force computations, which achieves 5.1 GFLOPS on a 128-node CM-5E. The force computation is as efficient as any serial implementation would be, except for a 20% reduction due to variations in neighbor-list length. The parallel code does local indirect memory addressing less than optimally due to compiler shortcomings, so truly optimum efficiency would require an improved compiler or a recoding in CDPEAC assembler for the CM-5. For the case of neighbor-list construction we used the latter approach. The construction of neighbor-lists takes about 1.6 times the force computation, but it only needs to be carried out after several MD timesteps.

The final load-balancing issue is related to distributing the system in a uniform grid, but for the case of fairly homogeneous solid systems such as the metals we're studying this issue is not significant.

The parallel Verlet neighbor-list code is about 5 times faster than our previous parallel code, which did not emply neighbor-lists.

In conclusion, we only need the CM-Fortran compiler to be able to generate efficient code for local indirect-memory addressing on the CM-5E vector-units in order for our data-parallel CM-Fortran code to be optimally efficient. If this were achieved, the code's performance should be as high as that of any serial neighbor-list algorithm, except for the minor load-imbalances discussed above.

Our CM-Fortran code is not restricted to Connection Machines, but should be portable with a limited effort to High-Performance Fortran (HPF) compilers that support the `FORALL` statement.

5 Acknowledgments

Dr. Jakob Schiøtz provided important feedback in the optimization process of the present CM-Fortran code. We are grateful to Palle Pedersen of Thinking Machines Corp. for CDPEAC assembler optimization of the CM-Fortran neighborlist subroutine. CAMP is sponsored by the Danish National Research Foundation. The author would like to acknowledge computing facilities at JRCAT Supercomputer System in Tsukuba, Japan, as financially supported by New Energy and Industrial Technology Development Organization (NEDO).

References

1. K. W. Jacobsen, J. K. Nørskov, and M. J. Puska, Phys. Rev. B **35**, 7423 (1987); K. W. Jacobsen, Comments Cond. Mat. Phys. **14**, 129 (1988).
2. S. M. Foiles, M. I. Baskes, and M. S. Daw, Phys. Rev. B **33**, 7983 (1986)
3. David M. Beazley, Peter S. Lomdahl, Niels Grønbech-Jensen, Roscoe Giles, and Pablo Tamayo, *Parallel Algorithms for Short-Range Molecular Dynamics*, to appear in Annual Review of Computational Physics **3**, Ed. D. Stauffer (World Scientific, 1995). Available on Internet at `URL:ftp://think.com/users/tamayo/md`
4. Ole H. Nielsen, *Molecular dynamics: experiences with Connection Machines CM-200 and CM-5*, in proceedings of *Workshop on Parallel Scientific Computing (PARA94)*, ed. J. Wasniewski, (Springer-Verlag, Berlin, 1994), Lecture Notes in Computer Science, vol. 879.
5. L. Verlet, Phys. Rev. **159**, 98 (1967).
6. J. Schiøtz, K. W. Jacobsen and O. H. Nielsen, *Kinematic generation of dislocations*, Phil. Mag. Lett., in press.
7. O. H. Nielsen, J. P. Sethna, P. Stoltze, K. W. Jacobsen and J. K. Nørskov, *Melting a copper cluster: Critical droplet theory*, Europhysics Letters, **26**, 51 (1994).

This article was processed using the LaTeX macro package with LLNCS style