Performance Analysis and Optimisation Procedures of a Parallel Fluid Dynamics Code for Thermal Comfort Assessment Applications

Jérôme FRISCH [1 a], Ralf-Peter MUNDANI [1], Christoph VAN TREECK [2], and Ernst RANK [1]
[1] Computation in Engineering, Technische Universität München, Germany
[2] Fraunhofer Institute for Building Physics IBP, Holzkirchen, Germany
[a] corresponding author address: *frisch@tum.de*

ABSTRACT

In this paper, we present the performance analysis of an existing parallel fluid dynamics code developed at the Chair for Computation in Engineering at Technische Universität München. Using the German national supercomputer HLRB II (SGI Altix 4700), performance factors were measured in order to discuss implications from the underlying hardware and to explore possible optimisation strategies.

Keywords: Parallel Computing, High Performance, Optimisation, Computational Fluid Dynamics Application

1. INTRODUCTION

Nowadays, thermal comfort assessment in early design stages of buildings and vehicles is a very important issue [9]. Evaluating the indoor climate in early design phases may not only indicate how comfortable human subjects might feel [3], but can furthermore economise valuable resources such as money and construction time. Issues of local thermal effects such as asymmetrical long wave radiation and air drafts can be identified and, hence, dealt with before the actual construction takes place in order to shorten the construction process and prevent from expensive subsequent changes.

In this paper, we focus on the performance analysis of a computational fluid dynamics (CFD) code, using a Lattice-Boltzmann-Method (LBM) that was implemented in order to evaluate local thermal effects of air drafts by computing the velocity field and temperature distribution in a closed environment [4, 10]. As realistic simulations – yielding to reliable physical results – entail a huge amount of computational effort, a parallel implementation of the LBM code is inevitable.

In a first step, typical properties of the parallel code were profiled and analysed running on more than 1000 nodes on the German national supercomputer HLRB II (SGI Altix 4700) in order to identify (communication) bottlenecks and weak aspects of the parallelisation.

Based on these results, we propose modifications for performance optimisation on the one hand as well as for hardware-awareness on the other hand to perfectly suit the underlying hardware architecture.

We close the paper with an outlook to our long-term objective of coupling the code with a numerical thermal manikin [9] for interactive computational steering purposes of thermal comfort assessment. In case of an interactive computation, the problem size will of course be smaller, in order to have a real time response of the simulation.

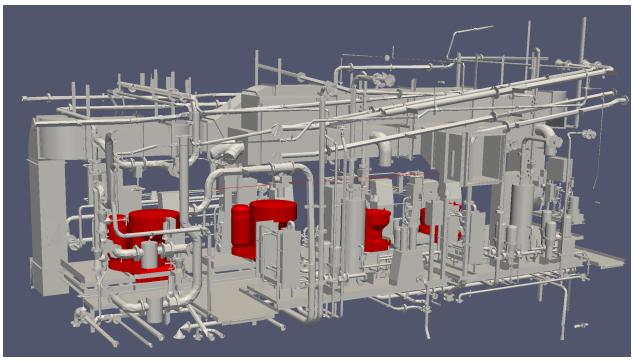


Figure 1: original geometry of the computational domain representing a part of an engine room in a ship containing four oil separators (highlighted)

2. PARALLEL CFD CODE

Computational Fluid Dynamics

As numerical method for solving the physical relations of turbulent convective airflows, represented by incompressible Newtonian fluids, the Lattice-Boltzmann-Method (LBM) is used. Unlike the classical method, which discretises the basic hydrodynamical equations (i. e. the Navier-Stokes-Equations), the Lattice-Boltzmann-Method is based on the concepts of statistical physics. It makes use of a first order finite difference approach in space and time resulting in a relatively simple scheme regarding implementation. To gain additional numerical stability, the so called multiple-relaxation-time (MRT) model by *d'Humières* [2] was used, and the simulation of convective airflows was achieved using the hybrid thermal model proposed by *Lallemand & Luo* [7]. For details refer to [4, 6, 10].

	spatial discretisation			amount of
	x	y	z	voxels [10^6]
(a)	100	37	50	0.185
(b)	200	75	100	1.500
(c)	300	112	150	5.040
(d)	400	150	200	12.000
(e)	500	187	250	23.375

Table 1: geometric discretisation of the computational domain

Computational Domain

Compared to the classical approach, the computational domain needs not necessarily to be meshed as precisely as required for a finite volume or finite element approach. In fact, it is sufficient to discretise the domain in the three spacial directions x, y, z forming a uniform Cartesian grid composed of voxels[1]. Applying this uniform discretisation, each voxel knows in advance the neighbouring relations in terms of the topological connections.

Furthermore, each geometry can be automatically converted without user interaction into a computational domain composed of a voxel geometry, whereas standard finite volume or finite element meshes usually need to be postprocessed after automatic generation by a specially trained engineer knowing about possible complications arising from gaps or overlaps in the final mesh, for instance.

The possibility of an automated domain generation leads to another beneficial advantage: In order to evaluate the performance of the code (which will be addressed in Section 3), computational domains of different sizes can be generated automatically. Table 1 gives an overview of some domain sizes and their respective spatial discretisation.

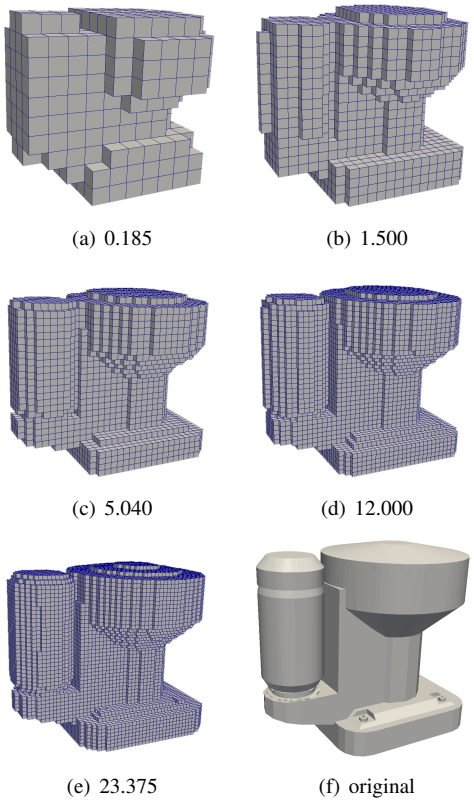
The voxel generator takes an arbitrary geometry and maps it to a uniform Cartesian grid with given dimensions using a tree based algorithm described in [13]. Furthermore, it can set the boundary conditions for the simulation.

Application

Figure 2(f) shows a part of the original geometry (depicted in Figure 1), a separator used in ships to control the quality of the engine's lub oil. In large ships, ferry boats, or container ships, e. g., it is ecologically and economically not possible to change the lub oil like in cars for example. Therefore, separators are used to clean the lub oil from dirt particles and other obstructions. A small quantity of new oil is added regularly, thus keeping the lub oil quality over the complete lifetime of a ship in good condition.

These separators usually become very hot, arising the question about the convective airflow inside the engine room while the separators are running. If convective cooling is sufficiently efficient, water cooling systems can be avoided, thus saving money in the design process [11]. Figure 4

[1] a voxel is the three-dimensional equivalent to a two-dimensional pixel



(a) 0.185 (b) 1.500

(c) 5.040 (d) 12.000

(e) 23.375 (f) original

Figure 2: separator according to spacial discretisation of Table 1. The labels display the overall amount of million voxels representing the size of the computational domain.

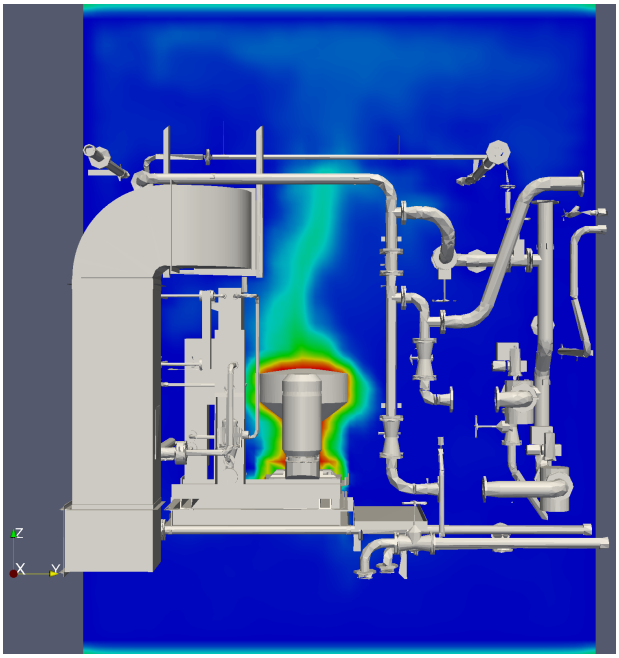


Figure 3: temperature distribution field through the hot separators as result of a CFD simulation (side view)

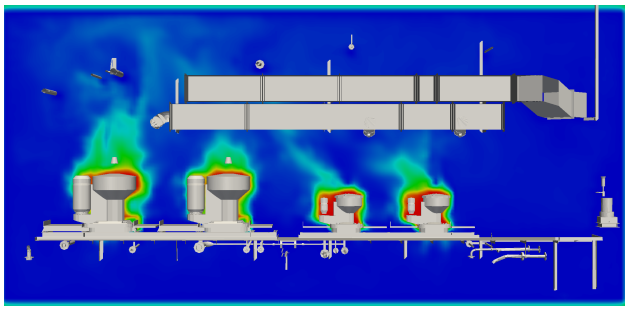


Figure 4: temperature distribution field through the hot separators as result of a CFD simulation (front view)

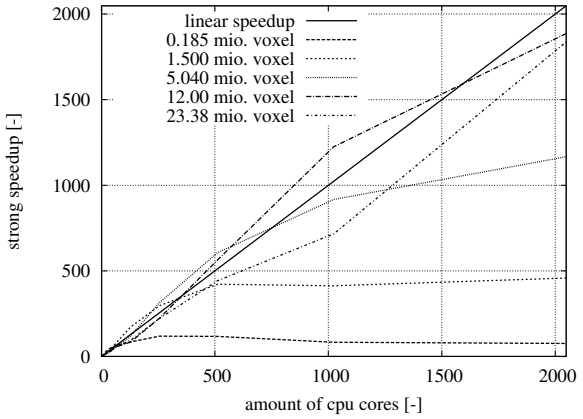


Figure 5: strong speedup depending on the amount of CPU cores used out of a range up to 2048

shows a computational result using a domain of 1.5 million voxels after half a million timesteps of simulation.
Figures 2(a) to 2(e) show different discretisation steps used for simulating the computational domain. It is obvious, that finer discretisations lead to better physical results. The domain containing 185,000 voxels gives only a very coarse description of the physical properties, whereas the domain containing 12 million voxels gives a quite accurate result.

3. PERFORMANCE ANALYSIS

The above described code was mainly used on a small-scale Linux cluster with maximum 16 processes. The first objective of the present paper was to port the code to the German national supercomputer SGI Altix 4700 with a total number of 9728 cores and 39 TByte (shared) memory of the entire system and to evaluate the performance – at first – without further optimisations. The results will be discussed in this section.
To obtain an overview over the parallel benefits at different simulation scenarios, several domain sizes using different numbers of CPU cores were analysed. Simulations were performed with a computational domain size ranging from

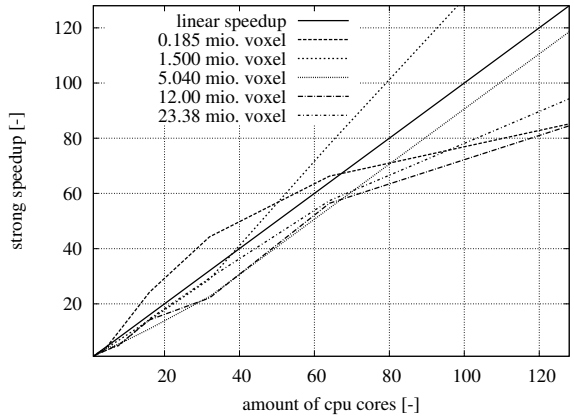


Figure 6: strong speedup depending on the amount of CPU cores used out of a range up to 128

approximately 185,000 voxels (type (a) in Table 1) up to 96 million voxels. Unfortunately, the results for a domain size larger than 23 millon voxels could not be taken into account for the strong speedup analysis, as it was not possible to get reference measurements for the serial code due to the long run time of this computation. An estimate of the overall time for the serial computation run of the 96 million voxel domain would be approximately 85 days.

Strong Speedup

For evaluating the performance of the code, a strong speedup analysis was made. The speedup for p processes is computed by dividing the measured time of the serial process (T_1) by the measured time using p processes (T_p):

$$S_p = \frac{T_1}{T_p} \quad . \tag{1}$$

Strong speedup in this context means that the number of processes is increased while the domain size remains constant. At some point the performance of the code decreases, as there is a growing part of communication overhead for the rising amount of processes.
The strong speedup results for the domains in Table 1 are depicted in Figure 5. The reference line indicates a linear speedup which – rarely achieved – means the optimal case. Obviously this is not reachable for the strong speedup due to the above mentioned constant domain size. One can expect that for all domain sizes a performance collapse is appearing at some point.
Figure 6 gives a detailed view of the first part from Figure 5 in order to give a better impression of the speedup behaviour at different problem sizes. Focussing on the domain containing 185,000 voxels, one can see in Figure 6 that this code performs exceptionally well for 2 to 64 processes, where a super linear speedup can be observed. This is mainly due to cache effects. The domain chunks assigned to each process

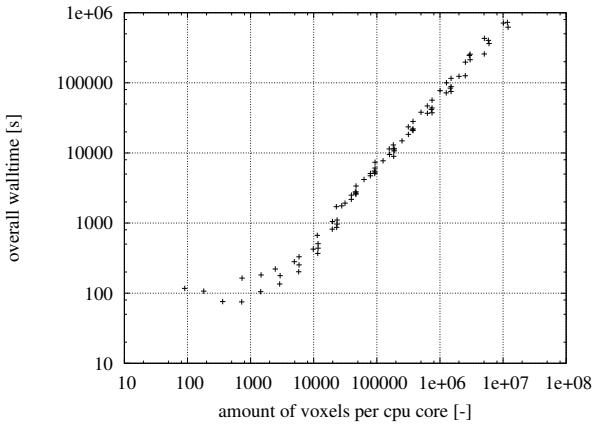


Figure 7: amount of voxels per CPU core versus overall walltime

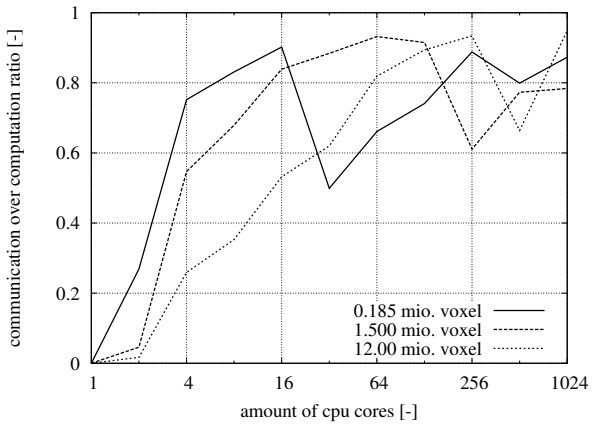


Figure 8: communication over computation ratio for 3 different domain sizes

are small enough to fit (entirely) into the respective cache. Hence, the cache-miss-rate decreases and the code performs better than in the serial case, even with occurring communication. As soon as the amount of CPUs exceeds a certain number of around 80 to 100 processes, the performance in terms of speedup drops drastically. Here, the assigned domain chunks become to small, thus the processes communicate much more than they compute, and a certain asymptote of around 100 is reached (Figure 5).

Looking closer at the other domains, we observe the same behaviour, except that the amount of processes, at which super linear behaviour and performance drops occur, are different. For the 1.5 million voxel domain, one can observe that until the amount of processes reaches approx. 38, the system is working with a lower rate than linear speedup. Here the domain chunks are still too big to achieve a high cache efficiency. As soon as the chunks get smaller, the cache efficient effects start to overwhelm the communication overhead. This is observable in all curves, even if the 23 million voxel example has not reached this state at 2048 processes yet.

Figure 7 gives an estimate of the necessary computation time for different domain sizes by comparing the amount of voxels per CPU core versus the overall computation walltime. Walltime in this context implies the real time elapsed from the beginning to the end of a job. Time measurements were taken with the same settings on eight different domain sizes with twelve different amounts of CPU cores to complete this diagram in order to get an accurate estimation of the code's run time behaviour. Performance issues can be observed at rates below 5000 voxels per CPU core where the walltime is mainly consumed by the initialization and finalization of the MPI processes distributed over several computing nodes.

Communication over Computation Ratio

The communication over computation ratio (CCR) is computed in this paper by dividing the sum of pure communication (and synchronisation) time $T_{com}^{(p)}$ of all slave processes p by the sum of the pure computation time of the application $T_{app}^{(p)}$ and the communication time:

$$CCR = \frac{\sum_p T_{com}^{(p)}}{\sum_p T_{com}^{(p)} + \sum_p T_{app}^{(p)}} \quad . \tag{2}$$

The run times for the CCR depicted in Figure 8 were measured and interpreted using VAMPIR [8] (Visualization and Analysis of MPI Resources).

Using a single process and running the code in serial mode, there is no communication and thus the ratio is zero. Using more processes, one can observe an increase in the communication over computation ratio induced by a higher communication effort. In this case, the rate of increase seems large due to necessary synchronisation rather than communication. The curves converge towards the value 1, as for a high number of processes the communication entails too much overhead, while the sum of all computation times stays nearly constant in case of fixed domain sizes for the CCR evaluation.

One can observe that the CCR drops for 185,000 voxels from 0.9 to 0.5 and rises again. This is mainly due to a quite specific reaction from the domain decomposition method used in the code. For this specific amount of voxels per CPU a quite ideal domain decomposition is existing which has a rather good surface per volume ratio[2] compared to other amount of processes. Furthermore, one can observe that the same behaviour exists for the domain size of 185,000 voxels and 1.5 million voxels. The first peak in the 185,000 voxel domain appears at approx. 5800 voxels per CPU while the first one in the 1.5 million voxel domain turns up at approx. 5850 voxels per CPU. This indicates that this behaviour is resulting from the code rather than the underlying architec-

[2] main matter for the communication when exchanging values at the domain boundaries with neighbouring process

ture, even if the architecture itself might have an influence on this effect.
Further studies are necessary to determine whether some improvements can be made to prevent this rapid inclining of the CCR or if an overall optimisation can be applied to reduce this specific effect.

4. OPTIMISATION STRATEGIES

As the performance analysis showed, the parallel implementation of the CFD code has not yet reached an optimal design. In this section, we discuss possible optimisation procedures, some of them already tested for the German national supercomputer.
One optimisation strategy on the Altix is called hybrid parallel programming. Until now, the code is parallelised using purely explicit Message Passage Interface (MPI) parallelisation. Another possibility would be to use an OpenMP based parallelisation concept, where code structures such as loops can be executed with several threads by adding so-called compiler directives. A mixture of both approaches is called hybrid parallelisation. The main structure of the already existing code is still communicating via MPI, while internal loops of one MPI process can be parallelised using OpenMP, which should suit the underlying shared memory structure of the Altix quite well.
Unfortunately, tests with code modifications using a hybrid parallel approach showed no run time improvement but a rather dramatical rising in run time behaviour. An analysis pointed out two major explanations for this behaviour: the independent loops, which can be parallelised in the slave computations are not very time consuming and lead to a high ratio of initialisation time versus actual loop runtime. The biggest, outermost loop in the slave process is formed by the main time loop over all the time steps, which cannot be parallelised by OpenMP due to dependency issues. Another fact is that by using the OpenMP concept in the independent loops, the cache efficiency of the slave process is drastically reduced. While fetching cache lines, the system has to jump to different places in the computation domain for different threads. If only one thread is used, the same cache line can be re-used more often, thus the cache efficiency increases and has a higher impact than the combined computation power of two threads with low cache efficiency. Hence, unless a complete remodelling of the code is done to increase the overall efficiency with respect to hybrid parallel programming, this scheme does not offer a suitable optimisation strategy.
Other optimisation ideas involve a redesign of the computational loops. Therefore the theory of the Lattice-Boltzmann-Method itself has to be evaluated with respect to possible mathematical transformations in order to rewrite formulations to increase efficiency, which is not the focus of this paper, but can be found in [12].
To further increase efficiency, another method of domain decomposition could be used. Until now, recursive bisection of the computational domain was applied, which is quite suitable for the Lattice-Boltzmann-Method with the underlying uniform Cartesian grid. Unfortunately, depending on the domain geometry, there can be domain chunks containing a lot of 'solid' voxels where no computation is done and which act only as boundary conditions. These domain chunks will have obviously a very low performance due to the fact that this computation node is merely waiting for the others, as its computation work is not very time consuming. In other words the domain chunk sizes per CPU node could be optimised using load balancing strategies paying the price of a non uniform distribution and a more complex communication scheme, as the neighbourhood information gets more difficult to manage like in the case of a uniform Cartesian distribution. As this strategy would involve major code changes, it was not tested in the scope of this paper but is thought as a mere outlook. Especially for highly non uniform distributions space-filling curves are advantageous for load balancing, as it has been shown with latest publications such as [1, 5].

5. APPLICATION COUPLING

A long term objective is to couple a numerical simulation of human thermoregulation with a CFD analysis. Thermoregulation describes the ability of the human body to adapt to the surrounding environment through means of active responses like vasoconstriction and -dilatation, sweating or shivering. Thermoreceptors at the skin and in other sensitive areas deliver values to the central nervous system which regulates all responses and tries to keep the body in a state of thermal neutrality. More information can be found in [9].
Doing a CFD computation, a human manikin in the computational domain behaves now different than an inanimate object like a piece of furniture, as these thermoregulatory aspects have to be considered. Here a bidirectional coupling is necessary, given that the room has an influence at the body and vice-versa. If the room is large compared to the body size and for certain boundary conditions, the bidirectional coupling can be neglected, and it is sufficient to compute an unidirectional coupling from the room to the manikin model. Stability issues of the coupling should not arise, as the human thermoregulation model is a quite inert model compared to the fluid dynamics behaviour of the room. Other problems as boundary treatments and exchange conditions from the fluid solver to the thermoregulation model will need thorough planning for a stable implementation. The code in its present state is not directly suited for such a coupling, as the slaves send data to the master process only in a very large interval for post processing reasons. The thermoregulation process, however, would need all the data from each slave at each computation step in order to fulfil the coupling conditions which might have a huge impact in efficiency, if the code structure stays unchanged.

6. CONCLUSIONS

In this paper, we presented the performance analysis of a parallel CFD solver based on a Lattice-Boltzmann-Method. In different graphs, performance measurements and evaluations were done to get an impression on the performance of the given code. There is still a lot of room for improvement concerning the communication over computation ratio which should be addressed and analysed in more details in a next step.
Hybrid parallel programming turned out not to be an option in the case of the given code, even if it fits well for the underlying architecture. Other promising optimisation strategies were mentioned, as well as the focus to couple a human thermoregulation model to a fluid computation of a room.

7. ACKNOWLEDGEMENTS

Parts of this work have been carried out with the financial support of KONWIHR II, the competence network for technical, scientific high performance computing in Bavaria.

8. REFERENCES

[1] M. Bader. **Exploiting the Locality Properties of Peano Curves for Parallel Matrix Multiplication**. 5168:801–810, Aug. 2008.

[2] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.-S. Luo. **Multiple-relaxation-time lattice Boltzmann models in three dimensions**. *Phil. Trans. R. Soc. Lond. A*, 360:437–451, 2002.

[3] P. Fanger. **Thermal Comfort**. Robert E. Krieger, Malabar, 1982.

[4] X. He and L.-S. Luo. **Lattice Boltzmann Model for the Incompressible Navier-Stokes Equation**. *J. Stat. Phys.*, 88(3/4):927–944, 1997.

[5] A. Heinecke and M. Bader. **Towards many-core implementation of LU decomposition using Peano Curves**. In *Conference On Computing Frontiers, UCHPC-MAW '09: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pages 21–30, New York, May 2009. ACM.

[6] M. Krafczyk, J. Tölke, and L.-S. Luo. **Large-eddy simulations with a multiple-relaxation-time LBE-model**. *Int. J. Modern Physics B*, 17:33–39, 2003.

[7] P. Lallemand and L.-S. Luo. **Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions**. *Physical Review E*, 68(036706), 2003.

[8] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. **VAMPIR: Visualization and Analysis of MPI Resources**. *Supercomputer*, 12:69–80, 1996.

[9] C. van Treeck, J. Frisch, M. Pfaffinger, E. Rank, S. Paulke, I. Schweinfurth, R. Schwab, R. Hellwig, and A. Holm. **Integrated thermal comfort analysis using a parametric manikin model for interactive real-time simulation**. *J Building Performance Simulation*, 2009.

[10] C. van Treeck, E. Rank, M. Krafczyk, J. Tölke, and B. Nachtwey. **Extension of a hybrid thermal LBE scheme for Large-Eddy simulations of turbulent convective flows**. *Computers and Fluids*, 35:863–871, 2006.

[11] C. van Treeck, P. Wenisch, A. Borrmann, M. Pfaffinger, O. Wenisch, and E. Rank. **ComfSim - Interaktive Simulation des thermischen Komforts in Innenräumen auf Höchstleistungsrechnern**. *Bauphysik*, 29(1):2–7, 2007. DOI: 10.1002/bapi.200710000.

[12] G. Wellein, T. Zeiser, G. Hager, and S. Donath. **On the single processor performance of simple lattice Boltzmann kernels**. *Computers & Fluids*, 35(8-9):910 – 919, 2006. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

[13] P. Wenisch, C. van Treeck, A. Borrmann, E. Rank, and O. Wenisch. **Computational steering on distributed systems: indoor comfort simulations as a case study of interactive CFD on supercomputers**. *Int. J. Parallel, Emergent and Distributed Systems*, 22(4):275–291, 2007.