Dynamic load-balancing of hierarchical tree algorithms on a cluster of multiprocessor PCs and on the Cray T3E.

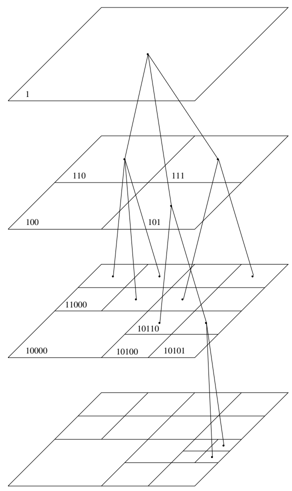
Attila Caglar, Michael Griebel, Marc A. Schweitzer and Gerhard Zumbusch†

Abstract

The solution of many problems in science and engineering is based on computational kernels for the numerical treatment of partial differential equations (PDEs) or N-body problems. Traditional solution methods however reduce these to linear algebra or brute force algorithms on structured data sets. Larger and larger simulations require smarter algorithms to be tractable. Hierarchical tree algorithms represent such a class, both for PDEs and for N-body problems. However, their efficient parallelization is not straightforward. Some difficulties can be removed, if one can provide a fast dynamic load-balancing scheme to cope with the tree variations of the unstructured data sets. In this paper we propose a very cheap yet very efficient load-balancing scheme for tree algorithms based on space-filling curves. Furthermore we present the Parnass2 cluster, on which such parallel codes perform extremely well. The cluster consists of SMP PCs and a Myrinet network at Gigabit/s speed configured with full bisection bandwidth. It turns out that it does not only has the obvious price/performance advantage, but also an absolute performance, which is comparable to the latest commercial Cray T3E.

1 Algorithmic Problem

Hierarchical numerical methods belong to the most effective sequential methods for solving problems in a wide range of applications. For example in the treatment of *partial differential equations* they allow for a multilevel solver like multigrid or the BPX preconditioner, see [5, 10] as well as adaptive refinement techniques, see [1, 3]. Here, the aim is to obtain an approximation to the continuous solution within a prescribed error tolerance with an amount of work which is proportional to N, i.e. the number of unknown of the finest adapted grid. The actual approximation error should be smaller than some given ε. To this end, in most practical three-dimensional applications (potential problems, temperature and diffusion processes, flow problems, elasticity problems etc.), the solutions of interest are in general not smooth, but exhibit steep variations and singularities. Here, uniform grids should not be employed but the use of adaptively refined grids is a must for reasons of efficiency. Thus the underlying tree algorithm leads to data which are not equi-distributed over the domain (see Figure 1 (left) for a simple example).



A partial tree of an N-body algorithm.

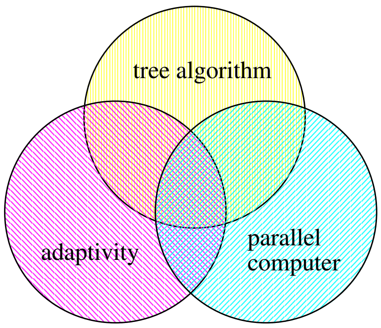
Furthermore, hierarchical tree codes are found in the treatment of integral equations,

†Sonderforschungsbereich 256 *Nonlinear Partial Differential Equations*, Institut für Angewandte Mathematik, Universität Bonn, Wegelerstr. 6, D-53115 Bonn, {caglar, griebel, schweitz, zumbusch}@iam.uni-bonn.de, http://wissrech.iam.uni-bonn.de

in *particle simulations* and *N-body problems*, where a fast, approximative evaluation of forces between particles in a force field with long range potential is to be computed. Here, applications range from astrophysics (simulation of structure formation of the universe, galaxy formation), over fluid flow problems (tree-SPH) and physics (particle methods) to material science, chemistry and biology (molecular dynamics), see [2, 9]. The two major tree algorithms are the Barnes-Hut method and the fast multipole technique. They are of orders $\mathcal{O}(N \log N)$ and $\mathcal{O}(N)$ complexity, respectively. Here, we encounter clustered data sets due to the movement of the particles in time. The distribution of the particles is adaptively resolved by the respective tree algorithm. Examples are given in Figure 1 (right and bottom).

Within both classes of hierarchical algorithms, i.e. the adaptive multilevel solution of PDEs and the fast algorithms for the N-body problem, we have to deal with tree-like data structures. They are in general not well balanced due to the non-uniform distribution of the data. Furthermore, the trees also change dynamically due to either adaptive grid refinement or the movement of the particles.

In the sequential case this imbalance of the trees does not affect efficiency. However, when it comes to parallelization, this imbalance poses a major problem. A straightforward parallelization based on a simple, static domain decomposition approach would lead to severe load imbalance between processors which can hamper the parallel efficiency substantially.



To overcome this difficulty we have to deal with a load balance problem. Due to the dynamically changing data distribution, load balancing has to be performed during run time of the respective algorithms and cannot be done in a pre-processing step. Note that the problem is in general N-P hard. Therefore one has to settle for cheaper suboptimal heuristics. In [21] we find a survey on mesh partitioning and data balancing techniques. Note furthermore, that our abovementioned tree algorithms possess an optimal complexity of $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$, respectively. Thus the applied heuristics should also have at most this complexity. Note finally that, after parallelization, our hierarchical tree algorithms would have a parallel complexity of $\mathcal{O}(N/P + \log P)$ or $\mathcal{O}(N/P \log N + \log P)$, respectively. Consequently a dynamical load balancing heuristics must also run *in parallel* and must have a comparable parallel complexity. Otherwise, the load balancing part of the overall algorithm would dominate the solution part which is not acceptable from the efficiency point of view. This problem of a cheap parallel load balancing heuristics rules out most of the existing approaches which work nicely for many other numerical algorithms with higher sequential complexities. The load-balancing becomes more challenging in the presence of more advanced, optimal complexity algorithms like the ones we will consider in this paper.

2 Parallel Approach

The search for optimal algorithms to solve a problem up to a given approximation error lead us to tree-type algorithms, e.g. for partial differential equations or integral equations. However, unbalanced trees due to adaptivity and clustering in space causes serious trouble for the parallelization. Especially it requires extremly cheap dynamic load-balancing methods. In order to accomplish these goals, we employ a very cheap heuristic, tightly integrated into the code and the surrounding tree algorithms. Of course, such a heuristic does perform only sub-optimal and in general leads to a larger communication overhead than the optimum partition would need. However, no heuristic can find an optimal solution for any large data set of interest, neither some polynomial complexity methods, nor the constant time method we propose here. In the past it has been a major undertaking to parallelize certain kinds of tree codes, see [17, 28, 7, 18, 25, 8, 3]. If the load-balancing problem was tackled at all, either based on different heuristics or on graph

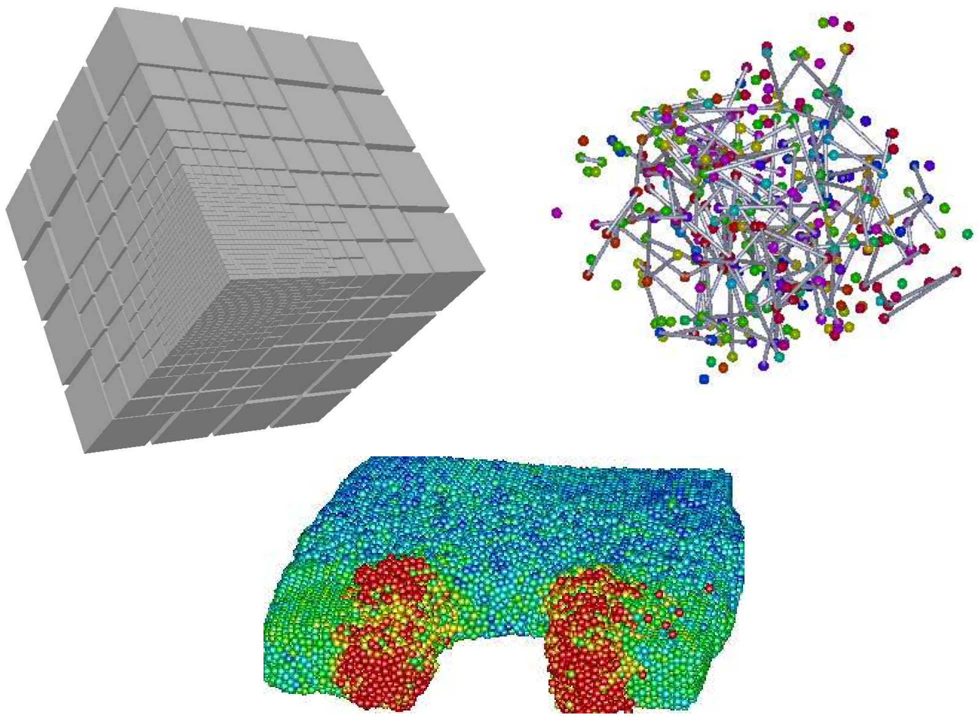
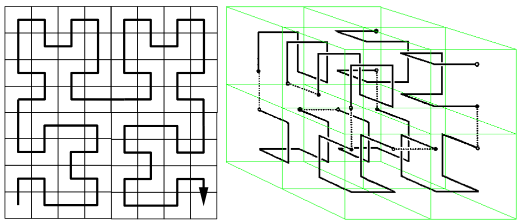


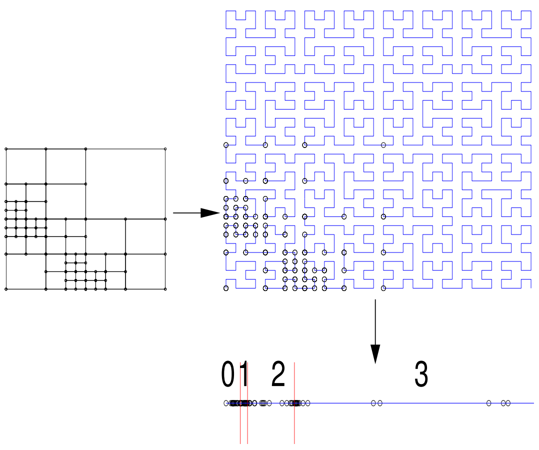
Figure 1: An adaptively refined grid for the solution of a PDE (upper left) and a clustered set of particles in a molecular dynamics simulation (upper right) and in crack propagation simulations in 3D (bottom).

based partitioning methods, the time spent in load-balancing and the quality were always the limiting factor for truly un-balanced trees and larger number of processors. Hence we propose a new balancing method.

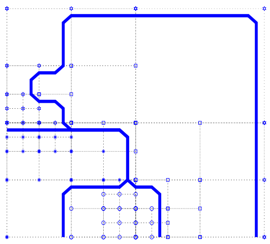


The Hilbert space-filling curve in 2D and 3D.

The load-balancing method to be described is based on space-filling curves, see [15, 22, 31]. The curve defines a mapping of the three-dimensional geometric space to the one-dimensional real axis. The one-dimensional problem is cut into P intervals of the same amount of nodes/particles/work. The inverse space-filling curve mapping results in a data partition that can be interpreted as a sophisticated domain decomposition. The pictures indicate a simple example. The final algorithm shifts and rotates the space-filling curve in order to reduce the partition boundaries.



Mapping a 2D adapted grid to a space-filling curve (left) and mapping points on a space filling curve to a parallel processor (right).



A decomposition of the domain induced by the space filling curve mapping.

While the one-dimensional partition is optimal, the three-dimensional is not: Compared to the optimum, roughly 30% overhead in communication is added. However, the load-balancing scheme itself is very cheap and runs efficiently also in parallel. It can be implemented by a simplistic parallel sort algorithm operating on a pre-sorted set which is given by the previous data partition. The tree algorithms can be tightly integrated into the load-balancing scheme: Due to the deterministic nature of the partition, it is possible to compute the positions where nodes such as neighbors or hierarchical parent or child nodes are located. Hence one gets rid of the standard administration and bookkeeping which gives both savings in memory, lines of code and computational efficiency. Space-filling curves have been proposed for load-balancing by [26, 23, 20, 19] and further details can be found in [12, 13, 14, 29, 6].

While the data is partitioned and mapped onto the parallel processors, the data-tree of the algorithm spans all processors. A tree traversal e.g. has to be implemented in a clever way in order to match the results of the sub-trees located on the processors. However, the whole traversal procedure can be re-interpreted as a re-balancing of the unbalanced tree, like it is done in the re-balancing algorithms with balanced trees such as the AVL tree, for example.

Due to the unbalanced tree, the precise communication patterns are not predictable a priori. Hence a parallel computer network is required, where geometric distances in the network topology do not matter. Furthermore, network bandwidth must be sufficiently high in that case. As we will see later, the Myrinet interconnection tree of the Parnass2 cluster fulfills this requirement, along with very low communication latencies needed in some other parts of the algorithm.

3 Hardware and Configuration of Parnass2 Cluster

Parnass2 is a dedicated cluster of personal computers (PCs) used for research in scientific computing at the Department of Applied Mathematics at the University of Bonn. This resource is used as a supercomputer to carry out computational tasks which are too large for conventional workstations or servers. The cluster utilizes off-the-shelf components (Intel Pentium-II, Myrinet, Linux), hence it can achieve 8.5 Gflops performance per $100K. But more importantly, on real applications it competes surprisingly well with systems costing 10–100 times more. Because the system is inexpensive (about the cost of a high-end workstation), it can be used as a *personal supercomputer*, rather than a resource that must be shared by many people, see [11].

In the current installation Parnass2 consists of 96 CPUs (48 Dual-Pentium-II 400 MHz) with 256 MB RAM, see [30]. The operating system of Parnass2 is Linux, currently with a 2.2.7 kernel, together with compilers for C, C++, Fortran77 and Fortran95. In contrast to standard numerical algorithms, tree based algorithms rely to a certain extend on integer operations and irregular memory access patterns. Hence processors of the Intel Pentium line with a theoretical ratio of 2.5 Operations per Flop together with small cache lines achieve much higher efficiencies than e.g. Compaq Alpha processors or traditional vector processors. Furthermore the space-filling curve load-balancing requires some additional integer operations to be executed frequently.

The 48 SMP computing nodes of Parnass2 are connected in a two-level fat-tree topology (Figure 4) with full bisection bandwidth of 61 Gbit/s. Using an MPI implementation from RWCP tailored for Myrinet and SMP nodes, a communication bandwidth of 850 Mbit/s and a latency of $11\mu s$ can be measured in application codes. There are even competing uniprocessor MPI implementations with latencies as low as $3\mu s$. In addition to this high-speed computing network, Parnass2 employs Fast-Ethernet for file I/O to/from an external file server.

Overall this hard- and software configuration enabled us to measure 18.3 GFlop/s

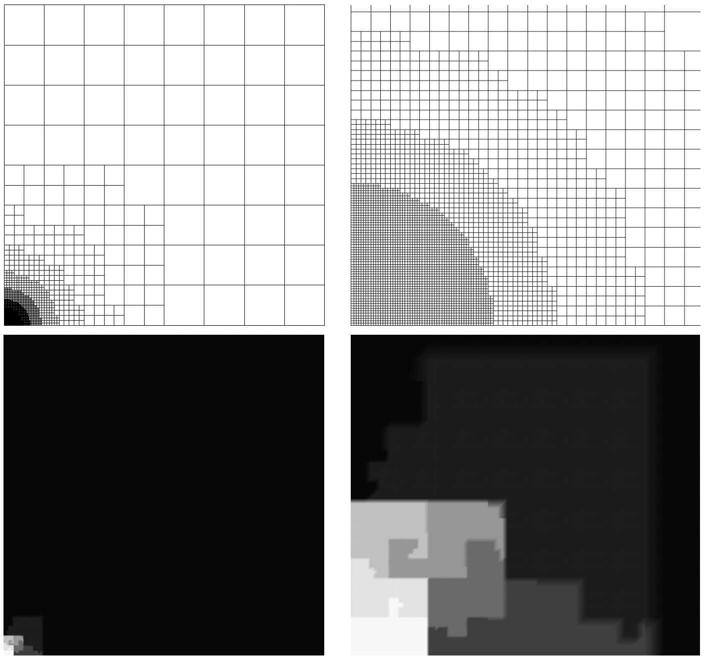


Figure 2: 2D refined grid, mapped to 8 processors, full domain left, zoomed image on the right, grid and color coded partition.

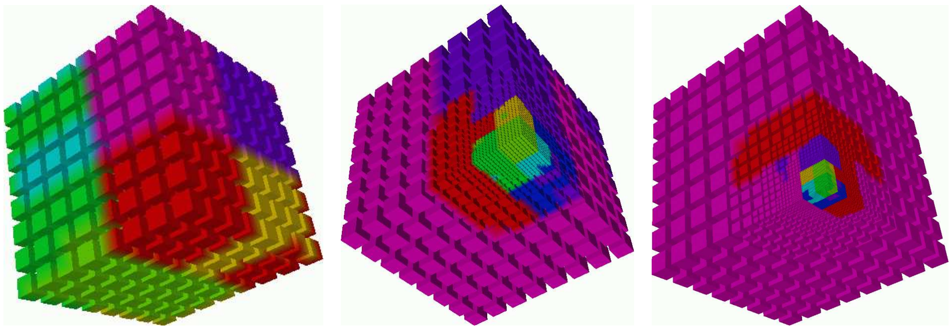


Figure 3: A sequence of adaptively refined grids mapped to 8 processors, partition is color coded.

in a Linpack benchmark on Parnass2, which is roughly 45% of the peak-performance. In the final installation of 128 CPUs connected with a Myrinet in a three-level fat-tree topology (Figure 5) with full bisection bandwidth. Parnass2 will have a peak performance of 51.2 Gflops/s and 32 GByte main memory. The overall prize of this final installation is $280K, giving an anticipated prize-performance ratio of $5.500 per peak Gflops/s and $12K per sustained Gflops/s.

However in the parallel codes presented here, small communication latencies play a dominant role, delivered either by shared memory or by the extreme low latency Myrinet network. The tree algorithms furthermore require high network bandwidths in some parts of the execution, which can be delivered by the full-bisection bandwidth fat-tree, in contrast to torus topologies used e.g. in the Cray T3 and Intel Paragon series (ASCI Red) which are designed for structured applications such as the Linpack benchmark. Further details may be found in [24].

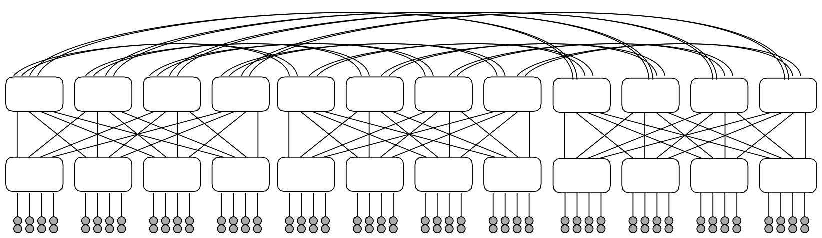


Figure 4: Two level fat-tree topology of 96 CPUs (48 nodes).

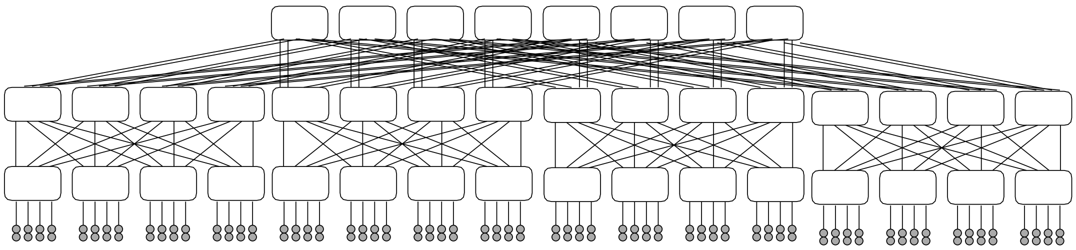


Figure 5: Three level fat-tree topology of 128 CPUs (64 nodes).

4 Experimental Results

Adaptive Finite Elements and Multilevel Solver for Linear Elasticity Problems.
The Lamé equation in the displacement formulation

$$\mu\vec{\Delta}\vec{u} + (\lambda+\mu)\nabla(\nabla\cdot\vec{u}) \quad = \quad \vec{f} \qquad (1)$$

with Lamé's constants λ, $\mu > 0$, was considered for our performance test with the adaptive finite element multilevel solver. A nested iteration is used for a solution up to discretization error. With an additive multilevel-V cycle preconditioned Krylov iteration on each grid this means a constant number of iterations. Together with a geometrically increasing number of grid points we end up with a total of $\mathcal{O}(N)$ operations. The adaptive grid refinement is controlled by a residual based error indicator, leading to a parallel adaptive grid refinement of 1-irregular grid built of hexahedral elements, where a finite difference type discretization is employed. The parallel version of the code uses a space-filling curve data partition after each adaptive grid refinement step.

As a test problem, we consider a homogeneous cube under internal forces parallel to one coordinate axis, fixed at three faces. The remaining edges or faces are free (homogeneous Neumann conditions). The solution develops singularities. Here adaptive refinement is used to resolve the singularities next to the edges or faces (left, bottom and front), which separate homogeneous Dirichlet (fixed faces) and homogeneous Neumann conditions (free faces), see Figure 1 (left).

Tables 1 and 2 give wall clock times measured on Parnass2 (400 MHz Pentium) and on a Cray T3E-1200 (600 MHz Alpha) for comparison. In general, the numbers show that the presented algorithm scales very well. We obtain a scaling of about a factor 3–5 from one level to the next finer level, i.e. the times are proportional to the number of unknowns for a fixed number of processors. This is due to the adaptive grid refinement heuristic. For a fixed number of processors, we observe a scaling of a factor from one level to the next finer level which corresponds to increase in the amount of unknowns on that level. Furthermore, for a fixed level the measured times scale roughly with $1/P$ of the number of processors. However, the 64 processors and more perform efficiently only for sufficiently large problems, i.e. for problems with more than some ten thousand degrees of freedom. For larger problems we even obtain some super-linear speed-ups, probably due to caching effects. If we fix the amount of work, that is the number of nodes per processor, we obtain the scale-up. Comparing a time at one level l and a number of processors P with the time

time		processors						
nodes	dof	1	2	4	8	16	32	64
125	375	0.10	0.12	0.11	1.50	2.91		
450	1350	1.44	0.99	0.80	1.35	0.50	0.39	1.05
1155	3465	**4.14**	2.48	1.71	1.32	1.00	0.70	2.74
4412	13236	19.0	10.3	**6.09**	5.23	3.07	1.89	1.21
18890	56670	98.6	50.3	28.1	20.6	**11.6**	6.35	3.70
93021	279063	582	294	157	102	54.8	28.2	**15.1**
506620	1519860				556	306	155	78.1
3178218	9534654							494

Table 1: Parallel execution times (in seconds) for the solution of Lamé's equation in 3D, Parnass2.

time		processors								
nodes	dof	1	4	16	64	128	256	512	768	1024
35937	107811	162	**34.1**	9.11	2.23	1.23	0.75	0.55	0.56	0.52
109873	329619	435	108	**29.6**	7.20	3.57	1.87	1.13	0.91	0.80
410546	1231638			114	**28.6**	14.2	7.02	3.51	2.48	1.94
1857030	5571090				133	67.1	**33.3**	16.5	11.0	

Table 2: Parallel execution times (in seconds) for the solution of Lamé's equation in 3D, Cray T3E-1200.

of one level finer $l+1$ and roughly $4 \cdot P$ processors, we obtain nearly a perfect scaling of the method.

The uniform initial coarse grids on Parnass2 and on the Cray had to be chosen differently to allow scaling up to 1024 processors. Therefore the number of unknowns per level and the corresponding amount of work for the multilevel V-cycle are not the same on the two machines, which results in slightly longer execution times on Parnass2. In fact, other experiments show slight performance advantages for Parnass2.

Further experiments reveal that the effort and computing time spent on load-balancing is below 1% of the time needed for the solution of the linear equation systems and is hence negligible. We actually can afford to perform a load-balancing every time, when an adaptive grid refinement takes place and the grid changes. In other words, the computation is load-balanced at every step of the overall computation.

Tree-Multipole-Method for Long-Range Molecular Dynamics Simulations

In the molecular dynamics approach, atoms and molecules are considered as classical particles in a Hamiltonian system, which move according to Newton's equations of motion. Here usually a time discretization scheme due to Verlet is employed. The interactions of atoms and molecules are approximated by analytic potential functions. We distinguish between bonded and non-bonded interactions. The bonded interactions are local and are restricted to the near-neighbors of each particle (atom). The non-bonded interactions are due to Van-der-Waals and electrostatic forces. From the numerical point of view the non-bonded interactions are most interesting. These can be modeled by the Lennard-Jones or the Coulomb potential, respectively. Then, the total potential V satisfies

$$V(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] + \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}}. \quad (2)$$

The Lennard-Jones potential is of short range type. By applying a cut-off technique, the complexity of its evaluation can be reduced to $\mathcal{O}(N)$, where N is the number of atoms. Here implementations are mainly based on the linked-list method due to Hockney and Eastwood [16]. The parallelization is usually done by a classical domain-decomposition [4].

The Coulomb potential describes *long range* interactions. Therefore, the cut-off technique can no longer be applied and a complexity of $\mathcal{O}(N^2)$ for a naive approach would result. Thus the major difficulty in MD-

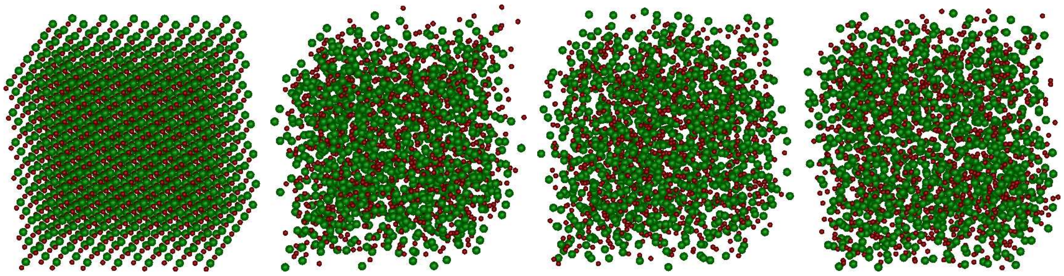


Figure 6: Melting process of NaCl crystal. Melting temperature 1800 K

simulation methods is the treatment of the long range force evaluation in each time step of the Verlet scheme. To cope with this problem, various multiscale type methods have been developed, i.e. tree codes, multipole approaches or multigrid techniques, which reduce the complexity to $\mathcal{O}(N \log N)$ (tree code of Barnes-Hut [2]) or even $\mathcal{O}(N)$ (fast multipole method of Greengard and Rokhlin [9]). A further reduction of execution time is possible by parallelization. Here, however – especially for the above mentioned adaptive tree-type methods – the implementation is quite difficult and cumbersome. To our knowledge, there exists no effective parallel version of the Barnes-Hut or multipole method for molecular dynamics applications yet.

We have implemented a mixture of the Barnes-Hut and the fast multipole technique within a tree type algorithm, we call it Tree-Multipole. We use it for the force evaluation in each time step. Furthermore we incorporated features necessary to deal with periodic boundary conditions and we incorporated the intra-molecular interactions into the tree-multipole structure to treat molecules and polymers as well. Here, we use a hash-technique to deal with adaptivity. To some extent, our approach joins the advantages of both methods. It gives the higher accuracy of the multipole approach within the simpler framework of the Barnes-Hut tree implementation, but avoids the quite expensive multipole-to-local transformations of the multipole series expansion and simplifies a parallel implementation. The complexity of our algorithm is $\mathcal{O}(N \log N)$.

The parallelization of the resulting tree multipole method is achieved by means of space filling curves as explained in the previous section 2. Note that for the Barnes-Hut algorithm for astrophysical applications, Salmon and Warren followed in [27] a similar approach. They employed a Morton curve which is however inferior to our Hilbert curve because it shows discontinuities and bad load-balancing in many cases. They also were aware of this fact but, at least to our knowledge, they never implemented this better variant. Probably our implementation is the first effective parallel realization of such an adaptive hierarchical multipole approach for molecular dynamics simulations which is based on the Hilbert curve.

We run our parallel tree multipole algorithm on Parnass2 and also for comparison reasons on the Cray T3E. As a test case we consider the melting of sodium-chloride (NaCl), see Figure 6, where each Na and Cl ion resembles a particle. Here, we encounter both, short range potentials and long range potentials. We use a total potential similar to (2), where in addition for the short range part the Lorentz-Berthelot mixing rules are employed. Note that the long range potential comes in due to the electrostatic nature of the molecule.

We conducted experiments with varying number N of particles and varying number P of processors. The results on Parnass2 are given in Table 3. Here we give the measured run times for one time step of the Verlet scheme with our tree multipole algorithm used for the force evaluation. The results on the Cray T3E are given in Tables 4 and 5. Here, we were able to simulate up to 64 million atoms (sic !) in a molecular dynamics simulation with intra- and intermolecular interaction potentials on a Cray T3E with 512 processors. From these numbers we see the anticipated logarithmic behavior of the scale-up on Parnass2 as well as on the Cray. But again we have a slightly better scaling of the code on the Cray than we have on our clus-

time	processors P										
particles N	1	2	4	8	16	24	32	48	64	80	96
2028	0.24	0.16	0.12								
10164	4.00	2.23	1.36	0.74	0.46		0.31	0.26	0.24		0.25
101614	66	33	17.4	8.9	4.9		2.7	2.0	1.7		1.2
1.000.000				**180**	74	49	32	23	17	14.5	12
2.000.000					**170**	105	74	51	35	29	
3.000.000						**245**	168	102	72	54	
4.000.000							**269**	155	106	79	
6.000.000								**267**	187	134	
8.000.000									**298**	213	
10.000.000										**279**	

Table 3: Parnass2: Parallel execution times $t(N, P)$ (in seconds) for one time-step of the simulation of *Melting of NaCl* using the tree multipole method within the Verlet–scheme.

time	processors P							
particles N	1	2	4	8	16	32	48	64
2028	0.24	0.21	0.17					
10164	4.00	2.23	1.36	0.74	0.46	0.31	0.26	0.24
101614	126	58	26	13	6	3.2	2.3	1.8
1000000				320	128	63	36	25

Table 4: Cray T3E-900: Parallel execution times (in seconds) for one time-step of the simulation of *Melting of NaCl* using the tree multipole method within the Verlet–scheme.

time	processors P		
particles N	128	256	512
1 mio.	16.12	8.18	5.30
2 mio.	35.40	16.94	
4 mio.	**86.12**	40.95	22.40
8 mio.	207.80	**91.56**	48.07
16 mio.	**511.42**	214.04	**99.65**
32 mio.	1763.53	**644.61**	
64 mio.			**746.88**

Table 5: Cray T3E-1200: Parallel execution times (in seconds) for one time-step of the simulation of *Melting of NaCl* using the tree multipole method within the Verlet–scheme.

ter. This is due to the superior network bandwidth and latency of the Cray.

The single processor numbers indicate that the Intel Pentium II 400 MHz processors perform roughly by a factor of two better than the more floating point oriented Compaq Alpha processors at 450 MHz. As in the multigrid example, also the irregular memory access patterns run more efficiently on the Pentium with small cache lines than on the streams interface of the Cray. Obviously the multi-million $ propriety network hardware of the Cray has orders of magnitude faster peak bandwiths than the Myrinet network of Parnass2. However, we roughly obtain a similar scaling behaviour of the algorithm with an increasing number of processors. This is surely due to the tree type structure of the algorithm and the fat-tree topology of Parnass2's Myrinet.

5 Concluding Remarks

In this paper, we proposed tree type numerical algorithms as computational kernels of different kinds of simulation codes used in science and engineering. We considered two examples, an adaptive parallel multigrid method for the solution of partial differential equations and a parallel Barnes-Hut/multipole method for the fast evaluation of forces in a molecular dynamics Verlet integrator. In both cases tree algorithms were succesfully applied to reduce the computational complexity in a substantial way. However, within the parallelization of these algorithms we had to cope with the problem of load-balancing in the presence of adaptive grid refinement and particle clustering. Here, a space-filling curve partitioning heuristic was used, which led to a very efficient parallelization of these advanced methods. Furthermore, a cluster of SMP PCs (Parnass2) connected by a fat-tree Myrinet communication network was presented, which is used for these simulations at our department. Numerical experiments and a comparison with the latest and largest Cray T3E computer showed the competitiveness of the cluster design. It turned out that Parnass2 is not only running at a much better price/performance ratio, but also the absolute performance figures are competitive. This is in part due to irregular memory access patterns and a large amount of integer operations of the tree algorithms. However, the good parallel scalability also stems from the high performance and low latency Myrinet network and the full bisection bandwith topology used in the cluster Parnass2. Assuming a fictive budget of 1 Mio. DM, it would be possible to simulate 128 million particles in molecular dynamics with long range potentials or more than 30 million degrees of freedom in partial differential equations, respectively. This fact, along with the experiences from our experiments let us predict a bright future for cluster computing.

6 Acknowledgement

We would like to thank NIC KFA-Jülich and Cray Research for the computing time on the Cray T3E-900 and T3E-1200, respectively, used for comparison with the Parnass2 cluster.

References

[1] R. E. Bank and T. Dupont, *An optimal order process for solving elliptic finite element equations*, Math. Comp., 36 (1981), pp. 35–51.

[2] J. Barnes and P. Hut, *A hierarchical $\mathcal{O}(N \log N)$ force calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[3] P. Bastian, *Parallele Adaptive Mehrgitterverfahren*, Teubner, Stuttgart, 1996.

[4] D. M. Beazley and P. S. Lomdahl, *Message passing multi-cell molecular-dynamics on the connection machine CM-5*, Parallel Computing, (1994). Theoretical Division and Advanced Computing Laboratory, Los Alamos National Laboratory, Los Alamos, NM 87545.

[5] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[6] A. Caglar and M. Griebel, *Fast Parallel Algorithms for Molecular Dynamics*, Journal of Molecular Liquids, (1999). to appear.

[7] J. De Keyser and D. Roose, *Partitioning and mapping adaptive multigrid hierarchies on distributed memory computers*, Tech. Rep. TW 166, Univ. Leuven, Dept. Computer Science, 1992.

[8] J. Dubinski, *A parallel tree code*, New Astronomy, 1 (1996), pp. 133–147.

[9] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, 73 (1987), pp. 325–348.

[10] M. Griebel, *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen*, Skripten zur Numerik, Teubner, Stuttgart, 1994.

[11] M. Griebel and G. Zumbusch, Parnass*: Porting Gigabit-LAN components to a workstation cluster*, in 1. Workshop Cluster Computing 1997, TU Chemnitz, 1997.

[12] ——, *Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization*, in Proc. Domain Decomposition Methods 10, J. Mandel, C. Farhat, and X.-C. Cai, eds., vol. 218 of Contemporary Mathematics, Providence, Rhode Island, 1998, AMS, pp. 279–286.

[13] ——, *Parallel multi-grid in an adaptive PDE solver based on hashing*, in ParCo'97, E. D'Hollander, G. R. Joubert, F. J. Peters, and U. Trottenberg, eds., Elsevier, 1998, pp. 589–599.

[14] ——, *Parallel Adaptive Subspace Correction Schemes with Applications to Elasticity*, Computer Methods in Applied Mechanics and Engineering (CMAME), (1999). submitted.

[15] D. Hilbert, *Über die stetige Abbildung einer Linie auf ein Flächenstück*, Mathematische Annalen, 38 (1891), pp. 459–460.

[16] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981.

[17] P. Leinen, *Ein schneller adaptiver Löser für elliptische Randwertprobleme auf Seriell- und Parallelrechnern*, PhD thesis, Universität Dortmund, 1990.

[18] M. Lemke and D. Quinlan, *Fast adaptive composite grid methods on distributed parallel architectures*, Comm. Appl. Num. Methods, 8 (1992), pp. 609–619.

[19] J. T. Oden, A. Patra, and Y. Feng, *Domain decomposition for adaptive hp finite element methods*, in Proc. Domain Decomposition 7, vol. 180 of Contemporary Mathematics, AMS, 1994, pp. 295–301.

[20] M. Parashar and J. Browne, *Distributed dynamic data-structures for parallel adaptive mesh-refinement*, in Proceedings of the International Conference for High Performance Computing, 1995.

[21] A. Pothen, *Graph partitioning algorithms with applications to scientific computing*, in Parallel Numerical Algorithms, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., Kluwer, 1997, pp. 323–368.

[22] H. Sagan, *Space-Filling Curves*, Springer, New York, 1994.

[23] J. K. Salmon, M. S. Warren, and G. S. Winckelmans, *Fast parallel tree codes for gravitational and fluid dynamical n-body problems*, Int. Journal of Supercomputer Applications, 8 (1994), pp. 129–142.

[24] M. A. Schweitzer, G. Zumbusch, and M. Griebel, Parnass2*: A cluster of dual-processor PCs*, in 2. Workshop Cluster Computing 1999, TU Chemnitz, 1999.

[25] J. K. Singer, *The Parallel Fast Multipole Method in Molecular Dynamics*, PhD thesis, Dept. of Mathematics, University of Houston, Texas, August 1995.

[26] M. Warren and J. Salmon, *A portable parallel particle program*, Comput. Phys. Comm., 87 (1995), pp. 266–290.

[27] M. S. Warren and J. K. Salmon, *A parallel hashed oct-tree n-body algorithm*, Tech. Rep. LAUR 93-1224, Los Alamos National Laboratory, 1993.

[28] G. Zumbusch, *Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme*, SFB-Report 342/19/91A, TUM-I9127, TU München, 1991.

[29] ——, *Dynamic loadbalancing in a lightweight adaptive parallel multigrid PDE solver*, in Proceedings Parallel Processing for Scientific Computing 1999, SIAM, 1999.

[30] http://wissrech.iam.uni-bonn.de/research/projects/parnass2.

[31] *Advances in relational database technology for spatial data management*, white paper, Oracle Corp., 1997.