Molecular Dynamics on NEC Vector Systems

Katharina Benkert[1] and Franz Gähler[2]

[1] High Performance Computing Center Stuttgart (HLRS)
University of Stuttgart
70569 Stuttgart, Germany
benkert@hlrs.de
http://www.hlrs.de/people/benkert

[2] Institute for Theoretical and Applied Physics (ITAP)
University of Stuttgart
70550 Stuttgart, Germany
gaehler@itap.physik.uni-stuttgart.de
http://www.itap.physik.uni-stuttgart.de/~gaehler

Abstract. Molecular dynamics codes are widely used on scalar architectures where they exhibit good performance and scalability. For vector architectures, special algorithms like Layered Link Cell and Grid Search have been developed. Nevertheless, the performance measured on the NEC SX-8 remains unsatisfactory. The reasons for these performance deficits are studied in this paper.

Keywords: Molecular dynamics, Vector architecture

1 Introduction

The origins of molecular dynamics date back to 1979 when Cundall and Strack [1] developed a numerical method to simulate the movement of a large number of particles. The particles are positioned with certain initial velocities. The relevant forces between the particles are summed up and Newton's equations of motion are integrated in time to determine the change in position and velocity of the particles. This process is iterated until the end of the simulation period is reached. For molecular simulations the particles only interact with nearby neighbors, so usually a cut-off radius delimits the interactions to be considered.

Since this time, the method has gained an important significance in material science. The properties of metals, ceramics, polymers, electronic, magnetic and biological materials can now be studied to understand material properties and to develop new materials. This progress has been made possible by the construction of accurate and reliable interaction potentials for many different kinds of materials, the development of efficient and scalable parallel algorithms, and the enormous increase of hardware performance. It is now possible to simulate multi-million atom samples over time scales of nanoseconds on a routinely basis, an application which clearly belongs to the domain of high performance computing. Such system sizes are indeed required for certain purposes, e.g. for the simulation of crack propagation [2] or the simulation of shock waves [3].

For these and similar applications with high computing requirements, NEC and the High Performance Computing Center Stuttgart (HLRS) formed the Teraflop Workbench [4], a public-private partnership to achieve TFlops sustained performance on the new 72 node SX-8 installation at HLRS.

In this paper, the differences in the implementation of a molecular dynamics program on scalar and vector architectures are explained and an investigation of performance problems on the NEC SX-8 is presented.

2 Implementing molecular dynamics simulations

The dynamics of a system of particles is completely determined by the potential energy function U of the system, shortly denoted as potential. Using Newton's law, the force F_i acting on an atom i is equal to $-\nabla_i U$. These equations are then integrated to retrieve the trajectories of the atoms in course of time. The potential can be simply modeled as an empirical pair potential such as the Lennard-Jones potential, but many systems require more elaborate potential models. For metals, so-called EAM potentials [5, 6] are widely used:

$$E = \sum_{i,j} \Phi_{ij}(r_{ij}) + \sum F_i(\rho_i)$$

where

$$\rho_i = \sum_j \Psi_j(r_{ij}).$$

Although being a many-body potential, EAM potentials are as easy to compute as pair potentials.

For short-range interactions, only particles having a distance smaller than the cut-off radius r_c are assumed to contribute to the forces. The algorithmic problem is to construct a parallelizable algorithm scaling linearly with system size to find interacting atom pairs quickly. Testing all possible combinations results in an $O(N^2)$ algorithm, where N is the number of atoms. A first step to reduce the computational effort is the use of Verlet lists [7]: all particles having a distance smaller than $r_c + r_s$, where r_s is the so-called skin, are saved to temporary lists. As long as one of the particles has not moved more than $\frac{r_s}{2}$, those lists can be used for the computation of the potential. To finally obtain an $O(N)$ algorithm, a grid with cells having side lengths slightly greater than $r_c + r_s$ is introduced. At first, the particles are inserted into the cells, and then, in a second step, the Verlet lists are constructed by considering only particles in the surrounding cells, resulting in the Link Cell (LC) method of Quentrec et al. [8] described well in Allen and Tildesley [9]. Parallelization is easily realized using geometric domain decomposition with additional buffer cells [10].

2.1 Implementation on scalar architectures

On scalar architectures, the Verlet lists are implemented as two lists, one having pointers into the other list, which in turn contains all particles with distances smaller than $r_c + r_s$. The implementation of the kernel, comprising the calculation and update of the forces, is straightforward. To achieve better cache-usage, all information local to a cell, e.g. particle positions, distances and Verlet lists, can be stored together. Although this introduces an extra level of indirection, execution times decrease.

2.2 Implementation on vector architectures

In contrast to scalar architectures, which depend on effective cache usage, vector architectures use pipelining to achieve high performance. Therefore, vector arithmetic instructions operate efficiently on large, independent data sets. Standard molecular dynamics codes are not well suited for vector architectures. Frequent if-clauses, e.g. when deciding whether particles interact or not, and short loop lengths over all particles that interact with a given one prohibit successful vectorization.

For this reason, new algorithms like Layered Link Cell (LLC) [11] and Grid Search (GS) [12] were developed which both use vectorization over all cells instead of vectorization over all particles within one cell. The performance of these algorithms on the NEC SX-8 has been investigated in [13]. Analogously to the LC algorithm, LLC uses cells with side lengths slightly greater than $r_c + r_s$ allowing several particles in one cell. The GS algorithm uses a finer grid with only one particle per cell, which simplifies vectorization, but complicates the choice of an optimal cell length and the distribution of particles into cells. Its runtime compared to LLC is generally lower since more advanced techniques like Neighbor Cell Assignments and Sub-Cell Grouping are used. The Verlet lists are organized as two lists, saving every particle pair whose distance is smaller than $r_c + r_s$.

2.3 The molecular dynamics program IMD

IMD [14] is a software package for classical molecular dynamics simulations developed using C. It supports several types of interactions, like central pair potentials, EAM potentials for metals, Stillinger-Weber and Tersoff potentials for covalent systems, and also more elaborate many-body potentials like MEAM [15] or ADP [16]. A rich choice of simulation options is available: different integrators for the simulation of the various thermodynamic ensembles, options that allow to shear and deform the sample during the simulation, and many more.

Its main design goals were to create a flexible and modular software achieving high performance on contemporary computer architectures, while being as portable as possible. Preprocessor macros allow to switch between scalar and vector versions of the code.

The performance of IMD on several architectures is shown in Table 1. On the SX-8, IMD implements the LLC algorithm. The "mono" option limits calculations to one atom type by hard-coding the atom type as zero. On the SX-8, this gives a considerable performance improvement. In order to allow for maximal flexibility, potentials are specified in the form of tabulated functions. For the pair potential, a Lennard-Jones potential was used. It can clearly be seen, that the price/performance ratio of IMD on vector architectures is dissatisfying.

Table 1. Timings for IMD in μs per step per atom for a sample with 128k atoms

Machine, compiler	**pair**	**EAM**
SX-8, mono, sxf90	1.93	2.73
SX-8, sxf90	2.16	3.68
Itanium2, 1.5 GHz, icc	2.58	5.05
Opteron, 2.2 GHz, icc	4.41	6.59
Xeon 64bit, 3.2 GHz, icc	4.64	7.44

3 Performance of the test kernel

To better understand the problems of molecular dynamics simulations on the NEC SX-8, a test kernel using the GS algorithm was implemented using Fortran 90.

As test case, a fcc crystal with 131k atoms was simulated for 100 time steps using a calculated Lennard-Jones potential. All following tables show extracts of performance analyses using the flow trace analysis tool *ftrace*. Since the usage of *ftrace* did hardly influence the execution time, statistical profiling results are not included in this paper.

The column "PROG. UNIT" displays the name of the routine or region, "FREQ." gives the number of times a routine was called. "EXCLUSIVE TIME" is the total time spent in the routine and it does not include time spent in other routines called by it. "MFLOPS" depicts the performance in millions of floating point operations per second. The vector operation ratio, i.e. the ratio of vector elements processed to the total number of vector operations, and the average vector length are given in the columns "V.OP RATIO" and "AVER. V.LEN", respectively. These metrics state which portion of the code has been vectorized and to what extent. The average vector length is bounded by the hardware vector length of 256. The time spent waiting until banks recover from memory access is given in the column "BANK CONFLICT".

Table 2 clearly illustrates that nearly all time is spent during force calculation. Although major portions of the force calculation are vectorized and possess a good average vector length of 225.8, the performance of 3.7 GFlops is unsatisfactory.

Update times per step per atom are 0.860μs. As IMD shows only a modest performance difference between tabulated and calculated Lennard-Jones potentials, this can be compared with the results of Table 2, which shows that the Fortran kernel using GS is about twice as fast as IMD using LLC.

Table 2. *Ftrace* performance output for the kernel

PROG. UNIT	**FREQ.**	**EXCLUSIVE TIME[sec](%)**	**MFLOPS**	**V.OP RATIO**	**AVER. V.LEN**	**BANK CONF**
total	113	11.336 (100.0)	3729.1	99.80	225.8	0.1199
forcecalculation	100	11.247 (99.2)	3717.7	99.81	225.8	0.1185

The structure of the kernel is divided into three parts: the construction of the lists of interacting particle pairs and at times the update of the Verlet lists, the calculation of the potential, and the update of the forces.

```
if (verlet lists need to be updated) then
     - find potentially interacting particles
     - build new verlet lists
     - build lists of interacting particles and save distances in
       x, y and z direction as well as squared distance to arrays
else ! old verlet lists are used
     - find interacting particles
     - build lists of interacting particles and save distances in
       x, y and z direction as well as squared distance to arrays
end if
- calculate potential
- update forces
```

3.1 Construction and use of Verlet lists

If the Verlet lists need to be updated and there are particles at a given neighbor-cell-relation, the distances between those particles are calculated. If the distance is smaller than $r_c + r_s$, the particles need to be inserted into the Verlet lists. If the distance is also smaller than r_c, the particle numbers as well as the distances are saved to arrays for later use.

The performance characteristics of the construction of the Verlet lists are given in Table 3 and show the same behavior as those of the total kernel: although vectorization ratio and average vector length are good and the number of bank conflicts is small, the performance is low.

The key problems are the complicated loop structure with nested if-clauses and the high number of copy operations. The frequency with which the Verlet lists need to be updated depends on the skin r_s and on the amount of atomic motion. When simulating a solid, intervals between Verlet list updates are typically

Table 3. *Ftrace* performance output for construction of Verlet lists

PROG. UNIT	**FREQ.**	**EXCLUSIVE TIME[sec](%)**	**MFLOPS**	**V.OP RATIO**	**AVER. V.LEN**	**BANK CONF**
newlist	241	0.274 (2.4)	2880.2	99.71	256.0	0.0569

5 – 15 time steps, or even more when simulating at low temperature, whereas for the simulation of liquids more frequent updates may be necessary.

If the old Verlet lists are still valid, the distances between the particles have to be calculated. Those particles which actually interact are stored together with their distances into temporary arrays. The results are shown in Table 4.

Table 4. *Ftrace* performance output when old Verlet lists are used

PROG. UNIT	**FREQ.**	**EXCLUSIVE TIME[sec](%)**	**MFLOPS**	**V.OP RATIO**	**AVER. V.LEN**	**BANK CONF**
oldlist	6930	6.033 (53.2)	3613.5	99.83	225.8	0.0231

The major problems are again the high number of copy operations and the indirect access to retrieve the positions of the particles stored in the Verlet lists.

3.2 Calculation of potential

As interaction model, a calculated Lennard-Jones potential was used. Given that 16 floating point operations and only two memory operations are needed for one force evaluation, the performance of 9217.4 MFlops is not remarkable (Table 5).

Table 5. *Ftrace* performance for calculation of Lennard-Jones potential

PROG. UNIT	**FREQ.**	**EXCLUSIVE TIME[sec](%)**	**MFLOPS**	**V.OP RATIO**	**AVER. V.LEN**	**BANK CONF**
calcpotential	7171	1.220 (10.8)	9217.4	99.69	225.9	0.0002

Unfortunately, calculated potentials are not often used. For real applications, tabulated potentials fitted to reproduce results from DFT simulations are more flexible, which increases the number of memory accesses and therefore reduces the performance further.

3.3 Update of forces

During the update of the forces, the distance components in x-, y- and z-direction are multiplied by the calculated force and divided by the distance, and the result is added to the forces of the two particles.

```
    do i = 1, nInterAc
      sx(i) = sx(i) * forceOverDistance(i)
      sy(i) = sy(i) * forceOverDistance(i)
      sz(i) = sz(i) * forceOverDistance(i)
    end do

!CDIR NODEP
    do i = 1, nInterAc
      Fx   (interAcList2(i)) = Fx   (interAcList2(i)) + sx(i)
      Fy   (interAcList2(i)) = Fy   (interAcList2(i)) + sy(i)
      Fz   (interAcList2(i)) = Fz   (interAcList2(i)) + sz(i)
      Fxtmp(interAcList1(i)) = Fxtmp(interAcList1(i)) - sx(i)
      Fytmp(interAcList1(i)) = Fytmp(interAcList1(i)) - sy(i)
      Fztmp(interAcList1(i)) = Fztmp(interAcList1(i)) - sz(i)
    end do
```

As can be seen from the above code segment, heavy indirect addressing is needed which is reflected in the performance (Table 6).

Table 6. *Ftrace* performance output for force update

PROG. UNIT	**FREQ.**	**EXCLUSIVE TIME[sec](%)**	**MFLOPS**	**V.OP RATIO**	**AVER. V.LEN**	**BANK CONF**
updateforces	7171	3.669 (32.4)	2121.5	99.82	225.9	0.0378

The update of the forces is the most critical part of the total force calculation. The percentage of time spent in this routine and the low performance due to heavy indirect addressing is a major cause for the unsatisfactory total performance.

4 Summary

Molecular dynamics simulations on vector machines suffer from the latencies involved in indirect memory addressing. For our test kernel using GS, most time is spent when using old Verlet lists and updating forces, whereas simulations with IMD (using LLC) are dominated by the time spent during force updates. Since the reasons for the low performance lie within the structure of LLC and GS, an improvement can only be achieved by developing new algorithms.

5 Acknowledgments

The authors would like to thank Uwe Küster of HLRS as well as Holger Berger and Stefan Haberhauer of 'NEC High Performance Computing Europe' for their continuing support.

References

1. Cundall, P., Strack, O.: A distinct element model for granular assemblies. Geotechnique **29(1)** (1979) 47–65
2. Rösch, F., Rudhart, C., Roth, J., Trebin, H.R., Gumbsch, P.: Dynamic fracture of icosahedral model quasicrystals: A molecular dynamics study. Phys. Rev. B **72** (2005) 014128
3. Roth, J.: ω-phase and solitary waves induced by shock compression of bcc crystals. Phys. Rev. B **72** (2005) 014126
4. http://www.teraflop-workbench.de/.
5. Daw, M.S., Baskes, M.I.: Semiempirical, quantum mechanical calculation of hydrogen embrittlement in metals. Phys. Rev. Lett. **50** (1983) 1285–1288
6. Daw, M.S., Baskes, M.I.: Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. Phys. Rev. B **29** (1984) 6443–6453
7. Verlet, L.: Computer experiments on classical fluids: I. Thermodynamical properties of Lennard-Jones molecules. Phys. Rev. **159** (1967) 98–103
8. Quentrec, B., Brot, C.: New methods for searching for neighbours in molecular dynamics computations. J. Comput. Phys. (1973) 430–432
9. Allen, M., Tildesley, D.: Computer simulation of liquids. Clarendon Press (1987)
10. Plimpton, S.J.: Fast parallel algorithms for short-range molecular dynamics. J. Comput. Phys. **117** (1995) 1–19
11. Grest, G., Dünweg, B., Kremer, K.: Vectorized link cell fortran code for molecular dynamics simulations for a large number of particles. Comp. Phys. Comm. **55** (1989) 269–285
12. Everaers, R., Kremer, K.: A fast grid search algorithm for molecular dynamics simulations with short-range interactions. Comp. Phys. Comm. **81** (1994) 19–55
13. Gähler, F., Benkert, K.: Atomistic simulations on scalar and vector computers. In: Proceedings of the 2nd Teraflop Workshop, HLRS, Germany, Springer (2005)
14. Stadler, J., Mikulla, R., Trebin, H.R.: IMD: A software package for molecular dynamics studies on parallel computers. Int. J. Mod. Phys. C **8** (1997) 1131–1140 http://www.itap.physik.uni-stuttgart.de/˜imd.
15. Baskes, M.I.: Modified embedded-atom potentials for cubic materials and impurities. Phys. Rev. B **46** (1992) 2727–2742
16. Mishin, Y., Mehl, M.J., Papaconstantopoulos, D.A.: Phase stability in the Fe-Ni system: Investigation by first-principles calculations and atomistic simulations. Acta Mat. **53** (2005) 4029–4041