Atomistic Simulations on Scalar and Vector Computers

Franz Gähler[1] and Katharina Benkert[2]

[1] Institute for Theoretical and Applied Physics,
University of Stuttgart, D-70550 Stuttgart, Germany
gaehler@itap.physik.uni-stuttgart.de
[2] High Performance Computing Center Stuttgart,
Allmandring 30, D-70550 Stuttgart, Germany
benkert@hlrs.de

Abstract. Large scale atomistic simulations are feasible only with classical effective potentials. Nevertheless, even for classical simulations some ab-initio computations are often necessary, e.g. for the development of potentials or the validation of the results. Ab-initio and classical simulations use rather different algorithms and make different requirements on the computer hardware. We present performance comparisons for the DFT code VASP and our classical molecular dynamics code IMD on different computer architectures, including both clusters of microprocessors and vector computers. VASP performs excellently on vector machines, whereas IMD is better suited for large clusters of microprocessors. We also report on our efforts to make IMD perform well even on vector machines.

1 Introduction

For many questions in materials science, it is essential to understand dynamical processes in the material at the atomistic level. Continuum simulations cannot elucidate the dynamics of atomic jump processes in diffusion, in a propagating dislocation core, or at a crack tip. Even for many static problems, like the study of the structure of grain boundaries, atomistic simulations are indispensable. The tool of choice for such simulations is molecular dynamics (MD). In this method, the equations of motion of a system of interacting particles (atoms) are directly integrated numerically. The advantage of the method is that one needs to model only the interactions between the particles, not the physical processes to be studied. The downside to this is a high computational effort.

The interactions between atoms are governed by quantum mechanics. Therefore an accurate and reliable simulation would actually require a quantum mechanical model of the interactions. While this is possible in principle, in practice it is feasible only for rather small systems. Computing the forces by ab-initio density functional theory (DFT) is limited to a few hundred atoms at most, especially if many transition metal atoms with a complex electronic structure are part of the system. For ab-initio MD, where many time steps are required, the

limits are even much smaller. Due to the bad scaling with the number of atoms (N^3 for part of the algorithm), there is little hope that one can exceed these limits in the foreseeable future. Order N algorithms, which are being studied for insulators, do not seem to be applicable to metal systems.

For many simulation problems, however, systems with a few hundred atoms are by far not big enough. Especially the study of mechanical processes, like dislocation motion, crack propagation, or nano-indentation would at least require multi-million atom systems. Such simulations are possible only with classical effective potentials. These must be carefully fitted to model the quantum mechanical interactions as closely as possible. One way to do this is by force matching [1]. In this method, for a collection of small reference structures, which should comprise all typical local configurations, the forces on all particles are computed quantum-mechanically, along with other quantities like energies and stresses. The effective potentials are then fitted to reproduce these reference forces. This procedure is well known for relatively simple materials, but has successfully been applied recently also to complex intermetallics [2]. Force matching provides a way to bridge the gap between the requirements of large scale MD simulations and what is possible with ab-initio methods, thus making quantum mechanical information available also to large scale simulations.

For accurate and reliable simulations of large systems, both classical and quantum simulations are necessary. The quantum simulations are needed not only for the development of effective potentials, but also for the validation of the results. The two kinds of simulations use rather different algorithms, and have different computer hardware requirements. If geometric domain decomposition is used, classical MD with short range interactions is known to scale well to very large particle and CPU numbers. It also performs very well on commodity microprocessors. For large simulations, big clusters of such machines, together with a low latency interconnect, are therefore the ideal choice. On the other hand, vector machines have the reputation of performing poorly on such codes.

With DFT simulations, the situation is different; they do not scale well to very large CPU numbers. Among other things this is due to 3D fast Fourier transforms (FFT) which takes about a third of the computation time. It is therefore important to have perhaps only a few, but very fast CPUs, rather than many slower ones. Moreover, the algorithms do mostly linear algebra and need, compared to classical MD, a very large memory. Vector machines like the NEC SX series therefore look very promising for the quantum part of the simulations.

The remainder of this article is organized into three parts. In the first part, we will analyze the performance of VASP [3–5], the Vienna Ab-initio Simulation Package, on the NEC SX and compare it to the performance on a powerful microprocessor based machine. VASP is a widely used DFT code and is very efficient for metals, which we are primarily interested in. In the second part, the algorithms and data layout of our in-house classical MD code IMD [6] are discussed and performance measurements on different cluster architectures are presented. In the third part, we describe our efforts to achieve competitive per-

formance with classical MD also on vector machines. So far, these efforts have seen only a limited success.

2 Ab-initio Simulations with VASP

The Vienna Ab-initio Simulation Package, VASP [3–5], is our main work horse for all ab-initio simulations. In recent years, its development has been concentrated on PC clusters, where it performs very well, but the algorithms used should also perform well on vector machines. As explained above, due to the modest scaling with increasing CPU numbers it is very important to have fast CPUs available. Vector computing is therefore the obvious option to explore. For these tests an optimized VASP version for the NEC SX has been used.

As test systems, we take two large complex metal systems: $Cd_{186}Ca_{34}$ with 220 atoms per unit cell and $Cd_{608}Ca_{104}$ with 712 atoms per unit cell. In each case, one electronic optimization was performed, which corresponds to one MD step. As we explain later, the runtimes for such large systems are too big to allow for a large number of steps. However, structure optimizations through relaxation simulations are possible. In all cases, k-space was sampled at the Γ-point only. Two VASP versions were used: a full complex version and a specialized Γ-point only version. The latter uses a slightly different algorithm which is faster and uses less memory, but can be used only for the Γ-point. Timings are given in Tab. 1. For comparison, also the timings on an Opteron cluster are included. These timings show that the vector machine has a clear advantage compared to a fast microprocessor machine. Also the absolute gigaflop rates are very satisfying, reaching up to 55% of the peak performance for the largest system.

Table 1. Timings for three large systems on the SX8 (with SX6 executables), the SX6+, and an Opteron cluster (2GHz, Myrinet). For the vector machines, both the total CPU time (in seconds) and the gigaflop rates are given.

	SX8		SX6+		Opteron
	time	GF	time	GF	time
712 atoms, 8 CPUs, complex	47256	70	88517	38	
712 atoms, 8 CPUs, Γ-point	13493	57	20903	36	70169
220 atoms, 4 CPUs, complex	2696	33	5782	15	13190

The scaling with the number of CPUs is shown in Fig. 1. As can be seen, the full complex version of VASP scales considerably better. This is especially true for the SX8, which shows excellent scaling up to 8 CPUs, whereas for the SX6+ the performance increases subproportionally beyond 6 CPUs. For the Γ-point only version, the scaling degrades beyond 4 CPUs, but this version is still faster than the full version. If only the Γ-point is needed, it is worthwhile to use this version.

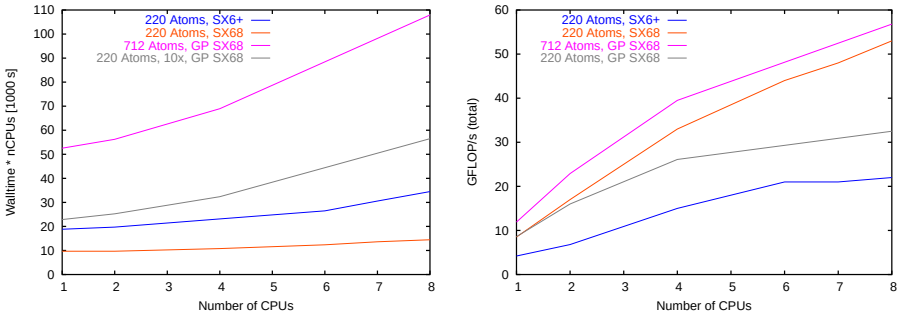

Fig. 1. Scaling of VASP for different systems on the SX8 (with SX6 executables) and the SX6+. Shown are total CPU times (left) and absolute gigaflop rates (right). The timings of the 220 atom system (Γ-point only version) on the SX8 have been multiplied by 10.

3 Classical Molecular Dynamics with IMD

For all classical MD simulations we use our in-house code, IMD [6]. It is written in ANSI C, parallelized with MPI, and runs efficiently on a large number of different hardware architectures. IMD supports many different short range interactions for metals and covalent ceramics. Different integrators and a number of other simulation options are available, which allow, e.g., to apply external forces and stresses on the sample. In the following, we describe only those parts of the algorithms and data layout, which are most relevant for the performance. These are all concerned with the force computation, which takes around 95% of the CPU time.

3.1 Algorithms and Data Layout

If the interactions have a finite range, the total computational effort of an MD step scales linearly with the number of atoms in the system. This requires, however, to quickly find those (few) atoms from a very large set, with which a given atom interacts. Searching the whole atom list each time is an order N^2 operation, and is not feasible for large systems. For moderately big systems, Verlet neighbor lists are often used. The idea is to construct for each atom a list of those atoms which are within the interaction radius r_c, plus an extra margin r_s (the skin). The construction of the neighbor lists is still an order N^2 operation, but depending on the value of r_s they can be reused for a larger or smaller number of steps. The neighbor lists remain valid as long as no atom has traveled a distance larger than $r_s/2$.

For very large systems, Verlet neighbor lists are still not good enough and link cells are usually used. In this method, the system is subdivided into cells, whose diameter is just a little bigger than the interaction cutoff. Atoms can then interact only with atoms in the same and in neighboring cells. Sorting the atoms

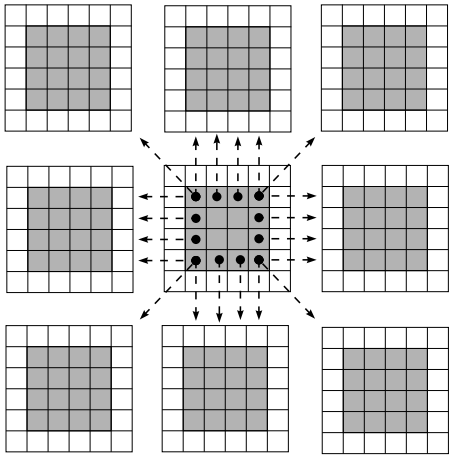

Fig. 2. Decomposition of the sample into blocks and cells. Each CPU deals with one block of cells. The white buffer cells contain copies of cells on neighbor CPUs, so that forces can be computed locally.

into the cells is an order N operation, and finding the atoms in the same and in neighboring cells is order N, too. In a parallel simulation, the sample is simply divided into blocks of cells, each CPU dealing with one block (Fig. 2). Each block is surrounded by a layer of buffer cells, which are filled before the force computation with copies of atoms on neighboring CPUs, so that the force can be computed completely locally.

This algorithm, which is manifestly of order N, is fairly standard for large scale MD simulations. Its implementation in IMD is somewhat special in one respect. The cells store the whole particle data in per-cell arrays and not indices into a big array of all atoms. This has the advantage that nearby atoms are stored closely together in memory as well, and stay close during the whole simulation. This is a considerable advantage on cache-based machines. The price to pay is an extra level of indirect addressing, which is a disadvantage on vector machines.

Although the link cell algorithm is of order N, there is still room for improvement. It can in fact be combined with Verlet neighbor lists. The advantage of doing this is explained below. The number of atoms in a given cell and its neighbors is roughly proportional to $(3r_c)^3$, where r_c is the interaction cutoff radius. In the link cell algorithm, these are the atoms which are potentially interacting with a given atom, and so at least the distance to these neighbors has to be computed. However, the number of atoms within the cutoff radius is only proportional to $\frac{4\pi}{3}r_c^3$, which is by a factor $\frac{81}{4\pi} \approx 6.45$ smaller. If Verlet lists are used, a large number of these distance computations can be avoided. The link cells are then used only to compute the neighbor lists (with an order N method), and the neighbor lists are used for the force computations. This leads to a runtime reduction of 30-40%, depending on the machine and the interac-

tion model (the simpler the interaction, the more important the avoided distance computations). The downside of using additional neighbor lists is a substantially increased memory footprint. On systems like the Cray T3E, neighbor lists are therefore not feasible, but on today's cluster systems they are a very worthwhile option.

There is one delicate point to be observed, however. If any atom is moved from one cell to another, or from one CPU to another, the neighbor lists are invalidated. As this could happen in every step *somewhere* in the system, these rearrangements of the atom distribution must be postponed until the neighbor tables have to be recomputed. Until then, atoms can leave their cell or CPU at most by a small amount $r_s/2$, which does not matter. The neighbor tables contain at each time all interacting neighbor particles.

3.2 Performance Measurements

We have measured the performance and scaling of IMD on four different cluster systems: a HP XC6000 cluster with 1.5 GHz Itanium processors and Quadrics interconnect, a 3.2 GHz Xeon EM64T cluster with Infiniband interconnect, a 2 GHz Opteron cluster with Myrinet 2000 interconnect, and an IBM Regatta cluster (1.7 GHz Power4+) with IBM High Performance Switch (Figs. 3-4). Shown is the CPU time per step and atom, which should ideally be a horizontal line. On each machine, systems of three sizes and with two different interactions are simulated. The systems have about 2k, 16k, and 128k atoms per CPU. One system is an FCC crystal interacting with Lennard-Jones pair interactions, the other a B2 NiAl crystal interacting with EAM [7] many-body potentials. The different system sizes probe the performance of the interconnect: the smaller the system per CPU, the more important the communication, especially the latency. As little as 2000 atoms per CPU is already very demanding on the interconnect.

The fastest machine is the Itanium system, with excellent scaling for all system sizes. For the smallest systems and very small CPU number, the performance increases still further, which is probably a cache effect. This performance was not easy to achieve, however. It required careful tuning and some rewriting of the innermost loops (which do not harm the performance on the other machines). Without these measures the code was $3-4$ times slower, which would not be acceptable. Unfortunately, while the tuning measures had the desired effect with the Intel compiler releases 7.1, 8.0, and 8.1 up to 8.1.021, they do not seem to work with the newest releases 8.1.026 and 8.1.028, with which the code is again slow. So, achieving good performance on the Itanium is a delicate matter.

The next best performance was obtained on the 64bit Xeon system. Its Infiniband interconnect also provides excellent scaling for all system sizes. One should note, however, that on this system we could only use up to 64 processes, because the other nodes had hyperthreading enabled. With hyperthreading it often happens that both processes of a node run on the same physical CPU, resulting in a large performance penalty. For a simulation with four processes per node, there was not enough memory, because the Infiniband MPI library allocates buffer space in each process for every other process.

The Opteron system also shows excellent performance, but only for the two larger system sizes. The small systems seem to suffer from the interconnect latency. The performance penalty saturates, however, at about 20%. We should also mention that these measurements have been made with binaries compiled with gcc. We expect that using the PathScale or Intel compilers would result in a 5-10% improvement.

Finally, the IBM regatta system is the slowest of the four, but also shows excellent scaling for all system sizes. For very small CPU numbers, the performance was a bit erratic, which may be due to interferences with other processes running on the same 32 CPU node.

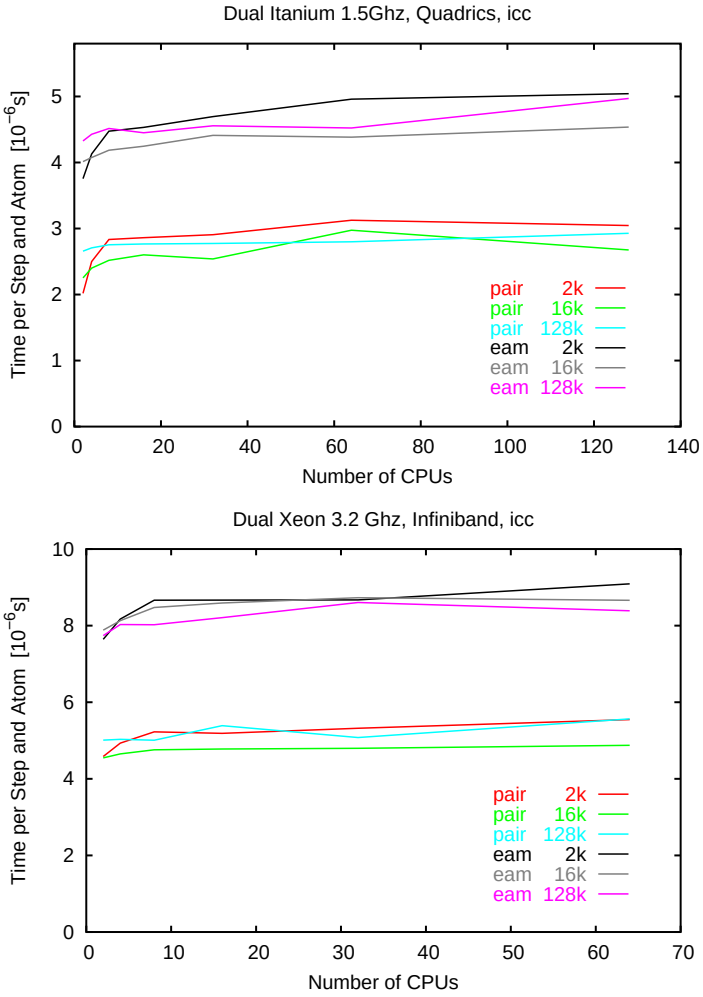

Fig. 3. Scaling of IMD on the Itanium (top) and Xeon (bottom) systems.

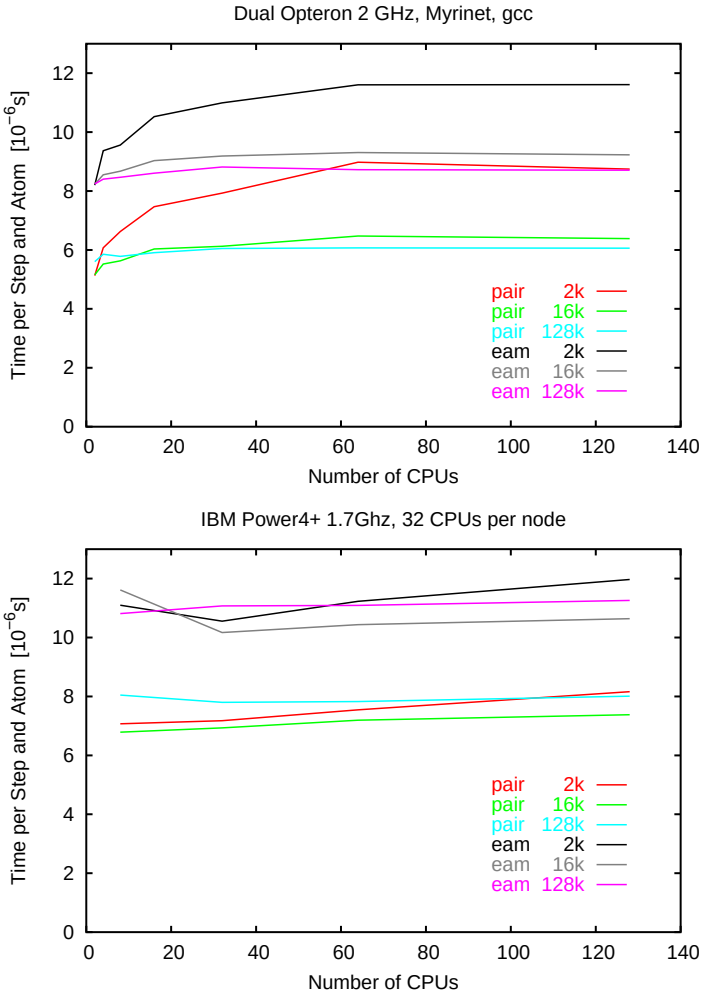

Fig. 4. Scaling of IMD on the Opteron (top) and IBM Regatta (bottom) systems.

4 Classical Molecular Dynamics on the NEC SX

The algorithm for the force computation sketched in Sect. 3.1 suffers from two problems, when executed on vector computers. The innermost loop over interacting neighbor particles is usually too short, and the storage of the particle data in per-cell arrays leads to an extra level of indirect addressing. The latter problem could be solved in IMD by using a different memory layout for the vector version, in which the particle data is stored in single big arrays and not in per-cell arrays. The cells then contain only indices into the big particle list. In order to keep as much code as possible in common between the vector and the scalar versions of IMD, all particle data is accessed via preprocessor macros.

The main difference between the two versions of the code is consequently the use of two different sets of access macros. The problem of the short loops has to be solved by a different loop structure. We have experimented with two different algorithms, the Layered Link Cell (LLC) algorithm [8], and the Grid Search algorithm [9].

4.1 The LLC Algorithm

The basic idea of the LLC algorithm [8] is to divide the list of all interacting atom pairs (implicitly contained in the Verlet neighbor list) into blocks of independent atom pairs. The pairs in a block are independent in the sense, that no particle occurs twice at the first position of the pairs in the block, nor twice at the second position. After all the forces between the atom pairs in a block have been computed, they can be added in a first loop to the particles at the first position, and in a second loop to the particles at the second position. Both loops are obviously vectorizable.

The blocks of independent atom pairs are constructed as follows. Let m be the maximal number of atoms in a cell. The set of particles at the first position of the pairs in the block is simply the set of all particles. The particle at position i in cell q is then paired with particle $i + k \bmod m$ in cell q', where q' is a cell at a fixed position relative to q (e.g., the cell just to the right of q), and k is a constant between 0 and m (0 is excluded, if $q = q'$). For each value of the neighbor cell separation and constant k, an independent block of atom pairs is obtained.

Among the atom pairs in the lists constructed above, there are of course many which are too far apart to be interacting. The lists are therefore reduced to those pairs, whose atoms have a distance not greater than $r_c + r_s$. These reduced pair lists replace the Verlet neighbor lists, and remain valid as long as no particle has traveled a distance larger than $r_s/2$, so that they need not be recomputed at every step.

The algorithm just described has been implemented in IMD, but its performance on the NEC SX is still modest (see Sect. 4.3). One limitation of the LLC algorithm is certainly that it requires the cells to have approximately the same number of atoms. Otherwise, the performance will degrade substantially. This condition was satisfied, however, by our crystalline test systems. In order to understand the reason for the modest performance, we have reimplemented the algorithm afresh, in a simple environment instead of a production code, both in Fortran 90 and in C. It turned out that the C version performs similarly to IMD, whereas the Fortran version is about twice as fast on the NEC SX (Sect. 4.3). The Fortran compiler apparently optimizes better than the C compiler.

4.2 The Grid Search Algorithm

As explained in Sect. 3.1, most of the particles in neighboring cells are too far away from a given one in the cell at the center to be interacting. This originates from the fact that a cube poorly approximates a sphere, especially if the cube

has edge length 1.5 times the diameter of the sphere, as it is dictated by the link cell algorithm. The resulting, far too many distance computations can be avoided to some extent using Verlet neighbor lists, but only an improved version of the LLC algorithm (the Grid Search algorithm) presents a true solution to this problem.

If one would use smaller cells, the sphere of interacting particles could be approximated much better. However, this would result in a larger number of singly occupied or empty cells, making it very inefficient to find interacting particles. A further problem is, that with each cell a certain bookkeeping overhead is involved. As the number of cells would be much larger, this cost is not negligible, and should be avoided.

The Grid Search algorithm tries to combine the advantages of a coarse and a fine cell grid, and avoids the respective disadvantages.

The initial grid is relatively coarse, having $2-3$ times more cells than particles. To use a simplified data structure, we demand at most one particle per cell, a precondition which cannot be guaranteed in reality. In case of multiply occupied cells, particles are reassigned to neighboring cells using *neighbor cell assignment* (NCA). This keeps the number of empty cells to a minimum. During NCA each particle gets a virtual position in addition to its true position. To put it forward in a simple way, the virtual positions of particles in multiply occupied cells are iteratively modified by shifting these particles away from the center of the cell on the ray connecting the center of the cell and the particle's true position. As soon as the precondition is satisfied the virtual positions are discarded. Only the now compliant assignment of particle to cell, stored in a one-dimensional array, and the largest virtual displacement d_{max}, denoting the maximal distance between the virtual and true position of all particles, are kept.

The so-called *sub-cell grouping* (SCG) exploits the exact positions of the particles relative to their cells by introducing a finer hierarchical grid. This reduces the number of unnecessarily examined particle pairs and distance calculations. To simplify the explanation, we assume in first instance that NCA is not used.

The basic idea of Grid Search is to palter with chance to get a "successful" distance computation. We consider a pair of two cells, the cell at the center C and a neighbor cell N, with one particle located in each cell. In the convenient case, the neighbor cell is sufficiently close to the cell at the center (Fig. 5), so that there is a good chance that the two particles contained in the cells are interacting.

In the complicated case, if the neighbor cell is so far apart of the cell at the center (Fig. 6) that there is only a slight chance that the particle pair gets inserted into the Verlet list, SCG comes into play. The cell at the center is divided into a number of sub-cells, depending on integer arithmetic. Extra sub-cells are added for particles that have been moved by NCA to neighboring cells for each quadrant/octant (Fig. 7). A fixed sub-cell/neighbor cell relation is denoted as *group*.

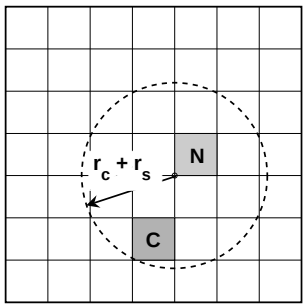

Fig. 5. Cell at the center C is sufficiently close to neighbor cell N.

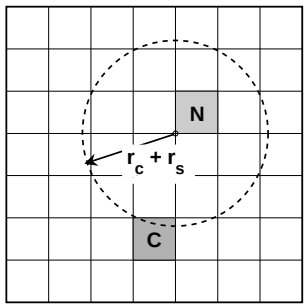

Fig. 6. Cell at the center C is not close enough to neighbor cell N.

By comparing the minimal distance between each sub-cell and the neighbor cell to $r_c + r_s$, a number of groups can be excluded in advance. As shown in Fig. 8, only $\frac{1}{4}$ of the initial cell at the center needs to be searched.

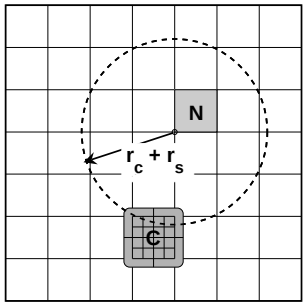

Fig. 7. Cell at the center is divided into sub-cells.

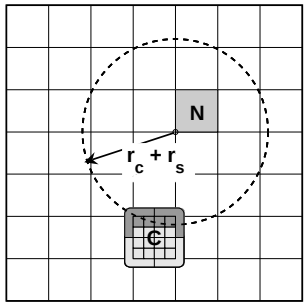

Fig. 8. Some groups can be excluded from search.

The use of NCA complicates SCG, because it changes the condition for excluding certain groups for a given neighbor cell relation in advance: the minimal distance between a sub-group and a neighbor cell does no longer have to be smaller or equal than $r_c + r_s$, but smaller or equal than $r_v = r_c + r_s + d_{max}$. The virtual displacement occurs only once in r_v, since one particle is known to be located in the sub-cell, and the other one can be displaced by as much as d_{max}. Thus, the set of groups that need to be considered changes whenever the particles are redistributed into the cells, i.e., whenever the Verlet list is updated.

In order to reduce the amount of calculations and to save memory, a data structure is established, stating whether a given group can contain interacting particles for a certain virtual displacement. For 32(64)-bit integer arithmetics, the cell at the center is divided into $4 \times 4 \times 3$ ($3 \times 3 \times 2$) sub-cells and eight extra-cells (one for each octant) resulting in 56(26) groups. So in a two-dimensional

integer array, the first dimension being the neighbor cell relation, the second indicating a certain pre-calculated value of d_{max}, the iGr-th bit (iGr is the group number) is set to 1 if the minimal distance between the sub-cell and the neighboring cell is not greater than r_v.

The traditional LLC data structures, a one-dimensional array with the number of particles in each cell and a two-dimensional array listing the particles in each cell, are used in Grid Search on the sub-cell level: a one-dimensional array storing the number of particles in each group and a two-dimensional array listing the particles in each group. Together with the array of cell inhabitants produced by the NCA, this represents a double data structure on cell and sub-cell level, respectively: for each cell we know the particle located in it, and for each sub-cell we know the total number and which particles are located in it.

As in the LLC algorithm, independent blocks of the Verlet list consist of all particle pairs having a constant neighbor cell relation. The following code examples describe the setup of the Verlet list. For neighbor cells sufficiently close to the cell at the center, the initial grid is used:

```
do for all particles j1
  if the neighbor cell of the cell with particle j1 contains a
    particle j2 then
      save particles to temporary lists
  endif
end do
```

If the distance of the neighbor cell to the cell at the center is close to r_v, then SCG is used:

```
do for all sub-cells
  if particles in this sub-cell and the given neighbor cell
  can interact then
    do for all particles in this sub-cell
      if the neighbor cell of the sub-cell with particle j1
        contains a particle j2 then
          save particles to temporary lists
      endif
    end do
  end if
end do
```

The temporary lists are then, as in the LLC algorithm, reduced to those pairs whose atoms have a distance not greater than $r_c + r_s$.

4.3 Performance Measurements

To compare the performance of the LLC and the Grid Search (GS) algorithms, an FCC crystal with 16384 or 131072 atoms with Lennard-Jones interactions is

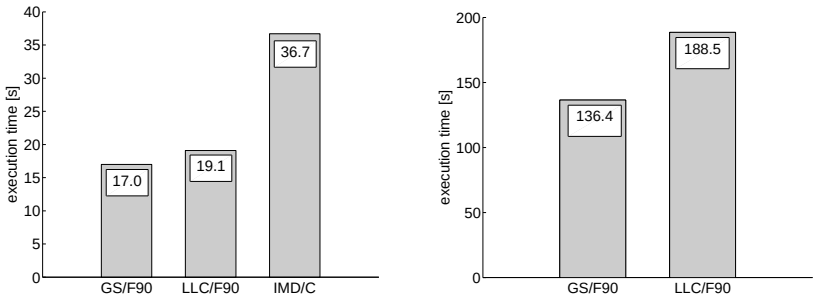

Fig. 9. Execution times of the different algorithms on the NEC SX8, for FCC crystals with 16k atoms (left) and 131k atoms (right).

simulated over 1000 time steps using a velocity Verlet integrator. As reference, the same system has also been simulated with the LLC algorithm as implemented in IMD. The execution times are given in Fig. 9. Not shown is the reimplementation of the LLC algorithm in C, which shows a similar performance as IMD.

For the Grid Search algorithm, the time per step and atom is about $1.0\mu s$, which is more than twice as fast as IMD on the Itanium system. However, such a comparison is slightly unfair. The Itanium machine simulated a system with two atom types and a tabulated Lennard-Jones potential, which could be replaced by any other potential without performance penalty. The vector version, in contrast, uses computed Lennard-Jones potentials and only one atom type (hard-coded), which is less flexible but faster. Moreover, there was no parallelization overhead. When simulating the same systems as on the Itanium with IMD on the NEC SX8, the best performance obtained with the 128k atom sample resulted in $2.5\mu s$ per step and atom. This is roughly on par with the Itanium machine. An equivalent implementation of Grid Search in Fortran would certainly be faster, but probably by a factor of less than two.

Next, we compare the performance on the NEC SX6+ and the new NEC SX8. The speedup of an SX6+ executable running on SX8 should theoretically be 1.78, since the SX6+ CPU has a peak performance of 9 GFlop/s, whereas the SX8 CPU has 16 GFlop/s. Recompiling on SX8 may lead to even faster execution times, benefiting e.g. from the hardware square root or an improved data access with stride 2.

As Fig. 10 shows, our implementation of the Grid Search algorithm takes advantage of the new architectural features of the SX8. The speedup of 2.14 is noticeably larger than the expected 1.78. On the other hand, IMD stays in the expected range, with a speedup of 1.83. The annotation 'SX6 exec.' refers to times obtained with SX6 executables on the SX8.

Acknowledgments The authors would like to thank Stefan Haberhauer for carrying out the VASP performance measurements.

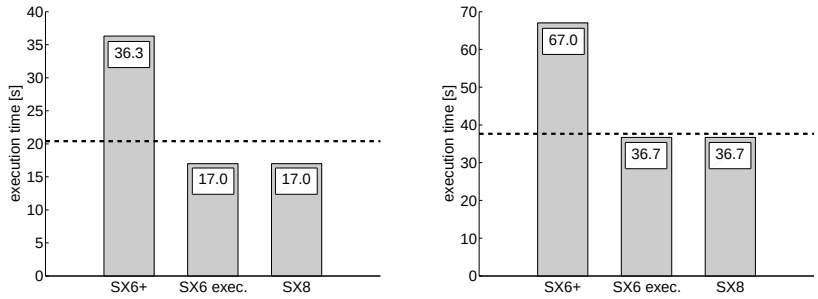

Fig. 10. Execution times on SX6+ and SX8 for an FCC crystal with 16k atoms using Grid Search (left) and IMD (right).

References

1. F. Ercolessi, J. B. Adams, *Interatomic Potentials from First-Principles Calculations: the Force-Matching Method*, Europhys. Lett. **26** (1994) 583–588.
2. P. Brommer, F. Gähler, *Effective potentials for quasicrystals from ab-initio data*, Phil. Mag. **86** (2006) 753–758.
3. G. Kresse, J. Hafner, *Ab-initio molecular dynamics for liquid metals*, Phys. Rev. B **47** (1993) 558–561.
4. G. Kresse, J. Furthmüller, *Efficient iterative schemes for ab-initio total-energy calculations using a plane wave basis set*, Phys. Rev. B **54** (1996) 11169–11186.
5. G. Kresse, J. Furtmüller, *VASP – The Vienna Ab-initio Simulation Package*, `http://cms.mpi.univie.ac.at/vasp/`
6. J. Stadler, R. Mikulla, and H.-R. Trebin, *IMD: A Software Package for Molecular Dynamics Studies on Parallel Computers*, Int. J. Mod. Phys. C **8** (1997) 1131–1140 `http://www.itap.physk.uni-stuttgart.de/~imd`
7. M. S. Daw, M. I. Baskes, *Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals*, Phys. Rev. B **29** (1984) 6443–6453.
8. G. S. Grest, B. Dünweg, K. Kremer, *Vectorized Link Cell Fortran Code for Molecular Dynamics Simulations for a Large Number of Particles*, Comp. Phys. Comm. **55** (1989) 269–285.
9. R. Everaers, K. Kremer, *A fast grid search algorithm for molecular dynamics simulations with short-range interactions*, Comp. Phys. Comm. **81** (1994) 19–55.