Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures

Thomas Pohl[1], Frank Deserno[1], Nils Thürey[1], Ulrich Rüde[1], Peter Lammers[2], Gerhard Wellein[3], and Thomas Zeiser[3]

[1] Lehrstuhl für Systemsimulation (LSS), University of Erlangen, Cauerstraße 6, D-91058 Erlangen, Germany
[2] Höchstleistungsrechenzentrum Stuttgart (HLRS), Allmandring 30, D-70550 Stuttgart, Germany
[3] Regionales Rechenzentrum Erlangen (RRZE), Martensstraße 1, D-91058 Erlangen, Germany

Corresponding author: thomas.pohl@informatik.uni-erlangen.de

4th November 2004

Summary:
Computationally intensive programs with moderate communication requirements such as CFD codes suffer from the standard slow interconnects of commodity "off the shelf" (COTS) hardware. We will introduce different large-scale applications of the Lattice Boltzmann Method (LBM) in fluid dynamics, material science, and chemical engineering and present results of the parallel performance on different architectures. It will be shown that a high speed communication network in combination with an efficient CPU is mandatory in order to achieve the required performance. An estimation of the necessary CPU count to meet the performance of 1 TFlop/s will be given as well as a prediction as to which architecture is the most suitable for LBM. Finally, ratios of costs to application performance for tailored HPC systems and COTS architectures will be presented.

1 Introduction

The Lattice Boltzmann Method (LBM) is a new computational alternative for simulating fluid flows and is rapidly gaining attention. It is an attractive method since it is based on a simple core algorithm which in turn makes it easy to adapt to complex application scenarios. Moreover, the base algorithm of the LBM can easily be extended to capture additional physical effects. Consequently, this method is being used as a universal tool in a rapidly increasing number of research projects. However, the flexibility of the LBM comes at a high price in terms of computational cost. which routinely requires the use of parallel supercomputers.

In this paper we will report on three challenging projects using the LBM on supercomputers. The first application is being used in nanotechnology to investigate the flow around agglomerates of particles. The second code simulates highly turbulent flows. The last application which is based on the nanotechnology code is currently being ported to supercomputers. It originates from material science where it helps to understand the flow of liquid metal during the production of metal foams.

These simulation projects are all based on the LBM and have originally been developed for three different target architectures. We will present and compare results of parallel simulation runs on

- Hitachi SR8000-F1,
- SGI Altix, and
- NEC SX6.

While this paper focuses on the parallel implementation of the LBM, we emphasize that all our applications are based on highly optimized computational kernels. The single processor tuning techniques are described and evaluated for a set of

0-7695-2153-3/04 $20.00

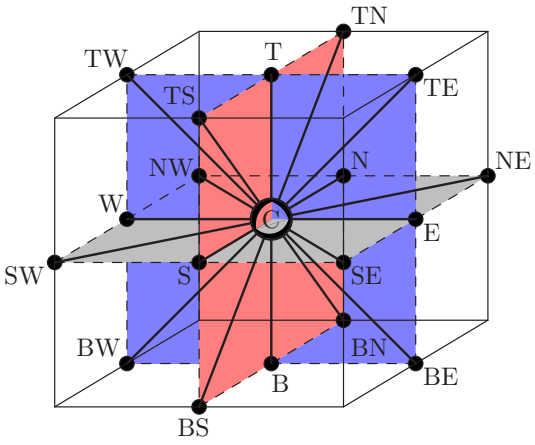

Figure 1: D3Q19 model. Velocities are labeled according to compass notation while T refers to "top" and B to "bottom"

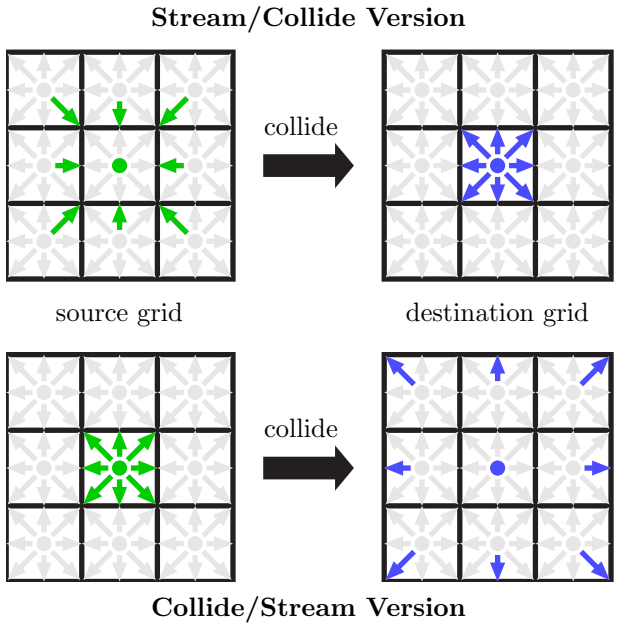

Figure 2: Fused stream/collide (top) and collide/stream step (bottom) for the center cell. After reading (green arrows) the required DFs from the source grid, new values are calculated in the collide step, which are stored in the destination grid (blue arrows). The only difference between the stream/collide and the collide/stream version are the memory access patterns

high performance architectures in [1]. For different COTS architectures similar methods and results are presented in [2] for 2D and in [3] for 3D LBM codes.

The major goal of parallelizing and tuning the codes, is clearly the reduction of simulation time. Besides presenting speed-up and scale-up[1] results, we will also try to predict which architecture is most effective for LBM-based simulations. A priori, it is not evident, whether a cluster of PCs with an inexpensive network most likely running only at a fraction of its peak performance is more efficient for executing LBM codes than a highly optimized supercomputer architecture like the NEC SX6, as a representative of a classical vector architecture, or the Hitachi SR8000, a system built on enhanced PowerPC chips that features a dedicated high performance network and a virtual vector node architecture [4].

Of course, these architectures are expensive, but they can possibly achieve a much higher fraction of their peak performance. We will present (extrapolated) results on how many of these nodes would be necessary to achieve a sustained TFlop/s of performance for the considered codes and give ratios of costs to performance for a variety of architectures.

2 The Lattice Boltzmann Method

The LBM is an extension of the lattice gas algorithms and approximates the Navier-Stokes equations with a cellular automaton. In contrast to conventional methods in fluid dynamics which are based on the discretization of macroscopic differential equations, the LBM follows a bottom-up approach by simulating the evolution of particle distribution functions (DF). A major advantage of the LBM is its ability to deal efficiently with complex geometries and topologies. Furthermore, it can be implemented easily and efficiently on parallel architectures.

The LBM operates on a lattice that consists of cubic cells of equal size, each of which stores particle distribution functions along a discrete number of velocity vectors. In 2D, models with 9 velocities (D2Q9) are in common use. Fig. 1 shows a discretization in 3D with 19 velocities [5] which is used in all introduced applications.

Each DF is stored as a floating point value and represents the number of particles in the current cell moving in the that direction. Density and velocity of the fluid in a cell can be calculated from these values by simple summations.

The LBM proceeds in two steps, the stream step and the collide step (also called propagation and relaxation). During the stream step, the particles are moved along their respective velocities, as shown in the lower half of Fig. 2. Normalizing the time step to

[1] scale-up: problem size per CPU is constant (weak scaling), speed-up: problem size is fixed (strong scaling)

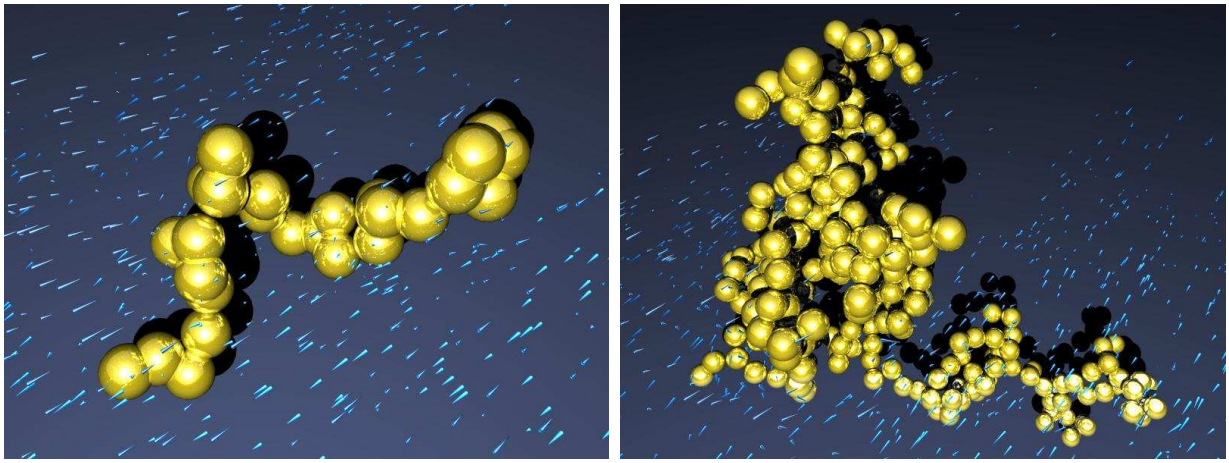

Figure 3: Two simulation setups for an agglomerate of 32 (left) and 256 particles (right). As can be seen from the movement of the tracers, the shear flow moves in opposing directions at the top and the bottom of the domain

one, the distance covered is exactly the distance to an adjacent cell. Thus all DFs are copied to the corresponding neighboring cells. The collide step subsequently computes the effect of the collisions which occur during the movement in the stream step. A new set of DFs is obtained from a weighted average of the streamed DFs with the equilibrium DFs; the latter are calculated from the fluid velocity and density in the cell. The global weighting factor is determined by the viscosity of the fluid [6].

Obstacles are handled by reflecting the particle DFs at the obstacle boundary, resulting in a normal and tangential velocity of zero (bounce back scheme).

3 Lattice Boltzmann Applications

The next section will illustrate with three examples the wide range of applicability of the LBM.

3.1 Nanotechnology

The detailed modeling and simulation of the behavior of rigid particles immersed in fluid flow has numerous important applications in chemical engineering. In the case considered here, particles are assumed to be of the size of 10s to 1000s of nanometers. They are modeled individually, while the flow is simulated by the LBM. Many interesting questions can be studied with such a simulation, e.g., the clustering of particles to agglomerates or the breakup of particle agglomerates. This project is still in the starting phase. However, the LBM is especially attractive for handling the dynamically changing flow domain due to particle movement.

Two examples of agglomerates provided by the LFG Institute Erlangen are shown in Fig. 3. The first represents a sintered agglomerate and consists of 32 particles while the second one is built of 256 particles before sintering. Each particle in an individual agglomerate is represented by a sphere. To compute the forces acting upon these agglomerates in a fluid, we expose them to a shear strain by initializing the flow as having two opposed velocities at the top and the bottom of the computational domain. The agglomerate structure which is placed near the center of the flow is represented by obstacle cells. Each particle is again approximated as a sphere, while in the simulation cubic obstacles are used to approximate the spherical shape. The forces acting on an agglomerate can be computed by integrating the pressure obtained from the LBM simulation over the surface area of the whole agglomerate [7].

3.2 Turbulence

The description of turbulence is a fundamental problem in flow engineering and theoretical research in fluid mechanics. Turbulent flows consist of large eddies as well as dissipating small structures and therefore cover a wide range of scales and energies. Sufficiently resolving all scales in a direct numerical simulation (DNS) is still a great challenge. Consequently, not all types of flows can be computed yet, nor is there any hope that they will be computable in the near future. However, turbulence research requires detailed information attained by numerical simulations and thus benefits from the development of new algorithms and the evolution of high performance computers. Therefore it remains an

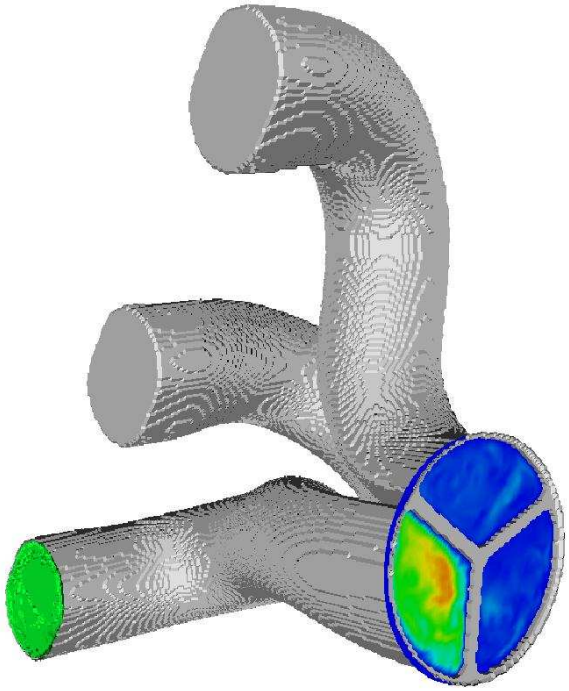

Figure 4: Large-eddy simulation of the flow through an exhaust system at $Re = 3 \cdot 10^5$. The computational grid is gained from CAD data by the marker-and-cell approach

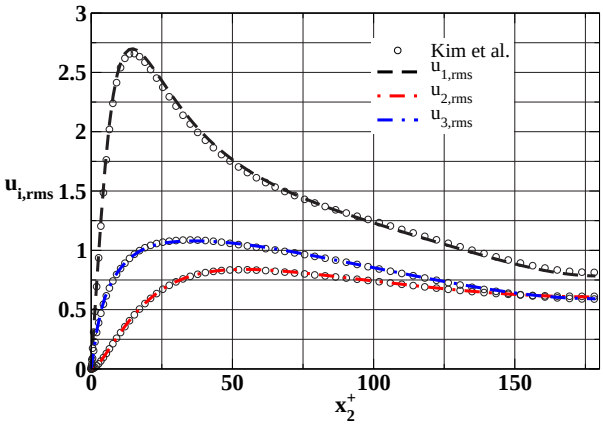

Figure 5: Comparison of LBM and pseudo-spectral results for a plane channel flow at $Re_\tau = 180$: RMS values of the three velocity components

open problem to develop a numerical method which presents the best compromise between the conflicting requirements of low computational cost and high numerical accuracy. Against this background the LBM has been studied as a candidate for turbulent flow computation at the Institute of Fluid Mechanics (LSTM) in Erlangen.

To illustrate the ability of the LBM to simulate turbulence, this section presents two examples. The first one is a technical application of the flow through an exhaust system. The aim here is the calculation of the pressure drop over the catalyst.

In Fig. 4 three exhaust manifolds are shown. Only one of them is charged with fluid. The Reynolds number which determines the ratio between convective and molecular transport is about $3 \cdot 10^5$. The higher the Reynolds number, the larger the range of scales that have to be resolved needs to be. Consequently, a large-eddy simulation (LES) is performed in this application. In a LES the large-scale structures are given by the Navier-Stokes equations whereas the influence of the scales smaller than the grid size is modeled by modifying the viscosity of the fluid in a more or less appropriate way. We use the popular Smagorinsky subgrid-scale model in which the modified viscosity depends on the shear stress tensor. The main advantage of the LBM framework is that this tensor can be computed locally at every grid node from the DFs without the need for numerical differentiations [8].

The geometry of the exhaust system can be easily described by the marker-and-cell approach. A voxel model is created from CAD data of the geometry and used together with the bounce back boundary condition. Details of this investigation can be found in [9].

The second example is an application of theoretical background: the DNS of wall-bounded turbulent flows. Direct numerical simulations have become the fundamental means for answering open questions in turbulence research because, in principle all required quantities can be extracted from the complete time dependent three-dimensional velocity and pressure fields. An example of DNS with LBM of a wall-bounded turbulent flow at low Reynolds number can be found in [10]. Here we present a comparison of the LBM and a pseudo-spectral simulation for a fully developed turbulent flow between two parallel plates with periodic boundary conditions in streamwise and spanwise direction. The geometry and the flow parameters have been chosen according to the case of Kim et al. [11]. In the reference code the equations are solved by Fourier transformation in streamwise and spanwise direction and by a Chebyshev series in wall-normal direction. The Reynolds number based on the channel width and the bulk viscosity is 5600 (which is equivalent to a Reynolds number of 180 based on the channel half-width and the wall shear velocity).

As seen in Fig. 5 the agreement with the reference data base [12] is very good. More information

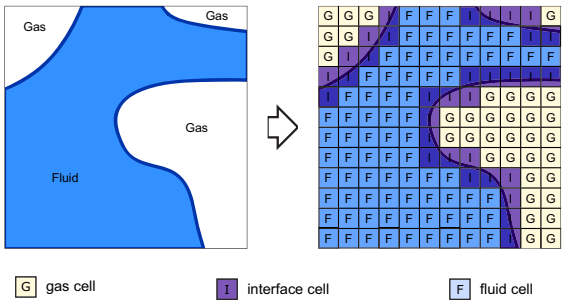

Figure 6: Discretization of the free surface topology for LBM

related to this test case is available in [13] and [14].

3.3 Metal Foams

The last example comes from material science. The goal is the development and high performance implementation of a model for the simulation of the formation process of metal foams

Metal foams are interesting as lightweight materials that have an excellent combination of mechanical, thermal and acoustic properties. However, the production process is currently not fully understood, and highly accurate numerical simulations are required to optimize the process parameters such as temperature and gas concentration to ensure the desired foam characteristics [15].

This project has as its primary target the Hitachi SR8000 architecture in Munich, and is a collaborative effort of the System Simulation Group (LSS) and the Institute of Science and Technology of Metals (WTM) at the University of Erlangen.

Within the model the liquid metal phase is treated as a Newtonian fluid. Its motion is caused by the foaming agent which in turn increases pressure and volume of the gas bubbles in the foam. We have developed a free surface model that simulates only the flow in the liquid metal phase, while the gas phase in the bubbles is treated in a very simplified way. This reduces the computational complexity and avoids stability problems full two-phase models might exhibit due to the large difference in viscosity of the gas and the liquid phase. During processing, the geometry and topology of the fluid domain changes very rapidly which can be treated easily within a LBM simulation. Despite the previously mentioned simplification, the simulations require large computational grids to capture the complex foam topology and to represent the low viscosity of the fluid phase correctly.

The gas and fluid phase of a metal foam furthermore require a distinction between cells that do not contain any fluid (gas cells), interface cells which are partially filled, and cells completely filled with fluid. An exemplary configuration is shown in Fig. 6. The fluid cells can be treated as described above, while no computation is necessary for gas cells. Only interface cells need to keep track of the amount of fluid that they currently contain. This can be done by computing the mass exchange with all neighboring fluid and interface cells, i.e., subtracting the outgoing DFs from the incoming ones during the stream step.

As gas cells are not included in the computation, all DFs which would be streamed from gas cells need to be reconstructed for the interface cells. This can be achieved by recalculating the equilibrium DFs using the velocity of the interface cell (the gas is assumed to have the same velocity as the fluid close to the interface) and the gas pressure which is determined by its initial value and subsequent volume changes.

A high gas density results in a pressure force at the fluid interface, pushing the fluid away, while a lower pressure applies a force towards the gas phase. This reconstruction step ensures that all DFs are known for interface cells. Therefore, these cells can be treated in the usual way during the collision step as described in Sec. 2.

Once interface cells become completely filled or empty – indicated by a normalized mass of zero or one, respectively – their type has to be adapted accordingly. This can also cause neighboring fluid or gas cells to be changed to interface cells, since the layer of interface cells has to be entirely closed. Otherwise neighboring gas and fluid cells would suffer from a loss of mass when DFs copied to the gas cell would not be included in the simulation anymore.

During the reconstruction step the surface tension can also be included by modifying the gas pressure according to the interface curvature, which is calculated from the geometry generated with a modified marching cubes algorithm [16]. This algorithm is often used to visualize isosurfaces from scalar fields, and generates a closed triangulated surface for a given isolevel. The points generated for the triangulation can be used to determine the curvature along two coordinate axis planes (chosen according to the approximated surface normal).

In Fig. 7 the simulation of a problem similar to foam formation is shown. Six bubbles with surface tension are rising in a container filled with fluid. Coalescence can be observed when larger bubbles below touch smaller ones.

Due to the geometrical nature of the algorithm, however, the curvature calculation can become inaccurate for very small curvatures resulting in unphys-

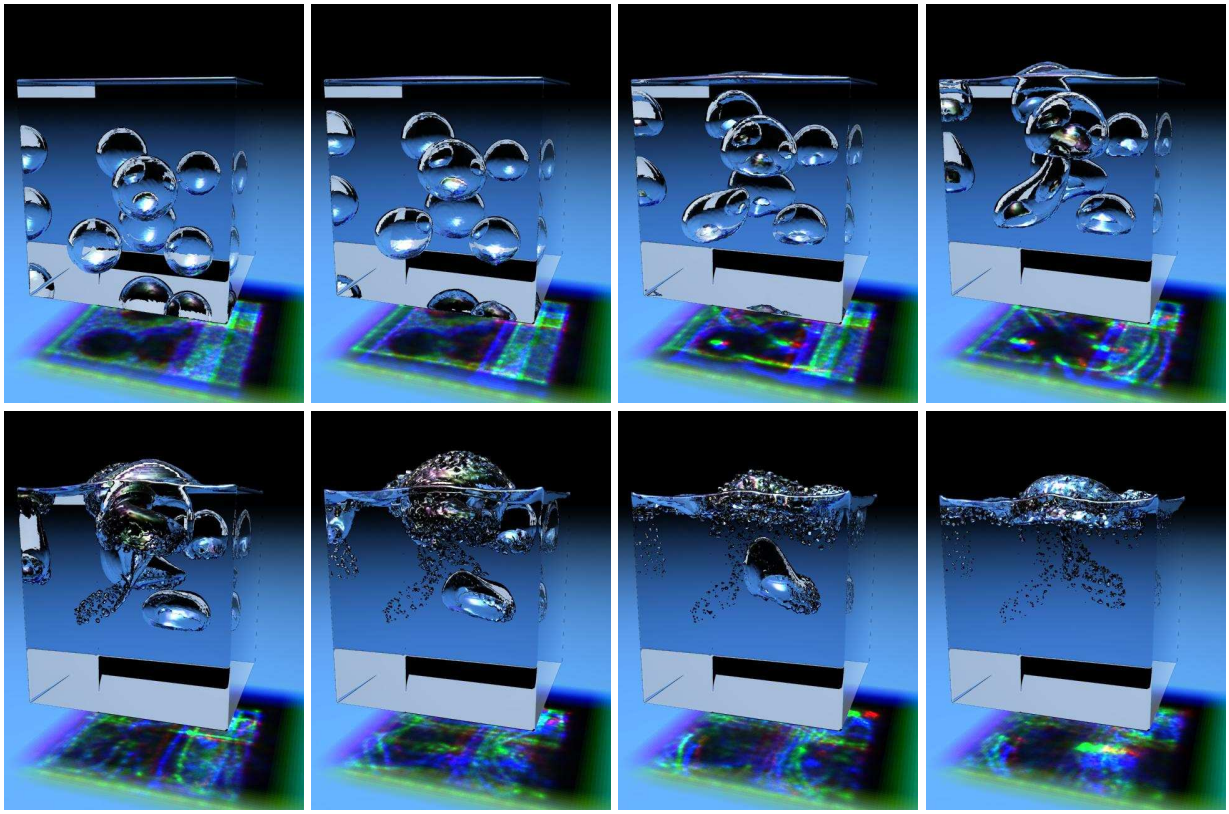

Figure 7: Six bubbles rising in a container filled with fluid due to a gravitational force

ical forces at the surface. To overcome the inaccuracies we are currently including the level set method [17] in our implementation. It treats the fluid surface as an implicit function, simplifying normal and curvature calculations as the level set values are also defined in a region around the interface. Still, more work has to be done to correctly track the interface.

4 Performance Evaluation

In the following sections we will use GFlop/s and million lattice site updates per second (MLup/s) as units for performance measurements. All presented codes have been specially tuned for their target architectures.

4.1 Nanotechnology code (NanoLBM)

Description of the Hitachi SR8000 architecture

Unlike many other supercomputers, the Hitachi SR8000 is not a classical vector computer, although it inherits some of the typical vector processing features. Its RISC CPUs [18], which are enhanced IBM PowerPC processors running at 375 MHz and a theoretical peak performance of 1.5 GFlop/s, are grouped in so-called nodes. Each node consists of nine CPUs of which eight can be used in application codes. Among other minor tasks the remaining processor handles I/O operations. These nine CPUs can access the local node memory of 8 or 16 GByte with 32 GByte/s. The nodes are connected with a 3D crossbar providing a network bandwidth of 2 GByte/s in full duplex mode [4]. The performance evaluation runs were performed on the Hitachi SR8000 at the Leibniz Rechenzentrum in Munich. This machine has 168 nodes and delivers a peak performance of 2.0 TFlop/s. It is placed at rank 127 in the TOP500 list of June 2004.

A major difference between this architecture and a normal RISC-based supercomputer is the extension to the CPU's instruction set. One important enhancement is the increase of the floating point registers to 160 which can be put to good use especially in scientific codes. Further extensions are special prefetch and preload instructions which both retrieve data from main memory and store it in the cache or in a floating point register, respectively. These instructions are issued by the compiler at appropriate points in the code to hide memory latency. Therefore, the compiler needs to know the memory access pattern of the code during compila-

tion. If the code does not allow for such an automatic memory access analysis, the application performance degrades drastically. In this sense, these RISC CPUs behave similar to vector CPUs which also rely on a simple and predictable data flow in the application.

The prefetch/preload mechanisms in combination with the extensive software pipelining done by the compiler are called Pseudo Vector Processing (PVP) as they are able to provide an uninterrupted stream of data from main memory to the processor circumventing the penalties of memory latency.

For parallelization issues several different software libraries are provided by Hitachi. If treated as MPP (Massively Parallel Processing/Processors) both intra- and inter-node communication can be handled with the standard Message Passing Interface (MPI). For intra-node communication there are two alternatives which exploit the benefits of the shared memory on a node more efficiently: the standardized OpenMP interface version 1.0 (see `http://www.openmp.org/`), and the proprietary COMPAS (Cooperative MicroProcessors in a single Address Space) which allows for easy thread parallelization of appropriate loops and sections.

In the following paragraphs we will describe our efforts to adapt the LBM kernel to these special requirements. Although the code implements a 3D model of the LBM with 19 DFs per cell, we will restrict the description and explanations to the 2D case with only 9 DFs for simplicity reasons. The extension to 3D is straight forward.

Description of the SGI Altix architecture

The SGI Altix architecture consists of shared memory ccNUMA systems with Intel Itanium2 CPUs. Our benchmarks were carried out on machines with 28 CPUs running at 1.3 GHz with an L3-cache of 3 MByte. Each processor has a peak performance of 5.2 GFlop/s. Four CPUs are grouped to a compute brick which in turn are connected by an SGI NUMALink3 interconnect. The configuration of each compute brick is depicted in Fig: 8. Each pair of CPUs shares a bandwidth of 6.4 GByte/s to local memory, while the bidirectional bandwidth between nodes amounts to only 1.6 GByte/s.

Memory layout and data access

The code optimizations presented here were applied to the nanotechnology code (NanoLBM) written in ANSI C. As explained in Sec. 2 the LBM is based on an equidistant Cartesian grid of cells each comprising a set of DFs. For reasons of simple and effective parallelization, we have chosen the memory layout

```
double grid[Y][X][N]
```

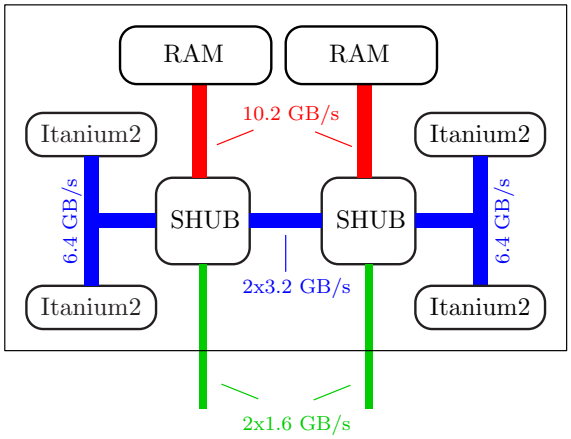

Figure 8: Schematic view of a compute brick of an SGI Altix

in C notation (`X` and `Y` denote the dimension of the domain; `N` is the number of DFs per cell), i.e., for each cell the `N` DFs are located in memory contiguously. To simplify the data access two complete grids are allocated. One serves as the source of all read accesses, the other one is used to store the new values. After a complete sweep over the grid, which corresponds to one time step, the two grids swap their roles.

The two basic LBM procedures which have to be executed in each time step (stream and collide) can be fused to a single block in order to decrease memory access. A choice has to be made about the ordering of the two involved actions, which has only a negligible influence on the computational results, but can lead to a significant difference in performance.

The stream/collide and the collide/stream version (see Fig. 2) both have in common that nine DFs of the source grid have to be loaded from memory in order to compute the new values in the collision step. Afterwards the set of nine new values has to be stored again to the destination grid in both cases.

However, there is a difference in the access pattern for reading and storing the values for both versions. Due to the chosen memory layout, the read data is located contiguously in the source grid for the collide/stream case, whereas it is scattered for the stream/collide version (vice versa for writing). On most architectures a contiguous data access is faster than a scattered one.

To perform the collision operations the reading of the values has to be completed. On the other hand, the successive cell updates do not depend on the data in the destination grid. Therefore the storing of the new values does not have to be completed be-

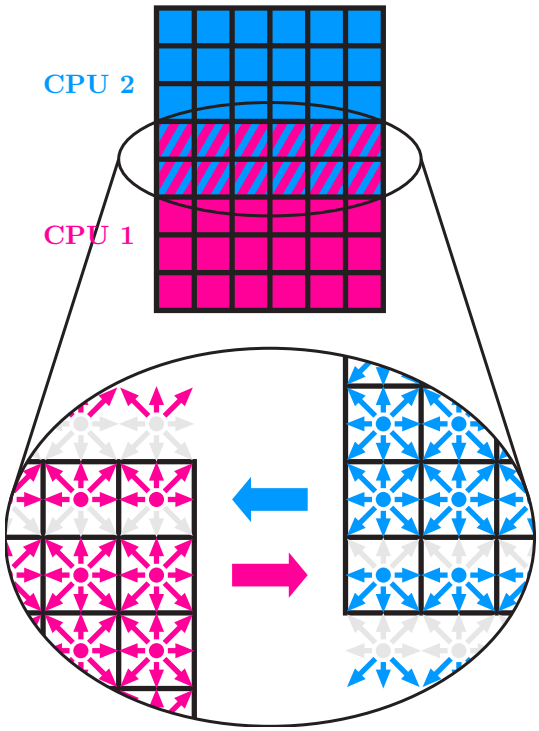

Figure 9: The upper part of the figure shows a simulation domain divided horizontally in two subdomains for parallelization. Due to the special stencil of the collide/stream operator (see Fig. 2) an overlap of two rows is necessary. The lower figure shows in detail the interface region of both subdomains (shifted left and right for easier depiction). CPU 1 sends the uppermost completely updated row to CPU 2 and CPU 2 sends the lowermost completely updated row to CPU 1

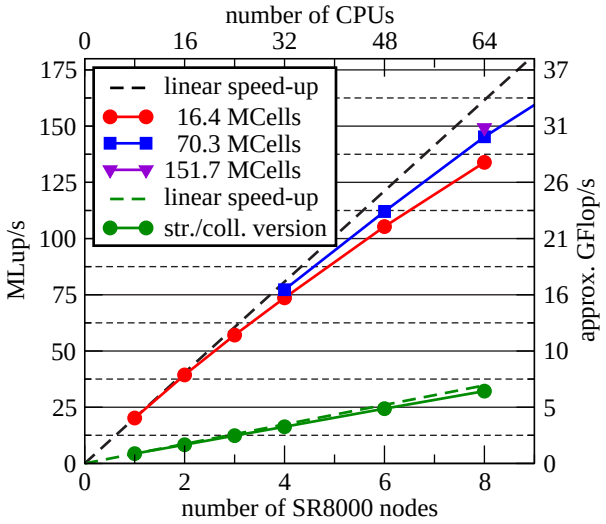

Figure 10: Comparison of the performance for different domain sizes on the Hitachi SR8000 (NanoLBM). The performance was measured in MLup/s. For our implementation a lattice site update corresponds to approximately 209 floating-point operations

fore starting work on the next cell. The C compiler on the Hitachi SR8000 is able to exploit this fact by applying PVP and produces code with better performance for the collide/stream case (see Fig. 10). The stream/collide version is about four times slower due to the lack of PVP.

Parallelization technique

Because of the high locality of the collide/stream operation, domain decomposition is very effective and widely used for LBM codes. Using a separate source and destination grid allows for optimizations like OpenMP or COMPAS intra-node parallelization for which independence of the sequence of cell updates is essential. In the presented code the intra-node parallelization with 8 CPUs per node is done with COMPAS. For inter-node communication we use MPI.

In the collide/stream version, the non-local write stencil makes it necessary to introduce an overlap of one row of cells at the interface of two adjacent subdomains (see Fig. 9). Each cell in both domains is updated by the collide/stream operator, even if some of the values are overwritten during communication afterwards. This is required to obtain completely updated rows at the interface which are then sent to neighboring domains. In total the number of cells is increased by a factor of

$$1 + \frac{2(N-1)}{\alpha s}$$

in the 3D case with N as the number of domains, a grid geometry of $s \times s \times \alpha s$, and for domain interfaces in the xy-plane.

For the largest domain handled by NanoLBM with a grid size of $646 \times 646 \times 2584$ cells and 64 subdomains this results in an increase of about 4.9 % additional cells that have to be updated in each step. To minimize this drawback of the collide/stream version the simulation domain should be divided into a minimal number of subdomains whose lower bound is determined for most architectures by the available memory.

The stream/collide version does not share the necessity of overlapping interface cells due to the difference in its operator stencil which explains its good speed-up behavior shown in Fig. 10.

A major advantage of the chosen memory layout for the grid are the contiguous data blocks for entire rows. Therefore rearranging data before sending and after receiving messages is unnecessary. In many other MPI-parallel applications this copy operations consume a considerable amount of proces-

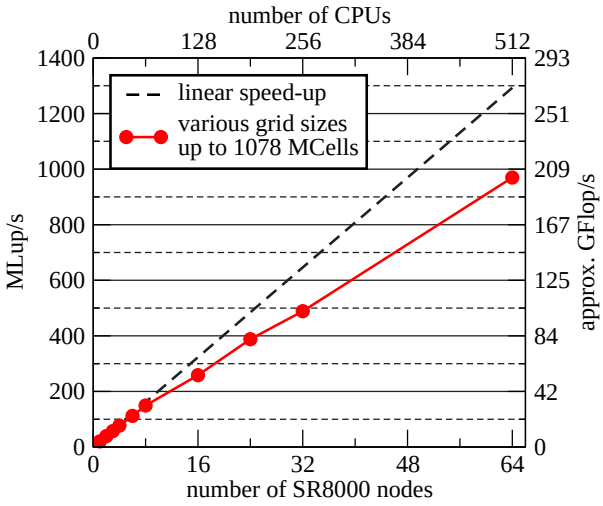

Figure 11: Scale-up performance on the Hitachi SR8000 (NanoLBM). The grid sizes were chosen to allocate most of the available memory on all nodes

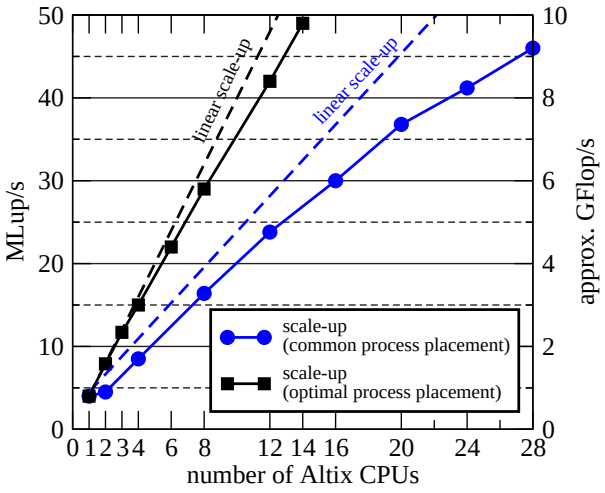

Figure 12: Scale-up and speed-up measurements on the SGI Altix architecture (NanoLBM). Depending on the process placement, different scaling behavior can be observed

sor time which could be spent for real computations otherwise.

Another feature of the presented code is its design which allows for overlapping of communication with computation in order to hide communication latencies if the architecture supports this. Instead of performing the collide/stream operator for the whole subdomain, the interface regions are updated first. After this step, the sending of the interface rows is initialized by MPI calls. Without waiting for their completion the rest of the subdomain is updated with the collide/stream operation. Before proceeding to the next time step, the corresponding receive operations have to be checked for completion. This gives the message handling subsystem of MPI more time to complete the pending communication calls.

Results for the Hitachi SR8000 and the SGI Altix

As can be seen in Fig. 10 the NanoLBM exhibits a good speed-up behavior for a small number of Hitachi SR8000 nodes which is close to linear scaling. When switching to a larger domain for the same number of nodes the performance increases because of the diminishing influence of the overlapping boundary interfaces which have been discussed previously. All performance results discussed in this section have been obtained with the collide/stream version.

Figure 11 shows the scale-up behavior for up to 64 Hitachi SR8000 nodes which amounts to 512 CPUs. For the largest simulation a total number of $1.08 \cdot 10^9$ cells have been used which require 370 GByte of memory in total. The effects of communication latency start to degrade the performance for such large-scale simulations as almost 64 MByte have to be sent to and received from each adjacent subdomain in every time step. Nevertheless, the code still achieves an efficiency of 75 % as compared to the single node performance.

For comparison, Fig. 12 shows the scale-up and speed-up behavior on the SGI Altix systems located at the Computing Center in Erlangen[2].

Within a compute brick a pair of two Itanium2 processors share one memory bus, which leads to an obvious degradation in performance when using both CPUs at the same time in a parallel run. Process placement can be handled accordingly. Hence, it is possible to obtain two different curves for speed-up and scale-up: one by continuously using all available CPUs in a compute brick and another by "intelligent" process placement and using only half of the available CPUs. However, on production systems such a process placement is often not possible so that the lower curve can be regarded as the "real" speed-up curve for the considered code.

A comparison of single CPU performance on the SGI Altix with the performance of one Hitachi SR8000 node reveals that 5–6 Itanium2 processors are necessary to meet the performance of one Hitachi SR8000 node.

4.2 Turbulence code (BEST)

While the NanoLBM code is written in C, the turbulence application called BEST (Boltzmann Equa-

[2] `http://www.rrze.uni-erlangen.de`

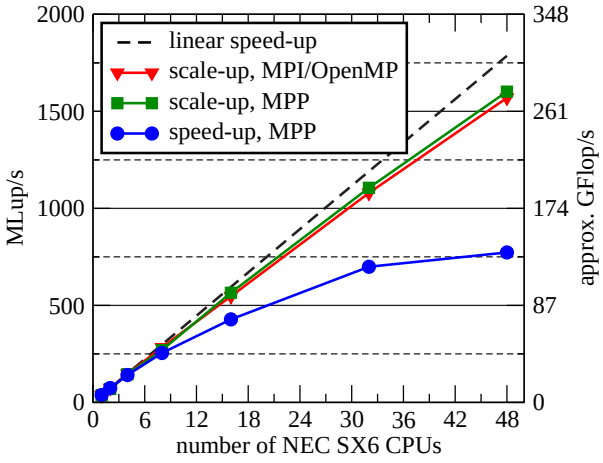

Figure 13: Scale-up performance on a NEC SX6 (BEST)

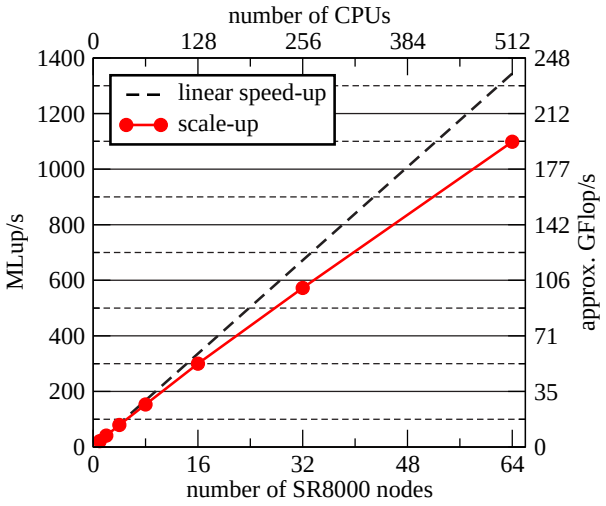

Figure 14: Scale-up performance on the Hitachi SR8000 (BEST) with 128^3 Cells/CPU and MPI/OpenMP parallelization

tion Solver Tool) which is discussed in this section has been implemented using FORTRAN.

Description of the NEC SX6 architecture

Whereas Hitachi tried to merge their S-3000 vector architecture and the MPP SR2201 into a new kind of high performance computer labeled SR8000, the Japanese vendor NEC embarks on the strategy of continuously improving the well established vector processing paradigm. The latest implementation of this paradigm, the NEC SX6, is based on a 565 MHz vector CPU with a peak performance of 9.2 GFlop/s and a memory bandwidth of 36 GByte/s per CPU. The vector unit consists of vector registers and 8 sets of vector pipelines for multiplication, add/shift, division, logical, mask, and load/store operations. Together with the vector unit a scalar unit is integrated on a single chip. This compact body improves the price performance of the architecture significantly. 8 CPUs share 64 GByte of memory with an aggregated memory bandwidth of 288 GByte/s. The nodes are connected by a 8 GByte/s crossbar switch (IXS). For our performance measurements we use the six node NEC SX6 with 48 vector CPUs installed at the High Performance Computing Center Stuttgart (HLRS).

Code optimizations

In oder to maximize the length of the loop for efficient vectorization the three spatial dimensions are collapsed into one in the BEST code. The memory layout is the same as for the NanoLBM code. To avoid data dependencies the DF array is also allocated twice. This simplifies the vectorization of the streaming step which is done immediately after the collision step. The collision step itself causes no problems as it is completely local. Within a shared memory node, the collision/streaming loop can be parallelized using OpenMP or equivalent vendor proprietary compiler directives. The inter-node parallelization implements domain decomposition and MPI for message passing. Instead of this hybrid approach, the architecture can be treated as an MPP using MPI also within each node.

In contrast to the NanoLBM implementation computation and communication are completely separated by introducing additional communication layers around the computational domain. Due to varying needs of the applications the domain decomposition is feasible in all three spatial directions.

Results for the NEC SX6 and the Hitachi SR8000

The resulting performance can be seen in Fig. 13 for a NEC SX6 with six nodes. In the plots three benchmarks are plotted. The first consists of a fixed grid size of 256×128^2 grid points. The performance is measured for up to 48 CPUs. On one CPU the LBM reaches almost 70 % of the peak performance. This value decreases to 3 GFlop/s per CPU on the entire machine for this fixed grid case.

For the second and the third benchmark the grid size is scaled with the number of CPUs. One CPU deals with 128^3 grid points. In this case the ratio of GFlop/s per CPU remains nearly constant regardless of the number of CPUs chosen. It can be seen that the pure MPI mode is slightly better than the hybrid/master-only mode for which OpenMP is used inside one node. In the figure the total values for MLup/s and GFlop/s are plotted. Additionally, the linear speed-up of the actual one CPU perfor-

mance is shown.

For comparison Fig. 14 shows the scale-up performance on the Hitachi SR8000 for the BEST code. It is nearly identical to performance of the NanoLBM code.

5 Scalability and cost effectiveness of COTS clusters

Finally, we discuss the scalability of the LBM application BEST on COTS clusters and comment on the usability and price/performance ratios in the context of the presented LBM applications.

We have chosen two flavors of COTS clusters: First, a GBit/Xeon cluster of 86 dual processor nodes (Intel Xeon 2.66 GHz; 4.3 GByte memory bandwidth per node) connected via a Cisco 4503 GBit Ethernet switch and running Debian GNU/Linux 3.0. All benchmark runs were compiled using the Intel Compiler version 7.1. Second, a Myrinet/Opteron cluster of 125 dual processor nodes (AMD Opteron 2.0 GHz; 5.4 GByte memory bandwidth per processor) connected via a Myrinet2000 network and running SuSE SLES 8 Linux. All benchmark runs on this system were compiled using the 64-bit PGI compiler version 5.0.

An important difference between the Intel and AMD design is the memory subsystem. While Intel still promotes bus based architectures where two or four processors share one path to the main memory, AMD uses a separate memory interface for each CPU providing full bandwidth for each CPU within a shared memory node.

Of course, the AMD design is favorable for the LBM programs concerning the scalability within the used dual processor nodes. In Tab. 1 we find a nearly linear speed-up within the AMD node, reducing the single processor performance gap substantially compared to the Itanium2 processor, if both processors on the dual nodes are used.

Architecture	1 CPU	2 CPUs
Intel Xeon	1.9	3.0
AMD Opteron	2.8	5.7
Intel Itanium2	5.0	7.2

Table 1: Speed-up performance within an 2-way node given in MLup/s (BEST)

Since the Opteron node provides a higher aggregate bandwidth (10.8 GByte/s) – which is the critical resource for LBM applications – than the Itanium2 systems (6.4 GByte/s) there should be some

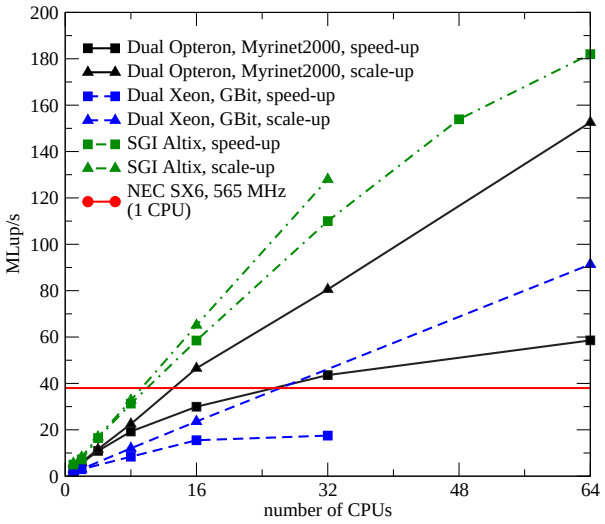

Figure 15: Scalability tests for modern cluster configurations (BEST). The domain size is $256 \times 129 \times 128$ for speed-up and 128^3 per processor for scale-up tests. For reference the corresponding results of a shared memory system (SGI Altix) and the single processor performance of the NEC SX6 are given

room for improvement by further optimizations or advances in compiler technology.

When discussing the impact of the cluster interconnects on the LBM performance we must distinguish between speed-up and scale-up performance: In the case of scale-up we find an almost linear increase in performance with the number of compute nodes involved. If we scale in units of dual nodes (which are the basic building blocks of our clusters) we find parallel efficiencies on 64 processors beyond 80 % both for GBit and Myrinet. As long as the network capacity is scaled up with the cluster size, e.g., with a fat-tree topology, we do not expect a significant breakdown for the scale-up case.

However, for the speed-up problem we recover the basic problems of cluster systems. With increasing processor count, the computational effort per processor decreases and network bandwidth and latency can impose severe limits on the achievable performance for a fixed problem to be solved. For example, on the GBit cluster reasonable scaling is not possible beyond 16 processors. Moreover, the cluster will never achieve the performance level of a single NEC vector processor for the problem size under consideration. Due to improved latencies and bandwidth, the Myrinet cluster scales significantly better but still requires roughly 24 processors to match the NEC single processor performance. The NUMA-Link3 interconnect of the SGI Altix provides a very high level of interconnect performance (roughly 5

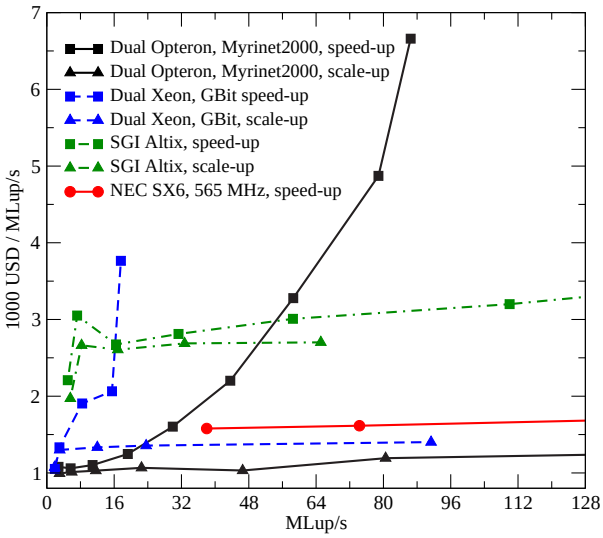

Figure 16: Costs per MLup/s to achieve a certain performance level (BEST). The domain size is $256 \times 129 \times 128$ for speed-up and 128^3 per processor for scale-up tests

times higher than the MPI bandwidth and 1/5 of its latency compared to Myrinet2000) and thus scales very well even for the speed-up case with a parallel efficiency of roughly 80 % on 64 processors (if using the two processor performance as the base and thus ignoring the bandwidth problem within the 2-way nodes as described in Tab. 1).

One reason for the success of COTS clusters is, that they provide immense aggregate peak performance and thus also high LINPACK numbers at a very moderate price level. However there is a controversial debate whether the price performance ratio of COTS clusters is still optimal if application performance is the metric. Using the performance data of Fig. 15 and price estimations based on public procurements of HPC systems[3] we present the price/performance comparison in Fig. 16.

To achieve a high application performance, architectures like dual Xeon or dual Opteron clusters become expensive compared to vector CPUs or the SGI Altix when the problem size is kept fixed. The step at the beginning of the SGI Altix curve shows once more the bandwidth problem already mentioned. In case the system size per CPU is constant, clusters of commodity hardware show a good price to performance ratio for this kind of code.

[3]estimated prices per processor: GBit/Xeon: 2000 USD; Myrinet/Opteron: 3000 USD; Altix/Itanium2: 11000 USD; NEC: 60000 USD

6 Conclusions

We have presented three large-scale applications which are based on the increasingly popular LBM. We explained in detail the optimization techniques which are required for the efficient use of parallel supercomputers. We were able to prove that these optimizations lead to excellent scale-up and speed-up behavior up to 512 processors on the Hitachi SR8000.

The results also demonstrate that only dedicated supercomputers featuring network and memory connections with extremely high bandwidth and low latency are suitable for these kinds of codes with large data sets. In turn, it is difficult or practically impossible to achieve the same performance and speed-up behavior even with several hundreds of processors even on architectures like the SGI Altix.

During the work on the application programs we experienced great differences in the quality of the code generated by different compilers. As a general rule it can be said that FORTRAN compilers are much more successful in producing highly efficient code compared to compilers for C or C++. Unfortunately, this is particularly true for compilers on most supercomputers. Nevertheless, it is possible to come close to the performance of FORTRAN codes as our C implementation NanoLBM has show.

A typical ccNUMA machine such as the SGI Altix performs reasonably well so that about five Itanium2 processors meet the performance of one Hitachi SR8000 node. However, the performance degradation due to insufficient memory bandwidth (one memory bus for two CPUs) reveals the weak point in the design of almost all SMP machines when considering memory bound applications like the LBM.

Requiring a sustained performance of 1 TFlop/s and extrapolating the presented performance numbers, 160 nodes of the NEC SX6, 320 nodes of the Hitachi SR8000 or roughly 3500 Itanium2 processors are necessary. Architectures like the Hitachi SR8000 or classical vector machines like the NEC SX6 or Cray X1 provide for high scalability up to several hundreds of processors and are therefore most suitable for large-scale parallel LBM applications. It is worth mentioning that the SR8000 was originally installed in 2000 and is still able to compete with current supercomputing architectures.

7 Acknowledgements

Part of this work is financially supported by the Competence Network for Technical, Scientific High

Performance Computing in Bavaria KONWIHR[4]. Furthermore, we would like to thank the LRZ in Munich and the CSAR in Manchester for providing access to their computing facilities. We acknowledge the helpful cooperation with our colleagues from LSTM, WTM, LFG at the University of Erlangen, and Georg Hager at the RRZE.

References

[1] Technical Report, RRZE, 2004. Available at `thomas.zeiser@rrze.uni-erlangen.de`.

[2] J. Wilke, T. Pohl, and M. Kowarschik and U. Rüde. Cache Performance Optimizations for Parallel Lattice Boltzmann Codes in 2D. In *Lecture Notes in Computer Science*, volume 2790, pages 441–450. Springer, 2003.

[3] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rüde. Optimization and Profiling of the Cache Performance of Parallel Lattice Boltzmann Codes. *Parallel Processing Letters*, 13(4):549–560, 2003.

[4] Horst D. Simon, C. William McCurdy, and William T.C. Kramer et al. Creating Science-Driven Computer Architecture: A New Path to Scientific Leadership, 2003. `http://www.nersc.gov/news/HECRTF-V4-2003.pdf`.

[5] Y. H. Qian, D. d'Humières, and P. Lallemand. Lattice BGK Models for Navier-Stokes Equation. *Europhys. Lett.*, 17(6):479–484, 1992.

[6] Dieter A. Wolf-Gladrow. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models*. Springer, 2000.

[7] A. J. C. Ladd. Numerical Simulations of particulate Suspensions via a discrete Boltzmann Equation Part 1. Theoretical foundation. *Journal of Fluid Mechanics*, pages 271–285, 1994.

[8] Shuling Hou, James D. Sterling, Shiyi Chen, and Gary Doolen. A Lattice Boltzmann Subgrid Model for High Reynolds Number Flow. *Fields Institute Communications*, 6:151–166, 1996.

[9] Christian Wimmer. Bewertung von Lattice-Boltzmann und Finite-Volumen Verfahren anhand von praxisrelevanten Aufgabenstellungen aus dem Bereich der Automobilaerodynamik. Master's thesis, Lehrstuhl für Strömungsmechanik, Universität Erlangen-Nürnberg, 2001.

[10] Peter Lammers, Jovan Jovanovic, and Franz Durst. Numerical experiment on wall turbulence. *Phys. Fluids*, 2003. submitted.

[11] John Kim, Parviz Moin, and Robert Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *J. Fluid Mech.*, 177:133–166, 1987.

[12] Robert Moser, John Kim, and Nagi Mansour. Direct numerical simulation of turbulent channel flow up to $\mathrm{Re}_\tau = 590$. *Phys. Fluids*, 11, 1999.

[13] Peter Lammers, Kamen Beronov, Gunther Brenner, and Franz Durst. Direct simulation with the lattice Boltzmann code BEST of developed turbulence in channel flows. In S. Wagner, W. Hanke, A. Bode, and F. Durst, editors, *High Performance Computing in Science and Engineering, Munich 2002*, pages 43–58. Springer, 2003.

[14] Peter Lammers, Kamen Beronov, Thomas Zeiser, and Franz Durst. Testing of closure assumption for fully developed turbulent channel flow with the aid of a lattice Boltzmann simulation. In *High Performance Computing in Science and Engineering, Munich 2004*, pages 77–92. Springer, 2004.

[15] C. Körner and R.F. Singer. Processing of Metal Foams - Challenges and Opportunities. *Advanced Engineering Materials*, 2(4):159–65, 2000.

[16] William Lorensen and Harvey Cline. Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm. In *Computer Graphics Vol. 21, No. 4*, pages 163–169, August 1987.

[17] S. Osher and J. Sethian. Fronts Propagation with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. In *Journal of Computational Physics 79*, pages 12–49, 1988.

[18] K. Shimada, T. Kawashimo, M. Hanawa, R. Yamagata, and E. Kamada. A Superscalar RISC Processor with 160 FPRs for Large Scale Scientific Processing. In *Proc. of International Conference on Computer Design*, pages 279–280, 1999.

[4] `http://konwihr.in.tum.de/index_e.html`