Efficient parallelization of molecular dynamics simulations with short-ranged forces

Ralf Meyer

Department of Mathematics and Computer Science and Department of Physics, Laurentian University, 935 Ramsey Lake Road, Sudbury, Ontario P3E 2C6, Canada

E-mail: rmeyer@cs.laurentian.ca

Abstract. Recently, an alternative strategy for the parallelization of molecular dynamics simulations with short-ranged forces has been proposed. In this work, this algorithm is tested on a variety of multi-core systems using three types of benchmark simulations. The results show that the new algorithm gives consistent speedups which are depending on the properties of the simulated system either comparable or superior to those obtained with spatial decomposition. Comparisons of the parallel speedup on different systems indicates that on multi-core machines the parallel efficiency of the method is mainly limited by memory access speed.

1. Introduction

Molecular dynamics (MD) is a particle based simulation technique that is widely used in many areas of computational science [1, 2, 3]. As computers became more and more powerful, simulations of larger and more complex systems became possible. Yet there are still many problems that can only be solved by even more powerful computers.

For the last years, increases of computing power were achieved through the integration of a growing number of processing units on so-called multi-core chips. In the near future processors with hundreds or thousands of CPU cores will become available. The power of such processors, can only be harnessed through highly parallel algorithms that achieve good parallel efficiencies for large numbers of threads.

Large-scale MD simulations in materials science frequently use short-ranged force models that are constructed in such a way that the force between two particles is zero if the distance between the particles exceeds a cutoff radius r_{cut}. A standard method for the parallelization of MD simulations with short-ranged forces is spatial decomposition [4]. In this method, the volume of the simulated system is subdivided into as many subvolumes as there are processors. Each processor then calculates the forces between particles in one of the subvolumes.

Spatial decomposition as a parallelization strategy for MD simulations has the advantage that it can be used on SMP machines as well as distributed systems using message passing techniques. The method scales well if the simulated system is large enough that the communication overhead can be neglected and sufficiently homogeneous to avoid load-balancing issues. The latter condition, however, becomes difficult to fulfill in simulations of complex nanosystems. If the simulated system contains a substantial amount of void volume or if multiple materials with different types of interactions are present, the even distribution of the computational workload among the processors becomes challenging.


Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

In a recent article, a new algorithm for the parallelization of MD simulations on SMP systems has been proposed [5]. The *cell task* method uses task-based programming which is a modern approach to parallel programming that is particularly suited for irregular algorithms [6]. In this approach, the problem is divided into small work units (tasks) without regard for low-level details like the number of processors. A task-managing software then executes the tasks with the help of a thread pool. While a task is running it can create new tasks that are added to the list of tasks. If there are much more tasks then processors, a dynamic assignment of the tasks to the threads masks differences in the execution time of the tasks and keeps all processors busy.

In this article the efficiency of the cell task method is tested on a variety of different multi-core systems.

2. The cell task algorithm

The cell task method was designed with two goals in mind. First, the method should be highly parallel so that it performs well on many-core systems with possibly hundreds or thousands of threads. Second, the algorithm should provide similar speedups for systems of the same size independent of the homogeneity of the configuration.

The computationally most intensive part of any MD simulation is the calculation of the forces on the particles. When using short-ranged forces the algorithm for the force calculations has typically the following structure:

for all particles p:
  for all neighbors n of particle p:
    Calculate force $\mathbf{F}_{n-p}$ between particles n and p.
    Add $\mathbf{F}_{n-p}$ to total force on particle p.
    Add $-\mathbf{F}_{n-p}$ to total force on particle n.
  end for
end for

In this algorithm, the neighbors of a particle are other particles within the cutoff radius r_{cut}. For each particle the program maintains a list of its neighbors that is updated in regular intervals. In order to save time, the algorithm exploits Newton's third law which says that $\mathbf{F}_{n-p} = -\mathbf{F}_{p-n}$ and assumes that each pair of neighbors appears only in the neighbor list of one of the two particles.

On an SMP machine, the outer loop of the algorithm given above can easily be executed in parallel using for example OpenMP [7, 8]. Unfortunately, this simple strategy is not efficient. The reason for this is the need for synchronization of the force updates in the inner loop. This is necessary to avoid race conditions when multiple processors update the same particle simultaneously. In addition to this, uncoordinated accesses to the same particle by different processors can increase the amount of cross-socket transfers of cache lines.

Most programs for MD simulations with short-ranged forces employ the linked-cell technique [1] during the construction of the neighbor lists. The linked-cell technique subdivides the simulation cell into a grid of small cells and bins the particles into these cells. The size of the cells is chosen to be just larger than the cutoff radius of the interactions r_{cut}. The cell information can then be used to reduce the number of candidates for the neighbors of a particle. If a particle is in a given cell, its neighbors are all within the 27 immediately surrounding cells.

The cell task method reuses the information of the linked-cell technique to avoid synchronization constructs in the inner loop of the force calculation. If a processor calculates the forces for a particle p in the algorithm given above, it is known that all neighbors n will be found in the 27 cells surrounding the particle. Therefore, if it can be guaranteed that no other processor updates the particles in these 27 cells, no synchronization of the particle updates in

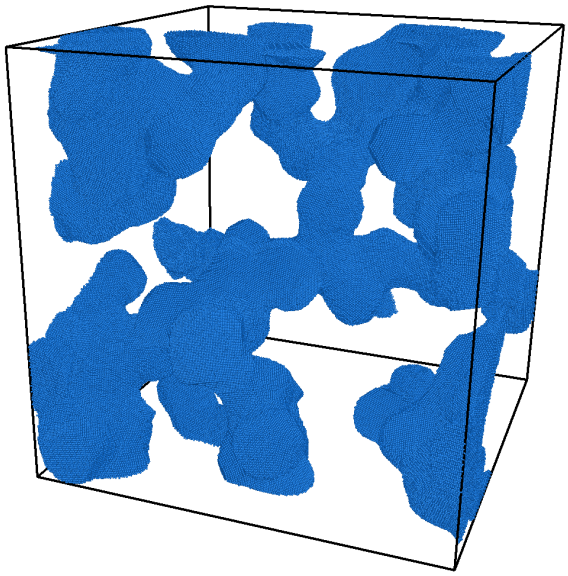

Figure 1. Configuration of the partially sintered nanocrystalline copper system

the inner loop is necessary. The other processors are however free to work on particles in other cells.

The cell task method organizes the force calculation into tasks. A task consists of the calculation of the forces on all particles within one grid cell (or a compact block of cells). The tasks are then organized in a dependent task schedule that ensures that simultaneously running tasks will only access non-overlapping sets of cells. A similar scheme is employed for the generation of the neighbor lists.

The current implementation of the cell task method is build on top of a general purpose MD program named `mdntp` [11]. This program has been designed for large scale simulations using many-body interactions like embedded-atom method [12] or Finnis-Sinclair style potentials [13]. The cell task version of `mdntp` uses Intel's *Thread Building Blocks* Library [9, 10] for the management of the tasks. Details of the construction of the task schedule and the implementation of the method can be found in [5].

3. Results from benchmark simulations

3.1. Benchmark Systems

In order to test the efficiency of the cell task method, three types of benchmark simulations have been repeated on a variety of multi-core systems. All simulations employed Cleri and Rosato's tight-binding second-moment potential for copper [14].

The first benchmark configuration is a cubic block of crystalline bulk copper build from $63 \times 63 \times 63$ fcc unit cells (1,000,188 atoms). This system is very homogeneous and therefore nearly ideal for the spatial decomposition method. Load imbalances occur in this system only if the spatial decomposition is incommensurable with the number of lattice planes.

The second benchmark system is a spherical copper nanoparticle with a diameter of 30 nm containing 1,177,151 atoms. Since this system does not fill the space completely, spatial decomposition becomes problematic if the decomposition does not cut the system into equal parts. Good performance of the spatial decomposition method can be expected for this system only for 2, 4, and 8 processors where an efficient decomposition is possible by cutting the sphere repeatedly in halves.

The last benchmark configuration is a porous system of nanocrystalline copper that was obtained from simulations of the sintering of copper nanoparticles (1,992,220 atoms; see figure 1). This system is very inhomogeneous with an irregular shape and a large fraction of void volume.

A good load balance can not be expected from a simple decomposition of this system.

All simulations were repeated five times and the average execution time (excluding initialization and saving of results at the end) was calculated from the five runs. The execution times were then used to calculate the parallel speedup with respect to the serial version.

Nearly direct comparisons between the parallel efficiency of spatial decomposition and the cell task method are possible since the original version of mdntp supports spatial decomposition. However the implementation of the cell task method in the program required some changes to the force calculation that lead to additional optimizations that are not present in the original version and that can not easily be imported into the spatial decomposition version of the program. For this reason two different serial programs were used for the calculations of parallel speedups. Parallel speedups $S(p)$ of the spatial decomposition method were calculated with respect to the execution time of the original serial program version. Speedups of the cell task method were calculated by dividing the execution time of the cell task version of the program running with 1 thread by the execution time of the same program running with p threads. This procedure effectively eliminates the effect of the additional optimizations that have nothing to do with the parallelization method.

3.2. Comparison with spatial decomposition

Figure 2 shows the speedups obtained for the three benchmarks system with the cell task method and spatial decomposition on three different multi-core machines. The figure shows that except for very few points in panel (g) the cell task method obtains better parallel speedups than the spatial decomposition method. As expected, the two methods give similar speedups in case of the homogeneous bulk system (left column). For the other two benchmark configurations the cell task method performs significantly better.

It can also be seen from figure 2 that the cell task algorithm yields considerably more consistent speedups. The speedups obtained by the cell task method increase continuously with the number of threads and on the same machine the behavior of the three benchmark systems is rather similar. In contrast to this the speedup of the spatial decomposition method is non-monotonous and the results vary from one benchmark system to another on the same machine.

Finally, figure 2 shows a growing deviation of the speedup from the ideal behavior. It is however unlikely that this loss of efficiency is due to a lack of parallelism of the cell task algorithm. If this were the reason, the spatial decomposition method should outperform the cell task method at least in the case of the bulk system since spatial decomposition scales well for large homogeneous systems. The left column of figure 2 shows however that for the bulk system the spatial decomposition method deviates from the ideal behavior in the same manner as the cell task method. This is a strong indication that the deviations are not caused by the parallel algorithms but by limitations of the test machines. In addition to this, benchmark data given in reference [5] show that the cell task method achieves parallel efficiencies above 80 % at $p = 60$ threads on a Xeon Phi coprocessor. The most likely reasons for the lower parallel efficiency on the multi-core machines at high numbers of threads are limitations of the memory system. This will be discussed in the next section.

3.3. Comparison between test systems

Table 1 presents for a variety of multi-core machines the execution times $t(1)$ of the cell task program with one thread and the speedups obtained with $p = 6$, 12, 16, 24 and 32 threads for the third benchmark system (porous nanocrystalline copper). A surprising observation is that despite the strong variations between the test machines, the speedup data are rather similar. Unfortunately most of the systems differ in more than one parameter. This makes it difficult to identify the parameters with the largest impact on the parallel performance.

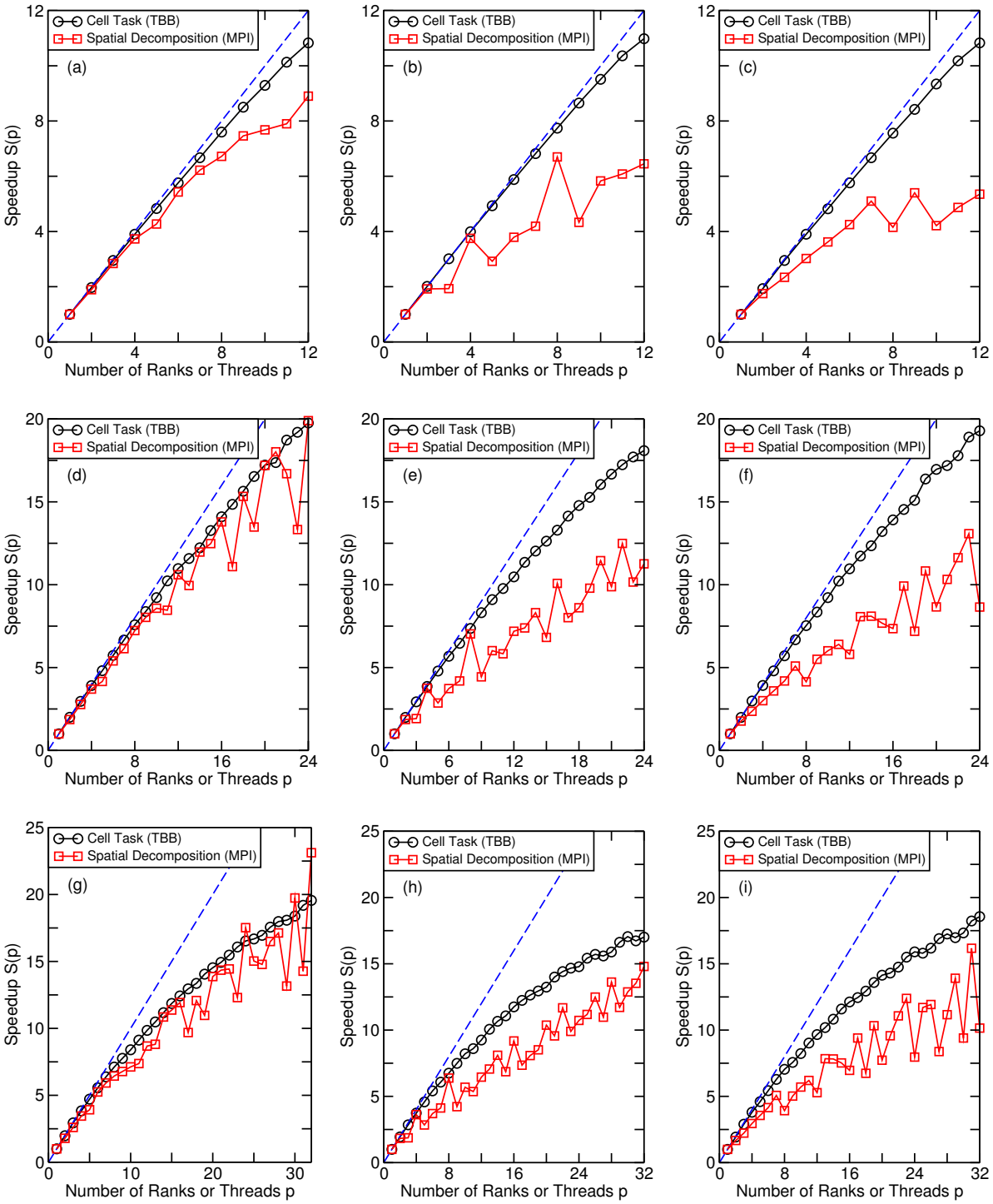

Figure 2. Parallel speedups obtained by the cell task method and spatial decomposition for the three benchmark systems bulk copper (left), copper nanoparticle (center) and porous nanocrystalline copper (right) on three test machines 12 core Xeon E5-2630 (top), 24 core Opteron 6174 (middle) and 32 core Opteron 8354 (bottom). Dashed blue lines indicate the ideal speedup of 1 per processor.

Table 1. Serial execution times $t(1)$ and speedups $S(p)$ for $p = 6$, 12, 16, 24 and 32 threads of the cell task method for the porous nanocrystalline benchmark system.

CPU Family	Xeon	Xeon	Xeon	Opteron	Opteron	Opteron	Xeon
CPU Model	X-5650	E5-2630	E-7340	6174	6238	8354	E5-4620
Cores	2×6	2×6	4×4	2×12	2×12	8×4	4×8
Clock Freq. [GHz]	2.7	2.3	2.4	2.2	2.6	2.2	2.2
Memory	DDR3 1333	DDR3 1600	DDR2 667	DDR3 667	DDR3 1600	DDR2 533	DDR3 1067
t(1) [s]	1030.2	1150.2	1772.1	1730.3	1305.3	1891.7	1020.4
S(6)	5.3	5.8	5.6	5.7	5.7	5.4	5.0
S(12)	9.7	10.8	10.4	10.9	10.5	9.7	9.3
S(16)			13.4	13.9	12.6	12.1	11.7
S(24)				19.2	15.8	15.9	15.4
S(32)						18.6	18.2

The two most similar systems in table 1 are probably the two Xeon E5 machines. The CPUs of these two systems have the same microarchitecture and very similar clock frequencies. Comparison of the two systems shows that better speedups are obtained on the system with faster memory. Another trend indicated by table 1 is that a low serial execution speed favors higher speedups at larger numbers of threads. This too is consistent with the idea that the parallel efficiency is mainly limited by the memory system since a slower program execution reduces the impact of memory access speed.

4. Summary and conclusions

The cell task algorithm provides an alternative parallelization method for MD simulations on multi- and many-core systems. The method is designed to be highly parallel and to give high speedups even for strongly inhomogeneous systems where spatial decomposition becomes less effective.

Benchmark calculations using three types of configurations on a variety of systems show that for inhomogeneous systems the cell task method yields a better parallel efficiency than spatial decomposition. For homogeneous systems the speedups are comparable.

On the multi-core systems used in this work, the parallel efficiency of the cell task method diminishes for larger numbers of cores. The reason for this is not yet fully understood. The fact that the spatial decomposition method is affected in a similar manner points however in the direction of system limitations rather than insufficient parallelism.

Although the data in table 1 do not allow to draw definite conclusions, they are compatible with the hypothesis that the main factor limiting the parallel efficiency of the MD simulations on the multi-core machines is memory access speed. More work is required to verify this hypothesis. One way to do this would be to perform simulations with a computationally more intensive potential. If the parallel efficiency in the current work is indeed bound by memory access speed, higher efficiencies can be expected for a potential that requires more computations per particle pair.

Acknowledgments
I wish to thank H. Merz from SHARCNET for his help to obtain the technical details of some of the computers used for the benchmarks. Financial support by Laurentian University and the Natural Science and Engineering Council of Canada (NSERC) is gratefully acknowledged. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca) and Compute/Calcul Canada.

References
[1] Allen M P and Tildesley D J 1987 *Computer Simulations of Liquids* (Oxford: Clarendon)
[2] Frenkel D and Smit B 2002 *Understanding Molecular Simulation* (San Diego: Academic)
[3] Rapaport D C 2004 *The Art of Molecular Dynamics Simulation* (Cambridge: Cambridge University Press)
[4] Plimpton S 1995 *J. Comp. Phys.* **117** 1–19
[5] Meyer R 2013 (*Preprint* `arXiv:1305.4196`)
[6] Korch M and Rauber T 2004 *Concurrency Computat.: Pract. Exper.* **16** 1–47
[7] `http://www.openmp.org/`
[8] Chapman B, Jost G and van der Pas R 2008 *Using OpenMP* (Cambridge: MIT Press)
[9] `http://threadingbuildingblocks.org/`
[10] James R 2007 *Intel Threading Building Blocks: Outfitting C++ For Multi-Core Processor Parallelism* (Sebastopol: O'Reilly Media)
[11] Meyer R 1998 Ph.D. thesis Gerhard-Mercator-Universität Duisburg Germany
[12] Daw M S and Baskes M I 1984 *Phys. Rev.* B **29** 6443–6453
[13] Finnis M W and Sinclair J E 1984 *Phil. Mag.* A **50** 45
[14] Cleri F and Rosato V 1993 *Phys. Rev.* B **48** 22–33