FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

Lehrstuhl für Informatik 10 (Systemsimulation)
www10.informatik.uni-erlangen.de

FreeWiHR — LBM with Free Surfaces

Carolin Körner, Thomas Pohl, Ulrich Rüde, Nils Thürey, and Torsten Hofmann

Lehrstuhlbericht 04-6

FreeWiHR — Lattice Boltzmann Methods with Free Surfaces and their Application in Material Technology

Carolin Körner[1], Thomas Pohl[2], Ulrich Rüde[2], Nils Thürey[2], Torsten Hofmann[1]

[1] Lehrstuhl für Werkstoffkunde und Technologie der Metalle, Universität Erlangen–Nürnberg, Martensstraße 5, D-91058 Erlangen, Germany
[2] Lehrstuhl für Systemsimulation (LSS), Universität Erlangen–Nürnberg, Cauerstraße 6, D-91058 Erlangen, Germany

Summary. 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. Therefore, the goal of the FreeWiHR project is the development and high performance implementation of a model for simulating the formation process of metal foams based on the lattice Boltzmann method.

1 Introduction

Foams show very interesting properties with respect to bending stiffness, energy absorption or damping behavior. Nearly every material can be foamed. However, the preparation of metal foams is a comparatively new field of research [1]. An example of an aluminum foam produced via the so-called powder metallurgical route is depicted in Fig. 1. Surprisingly, the physical understanding of foaming processes is yet very poor. This is particularly true for metal foam.

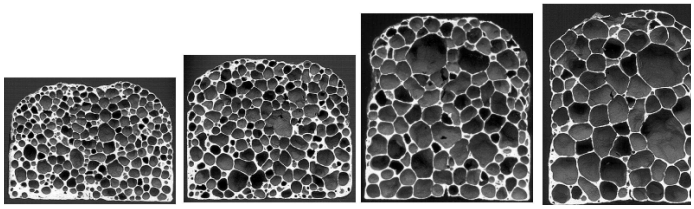

Fig. 1. Evolution of an aluminum foam structure

The main problem associated with the numerical simulation of foaming processes is the huge internal gas–liquid interface which strongly evolves with time. In addition, collapsing cell walls are able to induce avalanche-like coalescence and rearrangement processes of the whole foam structure. The time scale of these highly dynamic processes, which are governed by the Navier–Stokes equations (NSE), is typically much smaller than that of the foam expansion process itself.

A clear advantage of the LB approach compared to CFD lies in its local character, i. e. there are no global systems of equations which have to be solved. The computation time rises linearly with the system size. In addition, boundaries do not have a strong impact on the computation time. These features are essential regarding the complex internal structure of foams [2, 3].

The paper is subdivided into two parts. The first part describes a Lattice Boltzmann Model (LBM) for the simulation of foaming processes. The LBM comprises the underlying physical model and a new algorithm which has to be developed to treat 3D free surface problems within the LB approach. An example demonstrates the potential of the method. The second part describes the implementation and parallelization of the code for the SR8000.

2 Physical Model

The underlying physical model describes foaming by blowing agents including nucleation, bubbles growth, bubble coalescence and eventually foam collapse. The blowing agent releases gas which diffuses to bubble nuclei and leads to foam expansion which is in all stages intimately related with cell coalescence processes. Rupture of the cell walls occurs if their thickness falls below a critical value which is characteristic for the foaming material and is for metals about 20–50 μm.

The foam is considered in the liquid state; i.e, melting and solidification are not taken into account. Due to the large density difference between gas and liquid the two phase hydrodynamic system is reduced to a one phase system which describes fluid flow with free surfaces. That is, the exact dynamics of the gas is not taken into account. At the interface the gas pressure balances the hydrodynamic pressure. Bubble nucleation is assumed to be heterogeneous and statistical. Presently, gas diffusion is simply modeled by a continuous increase of the amount of gas within each bubble which is proportional to the bubble surface.

Pure melts do not foam. Capillary forces due to the surface energy rapidly destroy cell walls by thinning. Prerequisite for the development of a polygonal cell structure is the presence of a stabilizing mechanism. Generally, metal foams get stabilized by particles. The origin of the particles is quite different. They are either deliberately added to the melt or develop during foam preparation. The effect of these particles is to generate a restoring, stabilizing pressure, the disjoining pressure Π, if they are captured within a cell

wall. Both, the effect of the surface tension and the disjoining pressure are treated as a local modification of the gas pressure p^G at the gas–liquid interface $p^G \longrightarrow p^G - 2\,\kappa\sigma - \Pi$ where κ and σ denote the curvature and the surface energy, respectively. The disjoining pressure Π comprises the forces which stabilize the foam structure. Π is a function of the distance to the nearest neighboring interface d_{int}

$$\Pi(d_{int}) = \begin{cases} c_\Pi \left| d_{range} - d_{int} \right| \\ 0 \end{cases} \quad \text{for} \quad \begin{cases} d_{int} < d_{range} \\ d_{int} \geq d_{range} \end{cases} \tag{1}$$

where the magnitude and the range of the disjoining pressure are determined by the two phenomenological parameters c_Π and d_{range}.

In addition, a critical cell wall thickness is defined. If the wall thickness falls below that value, the cell wall ruptures and two bubbles merge.

3 Basic Equations

3.1 D3Q19 Model

We use the so-called D3Q19 model [4] which equilibrium distribution functions are defined as follows [5]

$$\begin{aligned} f_0^{eq}(\rho, \mathbf{v}) &= \tfrac{12}{36}\rho \left[1 - \tfrac{3}{2}\mathbf{v}\cdot\mathbf{v}\right] \\ f_i^{eq}(\rho, \mathbf{v}) &= \tfrac{2}{36}\rho \left[1 + 3\,(\mathbf{e}_i\cdot\mathbf{v}) + \tfrac{9}{2}(\mathbf{e}_i\cdot\mathbf{v})^2 - \tfrac{3}{2}\mathbf{v}\cdot\mathbf{v}\right], \quad |\mathbf{e}_i|^2 = 1 \\ f_i^{eq}(\rho, \mathbf{v}) &= w_2 = \tfrac{1}{36}\rho \left[1 + 3\,(\mathbf{e}_i\cdot\mathbf{v}) + \tfrac{9}{2}(\mathbf{e}_i\cdot\mathbf{v})^2 - \tfrac{3}{2}\mathbf{v}\cdot\mathbf{v}\right], |\mathbf{e}_i|^2 = 2. \end{aligned} \tag{2}$$

Numerically, the LBE is solved in two steps:

$$\textbf{Streaming} \qquad f_i^{in}(\mathbf{x}, t) = f_i^{out}(\mathbf{x} - \mathbf{e}_i\cdot\Delta t, t - \Delta t) \tag{3}$$

$$\textbf{Collision} \qquad f_i^{out}(\mathbf{x}, t) = f_i^{in}(\mathbf{x}, t) - \tfrac{\Delta t}{\tau}\left(f_i^{in}(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)\right) + \Delta t\cdot F_i \tag{4}$$

where τ is the relaxation time and F_i an external force. During streaming (3) all distribution functions but f_0 are advected to their neighbor lattice site defined by their velocity. After advection the particle distribution functions approach their equilibrium distributions due to a collision step given by (4). The incoming and outgoing distribution functions; i.e., before and after collision, are denoted with f_i^{in} and f_i^{out}, respectively. The macroscopic density ρ and momentum $\rho\mathbf{v}$ in a cell are the 0-th and 1-th moments of the distribution functions: $\rho = \sum_{i=0}^{18} f_i$ and $\rho\,\mathbf{v} = \sum_{i=1}^{18} f_i\,\mathbf{e}_i$

3.2 Free Surface and Fluid Advection

The description of the liquid–gas interface is very similar to that of volume of fluid methods. An additional variable, the volume fraction of fluid ϵ, defined as

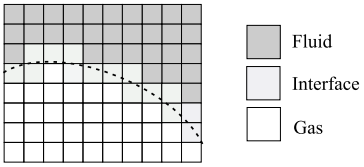

Fig. 2. 2D representation of a free liquid–gas interface by interface cells. The real interface (*dashed line*) is captured by assigning the interface cells their liquid fraction

the portion of the area of the cell filled with fluid, is assigned to each interface cell. The representation of liquid–gas interfaces is depicted in Fig. 2. Gas cells are separated from liquid cells by a layer of interface cells. These interface cells form a completely closed boundary in the sense that no distribution function is directly advected from fluid to gas cells and vice versa. This is a crucial point to assure mass conservation since mass coming from the liquid or mass transfered to the liquid always passes through the interface cells where the total mass is balanced. Hence, global conservation laws are fulfilled if mass and momentum conservation is ensured for interface cells. Per definition, the volume fraction ϵ of fluid and gas cells is 1 and 0, respectively. The fluid mass content of a cell is denoted with $M = M(\mathbf{x}, t)$. The mass content is a function of the volume fraction and the density. For a gas cell the fluid mass content M is zero whereas that of a fluid cell is given by its density ρ and the cell volume ΔV: $M(\mathbf{x}, t) = \rho(\mathbf{x}, t) \cdot \Delta V$ for $\mathbf{x} \in F$. Fluid cells gain and lose mass due to streaming of the f_i. For fluid cells M and ρ are equivalent. If interface cells are considered, M and ρ are not equivalent and we have to account for the partially filled state by introducing a second parameter, the volume fraction $\epsilon = \epsilon(\mathbf{x}, t)$. The fluid mass content M, the volume fraction ϵ and the density ρ are related by $M(\mathbf{x}, t) = \rho(\mathbf{x}, t) \cdot \epsilon \cdot \Delta V$ for $\mathbf{x} \in I$.

All cells are able to change their state. It is important to notice that direct state changes from fluid to gas and vice versa are not possible. Hence, fluid and gas cells are only allowed to transform into interface cells whereas interface cells can be transformed into both gas and fluid cells. A fluid cell is transformed into an interface cell if a direct neighbor is transformed into a gas cell. At the moment of transformation the fluid cell contains a certain amount of fluid mass M which is stored. During further development the interface cell may gain mass from or lose mass to the neighboring cells. These mass currents are calculated and lead to a temporal change of M. If M drops below zero, the interface cell is transformed into a gas cell. It is important to pronounce that mass and density are completely decoupled for interface cells. While the density of the interface cells is given by the pressure boundary conditions and fluid dynamics, M is determined by the mass exchange ΔM with the neighboring fluid and interface cells.

The mass exchange $\Delta M_i(\mathbf{x}, t)$ between an interface cell at lattice site $\mathbf{x}$ and its neighbor in $\mathbf{e}_i$-direction at $\mathbf{x} + \mathbf{e}_i$ is calculated as ($\mathbf{e}_{\bar{i}} = -\mathbf{e}_i$)

$$\Delta M_i(\mathbf{x},t) = \begin{cases} 0 \\ f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t) - f_i(\mathbf{x},t) \\ \frac{1}{2}\left[\epsilon(\mathbf{x},t)+\epsilon(\mathbf{x}+\mathbf{e}_i,t)\right]\left[f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t) - f_i(\mathbf{x},t)\right] \end{cases} \text{for} \quad \mathbf{x}+\mathbf{e}_i \in \begin{cases} G \\ F \\ I \end{cases} \tag{5}$$

There is no mass transfer between gas cells and interface cells. The interchange between an interface cell and a fluid cell should be the same as that of two fluid cells since the cell boundary is completely covered with liquid. In this case, the mass exchange can be directly calculated from the particle distribution functions. The interchange between two interface cells is approximated by assuming that the mass current is weighted by the mean occupied volume fraction. It is crucial to note that mass is explicitly conserved in (5):

$$\Delta M_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t) = -\Delta M_i(\mathbf{x},t). \tag{6}$$

That is, the mass which a certain cell receives from a neighboring cell is automatically lost there and vice versa. The temporal evolution of the mass content of an interface cell is thus given by

$$M(\mathbf{x},t+\Delta t) = M(\mathbf{x},t) + \sum_{i=1}^{18} \Delta M_i(\mathbf{x},t). \tag{7}$$

An interface cell is transformed into a gas or fluid cell if $M < 0$ or $M > \rho\,\Delta V$, respectively. At the same moment, new interface cells emerge in order to guarantee the continuity of the interface. The initial distribution functions of these new interface cells are extrapolated from the cells in normal direction towards the fluid.

The calculation of the local curvature of the interface is very complex and time consuming, see Fig. 3. In a first step, a marching cube algorithm is used to generate a triangulation of the interface. Secondly, the curvature κ belonging to each triangle is estimated by $\kappa = \frac{1}{2}\frac{\delta A}{\delta V}$ where δA denotes the alteration of the triangle area when its vertices are infinitesimally shifted in normal direction. The covered volume is denoted by δV. In the last step, the curvature of an interface cell is estimated by averaging the curvature of the triangles belonging to it.

3.3 Boundary Conditions

Interface cells separate gas cells from fluid cells. After streaming, only distribution functions from fluid and interface cells are defined. Distribution functions arriving from gas cells are not defined (see Fig. 4, left). The symmetry between known and unknown distribution functions; i.e., if f_i is known $f_{\bar{i}}$ is unknown, is essential to fulfill the boundary conditions. We demand force balance for opposite lattice directions. In addition, we make use of the fact that the forces exerted by the gas are known and are given by the gas pressure and the velocity at the interface. Hence, the missing distribution functions are reconstructed as

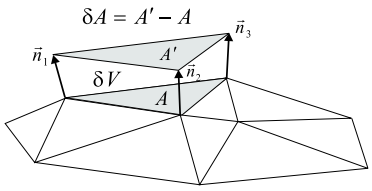

Fig. 3. Calculation of the curvature

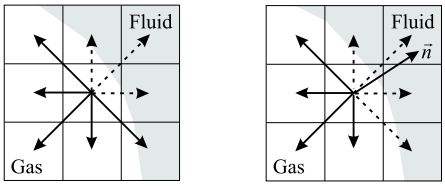

Fig. 4. Missing distribution functions at interface cells after streaming. Left: Undefined distribution functions after streaming (*broken lines*). Right: Set of distribution functions with $\mathbf{n} \cdot \mathbf{e}_i \geq 0$ (*broken lines*)

$$f_i^{out}(\mathbf{x} - \mathbf{e}_i, t) = f_i^{eq}(\rho^G, \mathbf{v}) + f_{\bar{i}}^{eq}(\rho^G, \mathbf{v}) - f_{\bar{i}}^{out}(\mathbf{x}, t) \qquad \forall i : \mathbf{n} \cdot \mathbf{e}_i \geq 0 \quad (8)$$

with the gas density $\rho^G = 3\,p^G$ and the velocity $\mathbf{v}$ of the interface cell.

It is important to note that not only the missing distribution functions are reconstructed but all distribution functions with $\mathbf{e}_i \cdot \mathbf{n} \geq 0$ (see Fig. 4, right). After completion of the whole set of distribution functions, the new density ρ and velocity $\mathbf{v}$ can be calculated. The outgoing distribution functions are calculated as

$$f_i^{out}(\mathbf{x}, t) = f_i^{in}(\mathbf{x}, t) - \frac{1}{\tau}\left(f_i^{in}(\mathbf{x}, t) - f_i^{eq}(\rho, \mathbf{v})\right) + \epsilon(\mathbf{x})\, w_i\, \rho\, \mathbf{e}_i\, \mathbf{g} \qquad \begin{matrix} i = 1, \cdots, b \\ \forall \mathbf{x} \in I \end{matrix}$$

where $w_0 = \frac{12}{36}, w_1 = \frac{2}{36}, w_2 = \frac{1}{36}$ and g is the gravity constant.

3.4 Example: 3D foam

The development of a 3D foam is depicted in Fig. 5. A large number of bubbles starts growing within the fluid. The disjoining pressure delays bubble coalescence but does not completely prevent it. Consequently, the number of bubbles decreases with increasing gas volume. At the end, a polygonal foam structure has developed.

4 Implementation and Parallelization

For testing and performance evaluation purposes we implemented a standard LBM code (SLBM). This code was used as a base for the free surface extensions (freeLB).

4.1 Data Layout

Due to the regular grids in LBM codes there are several possibilities for choosing the most appropriate data layout. Common to all is a simple array structure. A choice had to be made about the ordering of indices: three indices

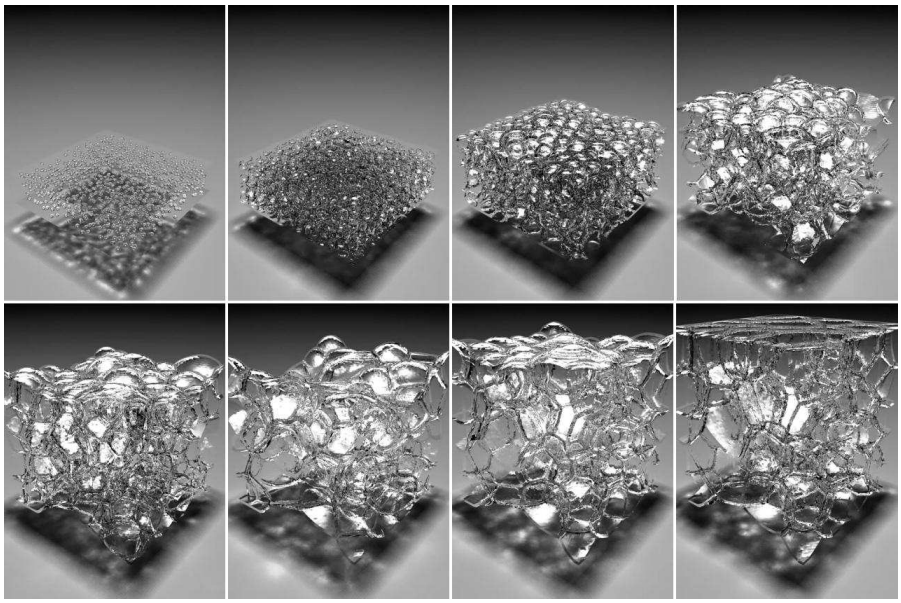

Fig. 5. 3D foam: The bubbles grow and coalescence occurs. The disjoining pressure Π stabilizes the foam and eventually a polygonal structure develops (initial number of bubbles: 1000; system size: $120 \times 120 \times 140$; $\tau = 0.8$; $g = 0$; $\sigma = 0.01$; $c_\Pi = 0.006$)

representing the coordinates (`X`, `Y`, `Z`), one index for selecting the data item in each cell (`I`), and one index determining the source or destination grid (`G`).

Although the mixing of the source and destination grid exhibited a performance increase on commodity PCs [6, 7], the performance decreased when applying the same data layout optimizations on the SR8000. Further investigations left three alternatives, that can be described in C notation as `double data[G][Z][Y][X][I]`, `double data[G][Z][Y][I][X]`, and `double data[G][Z][I][Y][X]`.

For testing purposes the first data layout where all data for one cell is stored contiguously in memory has been implemented as a standard LBM code (SLBM) in C [8]. As can be seen in Fig. 6 this code exhibits a good speed-up behavior for a small number of SR8000 nodes which is close to linear scaling. When switching to a larger domain for the same number of nodes, the performance per processor increases because of the diminishing influence of the overlapping boundary interfaces which will be discussed in Sect. 4.3.

Figure 7 shows the scale-up behavior for up to 64 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

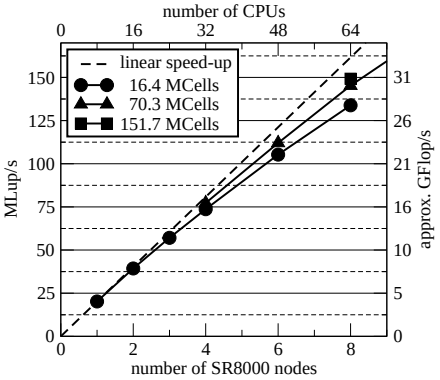

Fig. 6. Comparison of the performance for different domain sizes on the SR8000 (SLBM). The performance was measured in million lattice site updates per second (MLup/s). For our implementation a lattice site update corresponds to approximately 209 floating-point operations

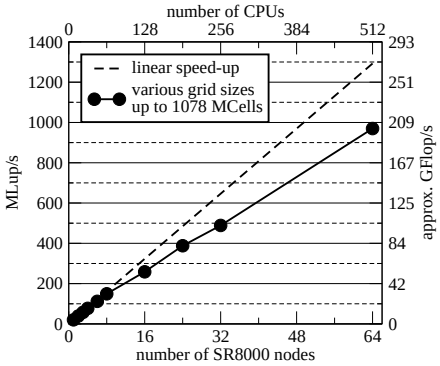

Fig. 7. Scale-up performance on the SR8000 (SLBM). The grid sizes were chosen to allocate most of the available memory on all nodes

adjacent subdomain in every time step. Nevertheless, the code still achieves an efficiency of 75 % as compared to the single node performance.

The second data layout has been tested by the HPC group at the RRZE with good performance results. For various reasons which will be described in Sect. 4.3 we chose the third data layout for the LBM code with the free surface extension (freeLB) based on the SLBM code.

4.2 Computations

In a standard LBM code with fused stream/collide step, a time step can be performed in a single sweep over the computational domain. For the more complex handling of free surfaces with the described method, however, five sweeps are necessary:

1. The first sweep starts with the well known streaming step augmented by the reconstruction of missing distribution functions from adjacent gas cells which takes the gas pressure of the concerned bubble into account. After that, the mass exchange with neighboring cells is calculated. For interface cells the new mass M leads to an updated fill value ϵ. This value determines whether an interface cell is now completely filled or emptied resulting in a conversion to a gas or fluid cell, respectively. Finally, the collision step according to the BGK approximation with an additional force term representing gravity is performed.

2. The second sweep checks and reestablishes the strict division of fluid and gas cells by the interface cell layer. Therefore, it might be necessary to undo cell conversions that have been scheduled in the first sweep.
3. In the third sweep converted cells have to initialized. Depending on the new cell type, a set of distribution functions (only for former gas cells) and the fill ϵ and mass value M are computed.
4. Interface cells converting to gas or fluid cells are hardly ever completely filled or empty. In almost every case, they still contain or miss a certain amount of mass. This possibly negative mass is distributed among the adjacent interface cells. Infrequently, no such cells can be found and the mass is lost. A better way to deal with this exception is to distribute the mass in a wider neighborhood.
5. During the last sweep, for all former and new interface cells, the change in the fill value ϵ is calculated and the volume of the bubble is updated.

Apart from the grid data the code also handles an array of data sets for each gas bubble in the computational domain containing the initial gas mass and the current volume of the bubble. This gas bubble data is not distributed among the involved CPUs, but stored entirely on each machine, because a single bubble can potentially span across all partitions of the computational domain.

4.3 Parallelization Technique

In general, domain decomposition is the canonical and most common way to parallelize LBM codes; i.e., the computational domain is divided up in several subdomains which are distributed to the computational units. Since information in the standard LBM can only travel one cell unit per time step, it is sufficient to have a halo (also known as ghost nodes) of one cell layer around any data adjacent to other subdomains. Due to the free surface extension with its several sweeps over the data a halo of four cell layers is necessary if the halo is updated only once per time step.

For the freeLB code we implemented a one dimensional domain partitioning; i.e., the computational domain is cut in slices parallel to the xy-plane. This allows for easy extensions as load balancing and can help to reduce the amount of data that has to be exchanged between neighboring subdomains in combination with the chosen data layout. Furthermore, the interesting domain sizes for real applications often feature a large spatial aspect ratio which attenuates the restrictions of a 1D domain partitioning.

The combination of the chosen data layout and the 1D domain partitioning features a major advantage compared to the other layouts. Without copying and/or reordering of data an entire cell layer in the xy-plane can be sent to or received from adjacent subdomains. By appropriately ordering the cell data it is even possible to communicate only a certain part of cell data; e.g, all

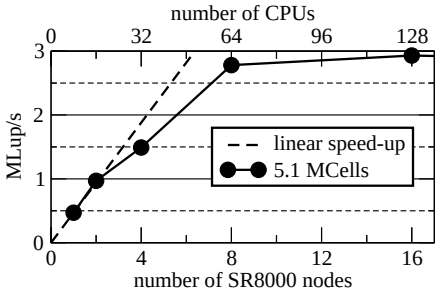

Fig. 8. Speed-up performance on the SR8000 (freeLB)

distribution functions pointing upward. The current implementation does not yet fully exploit this feature.

In addition to the communication of the grid data, the changes in the volume of the gas bubbles have to be collected and added for each subdomain to obtain the global volume change for each bubble. Therefore, the uppermost process starts to send its volume changes to its lower neighbor which adds its changes and sends this updated data to its lower neighbor. This procedure is continued until the lowermost process receives the volume update. Meanwhile the same chain of communication has been started at the lowermost process traveling upwards. When both chains reach the opposite end of the domain all processes have a global update of the gas bubbles' volumes. Instead of this procedure an "all to all" communication could be initiated using a dedicated MPI call. For a large number of subdomains, however, this would result in a large amount of send/receive events.

5 Performance Evaluation

In spite of the good performance of the SLBM code which was used as the base for the freeLB code with free surface handling, both the single node performance and the speed-up behavior exhibit a disappointing performance on the SR8000 as can be seen in Fig. 8.

The degradation of performance is caused by a combination of effects:

- Instead of just one layer of halo the freeLB code requires an update of four layers. The superior communication bandwidth of the SR8000 should be able to compensate the increased network traffic, but the additional workload caused by redundant cell updates degrades the performance particularly in speed-up performance tests.
- Profiling the code on an Intel Pentium 4 showed a significant increase in conditional branches. While the standard SLBM code requires only 2.9 conditionals per lattice site update, the new freeLB for handling the free surfaces needs as many as 51 conditional branches per cell update. The

performance on the Pentium 4 architecture is almost unaffected by the additional branches (SLBM: 2.5 MLup/s; freeLB: 2.2 MLup/s). The modified PowerPC architecture of the SR8000, however, draws its performance from a predictable and steady instruction and data stream based on its pseudo vector processing capabilities which can no longer be applied in the more complicated freeLB code.
- The compile logs indicate that the C compiler (`cc`) is no longer able to apply software pipelining in the freeLB code.

This all together leads to a performance degradation on each processor by a factor of 40 of the freeLB code as compared to the SLBM version. It is unclear at this time whether improved compilers could help to recover satisfactory node performance for codes as complicated as the present freeLB implementation. Manually restructuring the codes to avoid the conditionals in the innermost loops seems to be virtually impossible due to the dynamically changing geometry of the computational domains.

As reported previously, achieving high node performance on the SR8000 architecture is already quite nontrivial for the relatively simple SLBM code. From our experience, we therefore expect that at best moderate gains will be possible for the freeLB code.

6 Conclusions

The FreeWiHR project was based on a close collaboration of the Lehrstuhl für Werkstoffkunde und Technologie der Metalle and the Lehrstuhl für Systemsimulation at University of Erlangen-Nürnberg. It has achieved a number of ambitious project goals, including the

- development, implementation, and testing of a new algorithm for calculating the surface curvature in 3D which improved the stability and accuracy of the simulation.
- development, implementation, and testing of a model for the disjoining pressure of gas bubbles.
- implementation, performance tuning, and performance evaluation of a parallel LBM code for complex geometries using different data layouts and various optimization techniques.
- re-design of the algorithms for handling free surfaces that have been used in the sequential code in order to allow for parallelization.
- extending the parallel LBM code with the handling of free surfaces.
- performance evaluation of the parallel LBM code with free surface extensions.

The standard LBM code has been implemented in C using a conservative coding design which is well adapted to the peculiarities of the SR8000 architecture. Consequently, the SLBM code exhibits an excellent single-node

performance, scalability, and sustained performance on the SR8000. The more complex code with the free surface extensions, in contrast, suffers from a severe performance degradation caused by the large number of conditional branches which are unfortunately unavoidable when treating dynamically changing geometries within the LBM. At least to our current knowledge, and even assuming additional extensive profiling and optimization efforts, we cannot expect a significant improvement of the performance for this type of application on the SR8000.

A comparison with other architectures shows that the sustained performance of the freeLB code can be quite good: On an Pentium 4; e.g., freeLB runs essentially with the same performance as the SLBM code. We must therefore conclude that in the case of our algorithms for dynamically changing geometries, the SR8000 might not be the best choice of architecture.

Future work in the project will therefore concentrate on evaluating the suitability of alternative architectures, including clusters, and possibly also classical vector processors. Unfortunately, we must expect that standard clusters will suffer from insufficient communication bandwidth. Vector processors, on the other hand, may not be able to vectorize the freeLB codes easily. This is being evaluated in ongoing work.

Acknowledgements

We acknowledge the helpful cooperation with our colleagues from RRZE at the University of Erlangen, in particular Gerhard Wellein and Thomas Zeiser. This work has been financially supported by the Competence Network for Technical, Scientific High Performance Computing in Bavaria KONWIHR.

References

1. C. Körner, M. Thies, M. Arnold, and R. F. Singer. The Physics of Foaming: Structure Formation and Stability. In H. P. Degischer and B. Kriszt, editors, *Handbook of Cellular Metals*, pages 33–42. Wiley-VCH, 2002.
2. C. Körner, M. Thies, M. Arnold, and R. F. Singer. Modeling of metal foaming by in-situ gas formation. In J. Banhart et al., editor, *Cellular Metals. Manufacture, Properties, Applications*, pages 93–98. Verlag MIT Publishing, Bremen 2001.
3. C. Körner, M. Thies, and R. F. Singer. Modeling of metal foaming with Lattice Boltzmann Automata. *Advanced Engineering Materials*, 4:765–769, 2002.
4. X. He and L.-S. Luo. Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Physical Review E*, 6:6811, 1997.
5. Y.H. Qian, D. dHumières, and P. Lallemand. Lattice BGK Models for Navier-Stokes Equation. *Europhysics Letters*, 17:479–484, 1992.
6. J. Wilke, T. Pohl, 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.

7. 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.
8. T. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers, G. Wellein, and T. Zeiser. Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures. Supercomputing Conference 2004. Accepted.