Parallel Computing 46 (2015) 1–13

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Performance modeling and analysis of heterogeneous lattice Boltzmann simulations on CPU–GPU clusters

Christian Feichtinger [c,*], Johannes Habich [b], Harald Köstler [a], Ulrich Rüde [a], Takayuki Aoki [c]

[a] Chair for System Simulation, University of Erlangen–Nuremberg, Germany
[b] Regional Computing Center Erlangen, University of Erlangen–Nuremberg, Germany
[c] Aoki Laboratory, Tokyo Institute of Technology, Japan

ARTICLE INFO

Article history:
Received 8 January 2013
Received in revised form 12 June 2014
Accepted 15 December 2014
Available online 22 December 2014

Keywords:
Lattice Boltzmann method
CUDA
Heterogeneous computations
Performance modeling

ABSTRACT

Computational fluid dynamic simulations are in general very compute intensive. Only by parallel simulations on modern supercomputers the computational demands of complex simulation tasks can be satisfied. Facing these computational demands GPUs offer high performance, as they provide the high floating point performance and memory to processor chip bandwidth. To successfully utilize GPU clusters for the daily business of a large community, usable software frameworks must be established on these clusters. The development of such software frameworks is only feasible with maintainable software designs that consider performance as a design objective right from the start. For this work we extend the software design concepts to achieve more efficient and highly scalable multi-GPU parallelization within our software framework waLBerla for multi-physics simulations centered around the lattice Boltzmann method. Our software designs now also support a pure-MPI and a hybrid parallelization approach capable of heterogeneous simulations using CPUs and GPUs in parallel. For the first time weak and strong scaling performance results obtained on the Tsubame 2.0 cluster for more than 1000 GPUs are presented using waLBerla. With the help of a new communication model the parallel efficiency of our implementation is investigated and analyzed in a detailed and structured performance analysis. The suitability of the waLBerla framework for production runs on large GPU clusters is demonstrated. As one possible application we show results of strong scaling experiments for flows through a porous medium.

1. Introduction

For the modeling of complex engineering tasks parallel simulations are the only feasible option, especially in the multi-core area. The capital investment for new massively parallel clusters can be in the order of millions of euros, so that the need for an efficient exploitation of the hardware should be self-evident. Consequently, software designs for HPC (high performance computing) codes must consider performance right from the start. In order to achieve optimal serial performance as well as good scalability a deep understanding of the data and program flow, the bottlenecks, and the hardware is required. One possibility to gain the required level of insight is the prediction of the performance with the help of performance models. In addition, performance models can also be applied for the mandatory validation of the performance.

* Corresponding author.
E-mail address: Christian.Feichtinger@informatik.uni-erlangen.de (C. Feichtinger).

http://dx.doi.org/10.1016/j.parco.2014.12.003
0167-8191/

GPU architectures offer high floating performance and memory to processor chip bandwidth and promise a performance gain for LB simulations compared to CPUs. In order to utilize heterogeneous CPU–GPU clusters for complex engineering tasks, efficient and scalable software frameworks must be established. Amongst others this requires reusable and maintainable software design concepts, since parallelizing algorithms is a non trivial and time consuming task. In this work the performance of heterogeneous LB (lattice Boltzmann) simulations is investigated on the Tsubame 2.0 cluster [1]. The LBM (LB method) has evolved over the last two decades and is today widely accepted in academia and industry as an alternative for simulating incompressible flows. LBM is particularly well suited for massively parallel simulations [2–4] and efficient LBM codes for single GPUs have been described in [5–7]. Multi-GPU implementations of the LB method have been published in [8,9].

waLBerla[1] is a parallel multi-physics framework for simulating complex flows with the LBM method [10]. Its major aims are to aid scientists to focus on their simulation task and to provide a platform for collaboration and knowledge transfer. Amongst others, waLBerla has been used for free-surface flows [11], particulate flows on up to 294.912 compute cores [12], and self-propelled particle complexes [13]. The creation of scalable multi-GPU implementations is one of the crucial steps towards real-world simulations of complex engineering problems using GPUs. For this purpose the minimization of the communication overhead is essential. Our software design offers a good balance between maintainability and performance, so that it can be applied for the development of sustainable software frameworks.

In this work we use the term heterogeneous simulations for simulations using CPUs and GPUs in parallel. Recently, there is a trend towards heterogeneous compute nodes, e.g. compute nodes including GPUs or the Intel Xeon Phi. Hence, a implementation strategies are of interest, which exploit these compute nodes as efficiently as possible. This article addresses this point and extends our previous work on multi-GPU LBM implementations[14] with several major improvements for using heterogeneous compute nodes:

- Scalability to more than 1000 GPUs can now be demonstrated. This requires static load balancing and overlapping of communication and communication. The software design concepts for heterogeneous multi-GPU simulations were adapted accordingly.
- A pure-MPI and an hybrid OpenMP-MPI parallelization approach have been designed and were meticulously analyzed.
- A new predictive performance model is developed and validated to analyze communication overhead.
- We perform parallel multi-GPU and heterogeneous LB simulations with $256 \times 256 \times 3600$ lattice cells for realistic scenarios on more than 1000 GPUs of the Tsubame 2.0 supercomputer. Here, we are able to achieve almost 80 billion lattice cell updates per second (GFLUPS).

For the performance analysis, a model is derived to obtain an upper bound for the sustainable single compute node performance of LB simulations. In addition, we model the communication overhead to gain a deeper understanding of the data flow of parallel simulations. With our approach we profiled efficiency and the scalability of parallel implementations in order to find bottlenecks and hidden overheads. Further, the performance of parallel simulations can be predicted, e.g., to determine the optimal number of compute nodes for strong scaling scenarios.

This article gives a short introduction to the LB method in Section 2 and presents software design concepts for parallel simulations in Section 3. Section 4 provides information on the Tsubame 2.0 cluster and compilers, libraries, and the simulation setup. This is followed by the introduction of the performance model and a detailed analysis of the communication overhead in Section 5. In Section 6 the serial and parallel performance results of LB simulations on the massively parallel GPU cluster Tsubame 2.0 are discussed. Finally, we draw our conclusions in Section 7.

2. The lattice Boltzmann method

As a velocity discrete Boltzmann equation with an appropriate collision term, the LBM satisfies the Navier–Stokes equations in the macroscopic limit with second order accuracy [15,16]. In contrast to conventional usually finite element or finite volume based computational fluid dynamic methods, LBM uses a set of particle distribution functions (PDF) in each cell to describe the fluid flow. A PDF is defined as the expected value of particles in a volume located at lattice position $\vec{x}$ with lattice velocity $\vec{e}_i$. Computationally, the LBM is based on a uniform grid of cubic cells that are updated in each time step using an information exchange with nearest neighbor cells only. Structurally, this is equivalent to an explicit time stepping for a finite difference scheme. For LBM the lattice velocities $\vec{e}_i$ determine the finite difference stencil, where i represents an entry in the stencil. Here, we use the so-called D3Q19 model resulting in a 19 point stencil and 19 PDFs in each cell. The evolution of a single PDF f_i is described by

$$f_i(\vec{x}+\vec{e}_i\Delta t, t+\Delta t) = f_i^{\text{coll}}(\vec{x},t) = f_i(\vec{x},t) - \frac{1}{\tau}\left[f_i(\vec{x},t) - f_i^{eq}(\rho(\vec{x},t),\vec{u}(\vec{x},t))\right], \tag{1}$$

$$f_i^{eq}(\rho(\vec{x},t),\vec{u}(\vec{x},t)) = w_i\left[\rho + \rho_0\left(3\vec{e}_i\vec{u} + 4.5(\vec{e}_i\vec{u})^2 - 1.5\vec{u}^2\right)\right] \quad i = 0\ldots 19 \tag{2}$$

[1] http://www.walberla.net.

and can be split into two steps: A collision step applying the collision operator and a propagation step advecting the PDFs to the neighboring cells. In this article, we use the single relaxation time collision operator [16]. In Eq. (1), f_i^{coll} denotes the intermediate state after collision but before propagation. The relaxation time τ can be determined from the kinematic viscosity $\nu = (\tau - \frac{1}{2})c_s^2\delta t$, where c_s is the speed of sound. Further, f_i^{eq} results from a Taylor expansion of the Maxwell–Boltzmann equilibrium distribution function [16] optimized for incompressible flows [17]. For the isothermal case, f_i^{eq} depends on the macroscopic velocity $\vec{u}(\vec{x},t)$ and the macroscopic density $\rho(\vec{x},t)$, and the lattice weights w_i that depend on the direction are either $\frac{1}{3}$, $\frac{1}{18}$, or $\frac{1}{36}$. The macroscopic quantities ρ and $\vec{u}$ are determined from the 0^{th} and 1^{st} order moment of the distribution functions $\rho(\vec{x},t) = \rho_0 + \delta\rho(\vec{x},t) = \sum_{i=0}^{18} f_i(\vec{x},t)$, and $\rho_0\vec{u}(\vec{x},t) = \sum_{i=0}^{18}\vec{e}_i f_i(\vec{x},t)$, where ρ is split into a constant part ρ_0 and a slightly changing perturbation $\delta\rho$. The equation of state of an ideal gas provides the pressure $p(\vec{x},t) = c_s^2\rho(\vec{x},t)$.

The PDFs are initialized to $f_i^{eq}(\rho_0, 0)$. To increase the accuracy of simulations done in single floating point precision we use $\tilde{f}_i(\vec{x},t) = f_i(\vec{x},t) - f_i^{eq}(\rho_0,0)$ and $\tilde{f}_i^{eq}(\rho(\vec{x},t),\vec{u}(\vec{x},t)) = f_i^{eq}(\rho(\vec{x},t),\vec{u}(\vec{x},t)) - f_i^{eq}(\rho_0,0)$ as proposed by [17] resulting in PDF values centered around 0. According to [18], it is possible with the LBM scheme described above to achieve accurate single precision results.

For the implementation of the propagation, there are two options: pushing or pulling. In the first case, the PDFs in a cell are first collided and then pushed to the neighborhood. In the second case, the neighboring PDFs are first pulled into the lattice cell and then collided. In WaLBerla, we use the *pull* approach, since it is better suited for parallelization.

Additionally, the propagation step introduces data dependencies to LBM, which commonly result in an implementation of the LBM using two PDF grids. However, these dependencies are local, as only PDFs of neighboring cells are accessed, and therefore there exist also single grid implementations in literature [19].

The simplest approach for implementing solid wall boundaries in the LBM is the bounce-back (BB) rule [15,16], i.e. if a distribution is about to be propagated into a solid cell, its direction is reversed into the original cell. BB generally assumes that the wall is in the middle between the two cell centers, i.e. half-way. This formulation leads to:

$$f_{\bar{i}}(\vec{x}, t + \Delta t) = f_i(\vec{x},t) + 6w_i\rho_0\vec{e}_i\vec{u}_w,$$

with $\vec{e}_{\bar{i}} = -\vec{e}_i$ and u_w being the velocity prescribed at the wall.

3. Software design for parallel simulations

For the development of HPC applications, it is important that the applied software engineering considers performance as a design objective right from the start for scalable heterogeneous simulations on both CPUs and GPUs. Our approach attempts to provide a good balance between efficiency and maintainability. Essentially, this is achieved by the management of kernels. A kernel is a problem- and hardware-specific implementation of one work step of a simulation task, e.g. a LB kernel that updates all unknowns including the treatment of all BCs at a certain time step. A kernel is usually represented by a callable object, e.g a C-function or a CUDA kernel. In parallel simulations kernels usually operate on a subdomain of the computational domain. The design concepts have been implemented in the waLBerla software framework and are not limited to the LB method [20,10].

3.1. Kernel management

For the description and management of complex simulation tasks, each task is broken down into work steps, so called *Sweeps*. The actual work step in a *Sweep* is represented by one or several kernels, which are connected to meta data. Each kernel implements one flavor of the work step that can be selected at runtime with the help of the meta data. Examples for work step flavors for the LB kernel are, e.g. a CPU or a GPU implementation. Here, the meta data describes the different heterogeneous compute resources. Further, the sweep concept allows the execution of pre- and post-processing operations for each work step, e.g. communication of data or visualization. The concepts introduce no significant performance overhead. Operations in performance critical sections, e.g. operations involving the update of the unknowns, can be implemented as efficient as possible. The data fields are directly exposed to the kernels and no abstractions and encapsulations are applied. Further, several implementations provide optimizations for specific hardware architectures, e.g. by using different programming models on CPUs and GPUs.

3.2. Multi-GPU parallelization

In Fig. 1 a schematic overview of the multi-GPU design for 3D distributed memory parallelizations is given. This concept is used for all parallelization approaches introduced below. Multi-GPU simulations using several compute nodes involve three steps. First the boundary data is extracted from the GPU data structures and copied into buffers allocated on the GPU. This is required to provide an optimal data layout for the second step, in which the data is moved from device to host via PCI-E (peripheral component interconnect express) transfers. The data is copied into buffers allocated on the host, which have to be page-locked and registered to CUDA to optimize the bandwidth. The host buffers are also used for third step, the MPI communication. For plain LBM simulations only PDFs have to be transferred, but for more complex simulation tasks

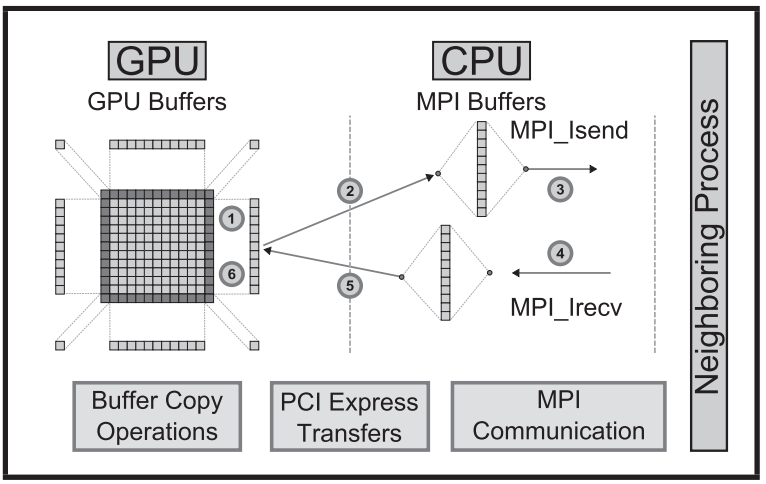


Fig. 1. 3D multi-GPU parallelization involving three communication steps. Numbers indicate the sequence of the work flow.

the communication of additional data is likely to be required. Our concept supports the bundling of messages of arbitrary data types and next neighbor communications to an arbitrary number of communication partners. Further details can be found in [10].

3.3. Overlapping of work and communication

The minimization of the communication overhead is of higher importance for GPUs than for CPUs, as the time spent communicating in comparison to the kernel runtime is larger. To further minimize the communication overhead we have overlapped work and communication, which ideally hides communication completely behind calculation. Therefore, it is necessary to split the LB kernel into two parts: an outer and an inner kernel. The outer kernel only updates the boundary data required for the communication, so that the inner kernel and the communication have no data dependencies and can be executed in parallel. Fig. 2 shows the work flow for the overlapping of work and communication. For the parallel execution of kernels CUDA provides so called *streams*. To each *stream* a sequence of kernels are assigned, which are executed in order. Different streams, however, can be executed in parallel.

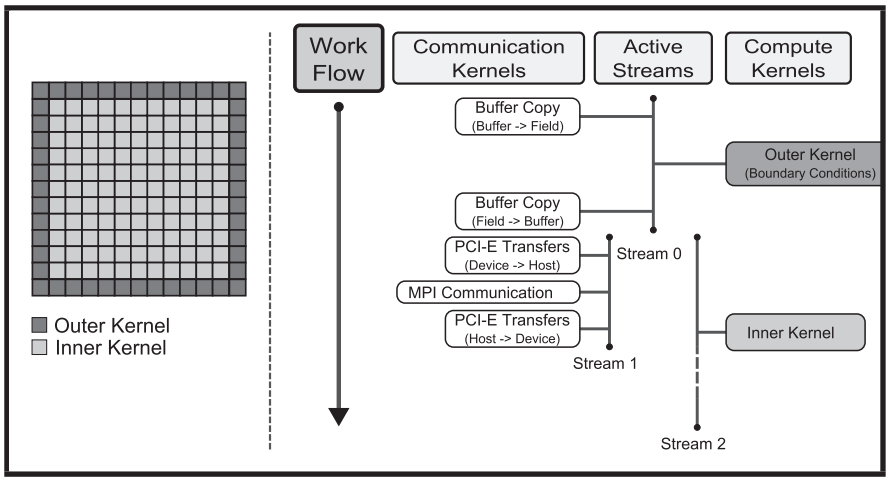


Fig. 2. Overlapping of work and communication. The kernels for the buffer copy operations and the outer kernel including the BC treatment for the entire simulation domain are executed in *stream 0*. After completion, the inner kernel and the communication kernels are executed in parallel in the two separate streams 0 and 1.

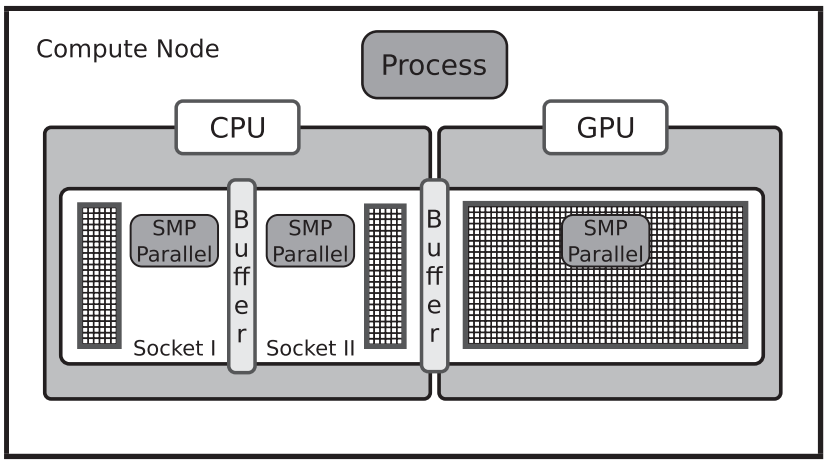


Fig. 3. Domain decomposition on one compute node using the hybrid parallelization approach and one GPU and two CPU sockets. Static load balancing is applied in z-dimension based on the compute capacities of the GPU and the CPU sockets. Communication data is exchanged between the compute resources via buffers.

3.4. Parallelization paradigms

For this work two different parallelization paradigms have been evaluated, a pure-MPI and a hybrid parallelization approach. Using the pure-MPI parallelization approach one process per GPU is executed and all GPUs process an equally sized part of the simulation domain. In contrast, for the hybrid parallelization approach one process per compute node is used and in parallel simulations all compute nodes have an equally sized part of the simulation domain. The subdomain on each compute node can be freely distributed along the z-dimension amongst the hardware components of the compute node. This allows static load balancing for heterogeneous architectures with diverging performance characteristics. Investigations [14,10] have shown that for heterogeneous simulations using a static load balancing based on the distribution of equally sized subregions can not deliver satisfactory performance. An example for the final domain decomposition on a compute node having one GPU and two CPU sockets is depicted in Fig. 3. The intra node parallelization of the hybrid parallelization approach is based on a tasking model implemented via *boost::threads* [21]. A CPU thread pool is assigned to each compute component, i.e. a GPU or a full CPU socket, within the compute node. These thread pools process the work tasks. For the GPU a thread pool with only one thread is assigned. This thread schedules the actual workload to the GPU. The remaining threads are distributed amongst the thread pools of the CPU sockets and process the actual workload of the CPU.

The design for the inter-node communication remains nearly unchanged for the hybrid parallelization approach. The only difference is that the data for the MPI buffers is collected from several subregions instead of one as for the pure-MPI approach and that threads of the thread pools have to be synchronized. This is necessary after the LB kernel and twice for copy operations involving the MPI buffers. For heterogeneous simulations architecture specific copy kernels have to be selected. This is supported by the waLBerla kernel management on the basis of the kernels' meta data. Thus, the communication buffers act as an interface between the different heterogeneous architectures. In the following, simulations using CPUs and GPUs are referenced to utilize a heterogeneous parallelization approach and simulations using only CPUs or GPUs are referenced as hybrid parallelization approach.

3.5. Challenges for heterogeneous simulations

The presented parallelization design can solve the challenges for efficient and scalable heterogeneous simulations.

Challenge	Solution
Description of the heterogeneous compute resources	Description of all compute components per compute node in the input file
Management of the communication and compute kernels for each architecture	Kernel management based on meta data
Common communication interface	Data exchange via communication buffers also for intra node communication
Minimization of the heterogeneous communication overhead	Overlapping of work and communication, non-uniform domain decomposition, and intra-node communication in z-dimension

4. Technical details

All simulations for this article have been carried out on the Tsubame 2.0 cluster (Table 1) located at Tokyo Institute of Technology in Japan. The simulated scenarios are a lid driven cavity and a flow through a porous medium. Further, the Intel complier (version 11.1), OpenMPI (version 1.4.2), and CUDA (version 4.0) have been used. All simulations have been executed in DP (double precision) and ECC (error–correction code) has been turned on. For weak and strong scaling scenarios the compute node numbers have been increased for all dimensions equally, e.g nodes numbers ranging from [1,8,27,...,343] have been used. Within a node the domain decomposition has been one-dimensional in z-dimension. For strong scaling experiments, the domain size has been chosen such that the required memory for the simulation can be allocated in the GPU main memory of a single compute node.

5. Modeling the performance of parallel LBM simulations

Fundamentals on performance optimization and modeling can be found in [22,23]. Schönauer et al. [24] introduces established the bandwidth obtained by the so called Schönauer triad for the evaluation of the "lightspeed" performance of supercomputers for memory bound implementations. The STREAM benchmarks [25] used in this work are essentially variants of the Schönauer triad with additional streaming kernels.

The goal of our performance model is the precise estimation of the communication overhead to investigate and predict the performance of parallel simulations in scaling experiments. Our performance model is split into two parts: the investigation of the compute node performance and of the performance of a multi-node parallel implementation.

The performance model for a single compute node has been discussed already in[14,6]. Thus, we will here only summarize it briefly and apply it to derive estimates for a Tsubame compute node. In SubSection 5.2, a new communication model will be introduced that allows us to estimate the communication overhead for parallel simulations.

The kernel performance $P_k = \frac{max(t_k^i)}{n_c}$ on one compute node is analyzed by upper bound "lightspeed" estimates. It is determined by the maximal time t_k^i spent on any process i in the LB kernel not including the communication overhead and the overall number of lattice cells n_c. This is of importance, as the report of scalability results based on unoptimized LB kernel implementations is pointless. Unoptimized implementations show better scalability, despite a higher time to solution. For the analysis of the parallel performance we use the measured kernel performance, instead of the "lightspeed" estimate, combined with predictions for the communication overhead to obtain precise estimates for the parallel performance, since the kernel performance has already been investigated on a per compute node level and no further insight will be gained by using the "lightspeed" estimate in parallel simulations. However, it is possible to precisely predict the parallel performance with the help of the measured kernel performance, as it accounts for the dependence of the performance on the domain or messages sizes. This is not the case for upper bound "lightspeed" estimates.

5.1. Upper bound for the kernel performance

For sufficiently large domain sizes implementations of the LBM using the SRT collision operator are usually memory bound. This can be seen by the machine balance B_m (Eq. (3)) and the code balance B_c (Eq. (4)) as proposed by the balance model [22]. B_m describes the ratio of sustained machine bandwidth b_s in [bytes] to the peak performance p_{max} in [FLOPS], whereas B_c provides the ratio of the number n_b of bytes loaded and stored for the execution of the algorithm and the number n_f of executed FLOPS.

On CPU architectures, the sustained memory bandwidth b_s can be estimated with the STREAM copy benchmark [25]. This synthetic benchmark implements a vector copy $a_i = b_i$, which is similar to the memory access pattern of the LB stream step. On GPUs an in-house implementation [6] of the benchmark has been used. For the LB method $n_b = 304$ bytes and approximately $n_f = 200$ FLOPS [26] are required for the update of one lattice cell.

$$B_m = \frac{b_s}{p_{max}}, \tag{3}$$

$$B_c = \frac{n_b}{n_f}, \tag{4}$$

$$l = min\left(1, \frac{B_m}{B_c}\right). \tag{5}$$

Table 1
Specifications of the Tsubame 2.0 cluster.

Nodes	CPU	GPU	GPUs	LINPACK	Power consumption	Interconnect
1442	Intel Xeon X5670	NVIDIA Tesla M2050	3 per Node	1192 [TFLOPS]	1399 [KW]	QDR InfiniBand

Table 2
Sustainable memory bandwidth and LBM kernel performance estimation. Bandwidth results have been obtained from [27,6] for an NVIDIA Tesla M2070 "Fermi" and a dual-socket Intel Xeon X5670 "Westmere". For the measured performance results BC handling has not been taken into account. Results are in DP and ECC has been turned on for all measurements.

Architecture	Theoretical DP FLOPS [TFLOPS]	Theoretical bandwidth [GB/s]	Sustainable bandwidth [GB/s]	B_m	B_c	l	LBM estimate [MFLUPS]	LBM estimate [MFLUPS]
"Fermi"	0.52	144	95	0.18	1.52	0.12	312	256
"Westmere"	0.12	64	40.4	0.34	1.52	0.22	133	105

If the "lightspeed" balance l in Eq. (5) is $l < 1$, a code is bandwidth limited. For the LBM method Table 2 indicates a memory bandwidth limitation, with a higher imbalance for GPUs than for CPUs. Hence, the LB kernel performance can be estimated by

$$P_e = \frac{b_s}{n_b}. \tag{6}$$

Comparing the measured performance with the estimated one shows that both implementations are efficient. The GPU implementation achieves around 82% and the CPU implementation 79% of the "lightspeed" estimate.

5.2. Communication model

The time to solution t_s for multi-GPU simulations can be determined by

$$t_s = t_k + t_b + t_{pci} + t_{mpi}, \tag{7}$$

in case of non-overlapping and

$$t_s = t_{k,outer} + \max(t_{k,inner}, t_b + t_{pci} + t_{mpi}), \tag{8}$$

in case of overlapping communication and computation. Eqs. (7) and (8) contain the LB kernel runtime t_k or $t_{k,inner} + t_{k,outer}$, t_b represents the time for the buffer copy operations, t_{pci} the time for the PCI-E transfers, and t_{mpi} the time spent in the MPI communication.

To estimate the communication overhead the time spent to transfer a message i has to be determined dependent on two quantities: the size of the message s_i and the bandwidth b_i with which the message is transferred.

$$t_\omega^i = \frac{s_i}{b_i(s_i, x_i, p_i, \omega)}, \tag{9}$$

where the bandwidth is dependent on s_i, the number of messages x_i that are concurrently transferred over the communication link, the relative position of the communication partners p_i, e.g. intra- or inter-node communication, and the communication link ω, e.g. if is an MPI or PCI-E transfer. Hence, the aggregated communication times t_{pci} und t_{mpi} given in Eqs. (7) and (8) can be determined by

$$t_{pci} = \sum_{i=0}^{N} t_{pci}^i, \quad t_{mpi} = \sum_{i=0}^{M} t_{mpi}^i, \tag{10}$$

where the number of messages N and M as well as s_i depend the parallelization approach, the number of computes nodes, and the simulation domain. For the estimation of the time to solution t_s, t_k is assumed to be identical on all processes, and t_b, t_{PCI}, and t_{MPI} are determined for the process having the highest communication overhead. In order to provide accurate estimates for parallel simulations, we measured t_k and t_b for all required domain sizes, which can easily be done on a single compute node and be used for all subsequent parallel simulations. Hence, the proposed performance model is based on an estimate for the communication overhead and the measured kernel performance, which has been validated by an upper bound performance model.

5.3. Communication overhead for parallel simulations

In Fig. 4 the communication overhead for parallel multi-GPU LB simulations is depicted for weak and strong scaling scenarios. The results show that it is possible to precisely predict the communication overhead with our communication model. The deviation between measurement and estimation in the strong scaling scenarios for the hybrid parallelization approach can be explained by the additional thread synchronization overhead that is not covered by the communication model.

5.3.1. Weak scaling

For the hybrid and heterogeneous weak scaling experiments the estimated as well as the measured communication overhead is higher compared to the pure-MPI parallelization approach. With the help of the input parameters for the commu-

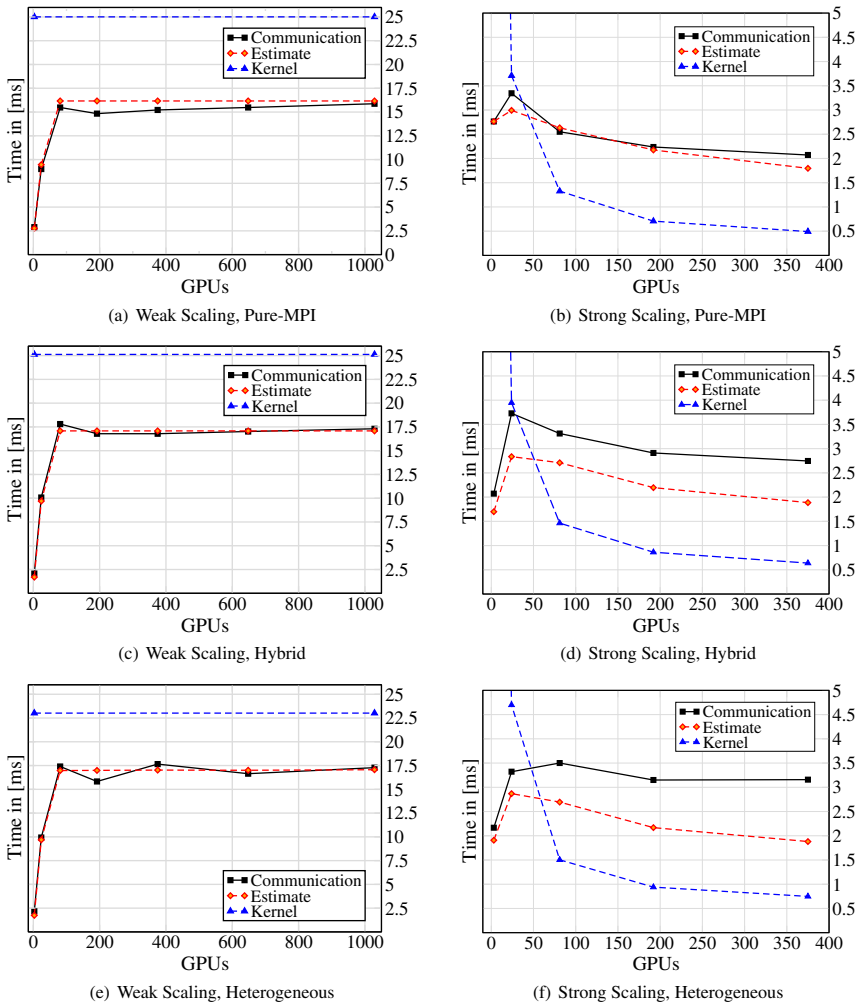


Fig. 4. Communication overhead of weak and strong scaling experiments using the pure-MPI, hybrid, and the heterogeneous parallelization approach using a domain size of 180^3 lattice cells per GPU in weak scaling experiments and a overall domain size of $180^3 * 3$ lattice cells for strong scaling experiments. Results have been obtained on Tsubame 2.0 in DP and with ECC turned on.

nication model the reason therefore can be identified: a higher InfiniBand bandwidth can be obtained using 3 processes per compute node (pure-MPI approach) than for one process (hybrid and heterogeneous approach). Further, the communication overhead for the hybrid and the heterogeneous parallelization approach are comparable. In summary, the communication overhead in the weak scaling experiments has an significant influence on the overall performance, but can be hidden behind the kernel runtime by overlapping work and communication.

5.3.2. Strong scaling

The pure-MPI parallelization approach achieves the lowest communication overhead in strong scaling scenarios, as there is no thread synchronization overhead like for the hybrid parallelization. The thread synchronization overhead is independent of the domain size. Thus, it is not significant for large domain sizes used in the weak scaling experiments, but it is for the small domain sizes occurring in strong scaling experiments. The thread synchronization overhead for the heterogeneous parallelization approach is highest, as it uses more threads per MPI process.

The estimated communication overhead for all parallelization approaches is similar. First, this is the case, because the thread synchronization overhead is neglected. Second, although the bandwidth obtained for the pure-MPI approach is increased by using three processes compared to one per compute node, it is also decreased due to the smaller message sizes. For the weak scaling experiments this had no effect, as there even for smaller messages of the pure-MPI parallelization approach resulted in a saturation of the bandwidth. In contrast to the weak scaling experiments, the communication overhead in strong scaling experiments can only be hidden partially.

6. Performance results

In this Section the single compute node and parallel performance of pure-LBM simulations is investigated for CPUs and GPUs.

6.1. Single compute node performance

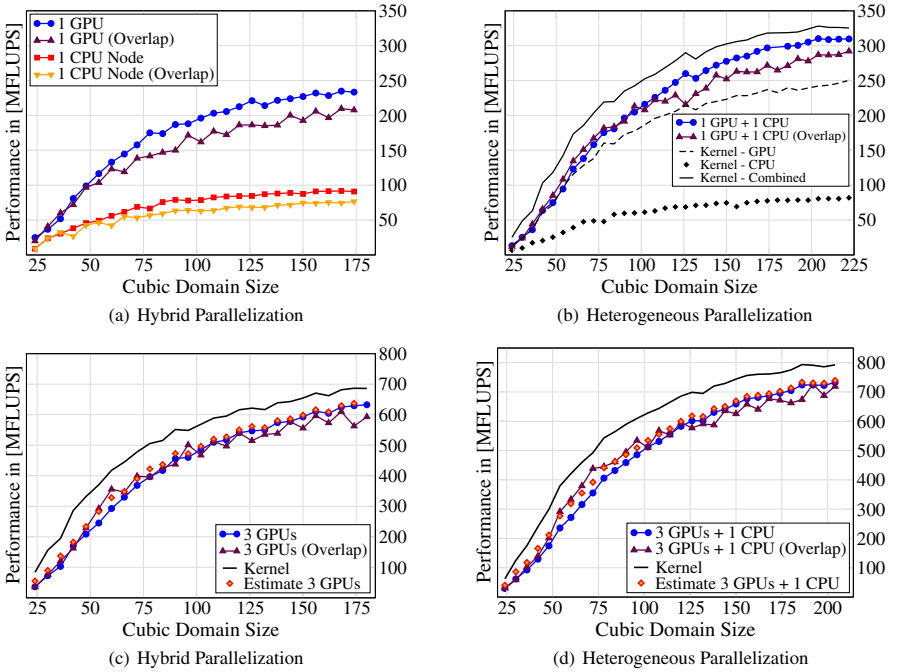
In Fig. 5 the LB performance results obtained on a single compute node are depicted. Only the results for the hybrid and heterogeneous parallelization approach are presented, as the results of the pure-MPI parallelization approach are similar to the hybrid parallelization approach. Fig. 5(a) shows the performance for a single GPU and a single CPU node without and with overlapping of work and communication. The CPU implementation achieves a maximal performance of 93 MFLUPS, while the GPU implementation achieves a maximum of 234 MFLUPS. The speed up of around 2.5 has been expected for bandwidth limited implementations, as the utilized GPU offers a around 2.4 times higher sustained bandwidth than the utilized CPU compute node (Table 2). Using the heterogeneous parallelization approach and one GPU plus one CPU compute node, a speed up of up to 28% is achieved over simulations only using GPUs (Fig. 5(b)). A maximum of 95% of the kernel performance, i.e. CPU + GPU kernel performance, is achieved with the heterogeneous parallelization approach.

Weak Scaling to three GPUs on one compute node shows a deviation of the measured performance from the kernel performance of around 10%. A maximum of 640 MFLUPS is achieved using 3 GPUs. In this scenario the heterogeneous approach achieves a 10% speed up over the simulations only using GPUs (Fig. 5(d)). Both results are accurately predicted by the performance model, which shows that there are no significant hidden overheads in the implementation of the communication. The overlapping of communication and work results in a degradation of the kernel performance, as can be seen in Fig. 5(a). This results from the overhead of splitting the LB kernel into an outer and an inner kernel. Using one compute node there is no need to overlap work and communication (Fig. 5(c) and (d)).

6.2. Parallel performance

The objective of this Section is to investigate the scalability of the LB method on GPUs using different parallelization approaches in weak and strong scaling scenarios for varying domain sizes.

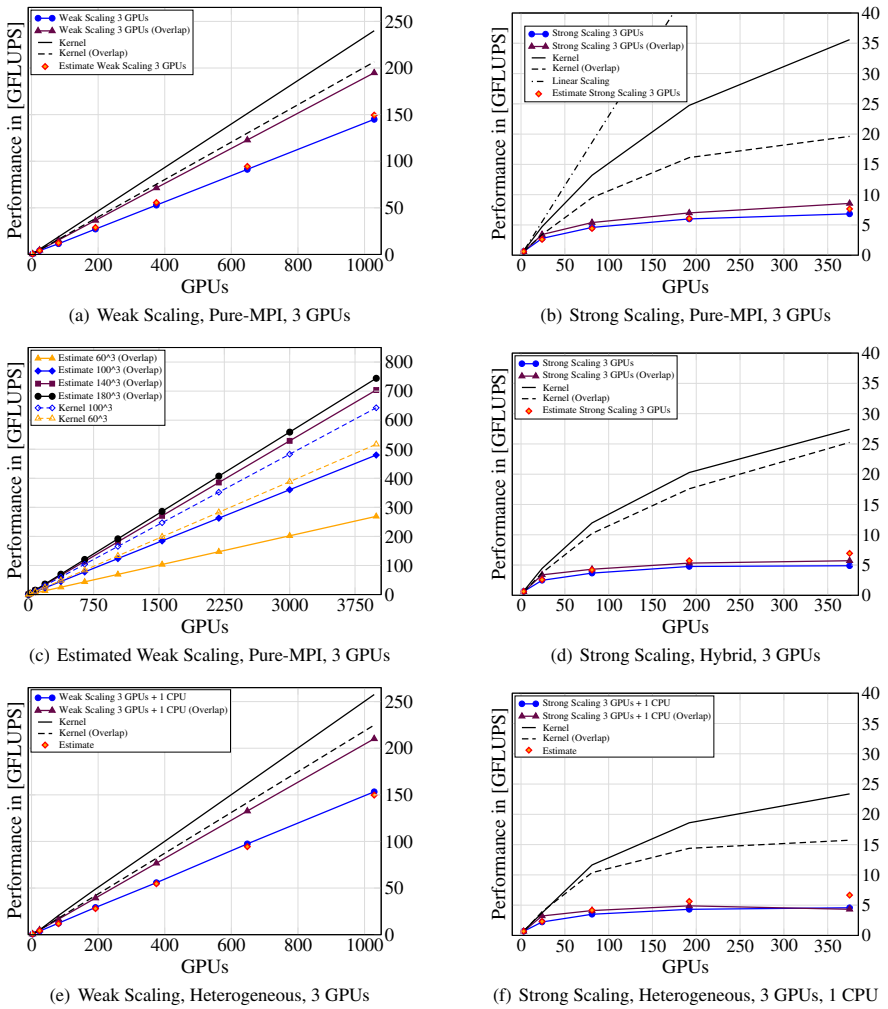
Weak Scaling. The investigation of weak scaling experiments in Fig. 6(a), (c) and (e) shows that for multi-GPU LB simulations the communication overhead is significant, i.e. the difference between linearly scaled single GPU performance representing ideal scaling and performance measurements. For the domain sizes of 180^3 lattice cells per GPU this overhead can be hidden behind work to a large extend. Fig. 2 shows that the buffer copies on the GPUs are not hidden. The estimations for different domain sizes depicted in Fig. 6(c) shows that for domain sizes of 100^3 or 60^3 lattice cells this is no longer pos-



(a) Hybrid Parallelization (b) Heterogeneous Parallelization

(c) Hybrid Parallelization (d) Heterogeneous Parallelization

Fig. 5. Single compute node performance for the hybrid and heterogeneous parallelization approach.



(a) Weak Scaling, Pure-MPI, 3 GPUs

(b) Strong Scaling, Pure-MPI, 3 GPUs

(c) Estimated Weak Scaling, Pure-MPI, 3 GPUs

(d) Strong Scaling, Hybrid, 3 GPUs

(e) Weak Scaling, Heterogeneous, 3 GPUs

(f) Strong Scaling, Heterogeneous, 3 GPUs, 1 CPU

Fig. 6. Weak and strong scaling performance on Tsubame 2.0 in DP and with ECC. The domain size for the weak scaling experiments has been 180^3 lattice cells per GPU in weak scaling experiments and for the strong scaling experiments an overall domain size of $180^3 * 3$ lattice cells has been used. For the weak scaling experiments up to 1029 GPUs have been used and for the strong scaling experiments a maximum of 375 GPUs have been used.

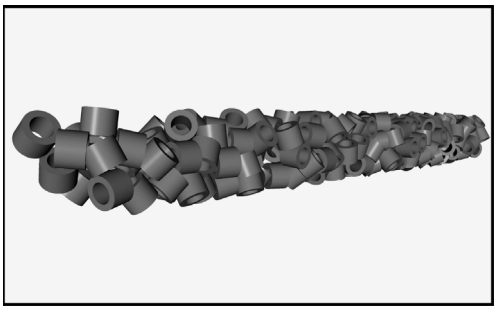
sible. Additionally, the predictions for the domain size of 180^3 lattice cells estimate a maximal performance of 745 GFLUPS for the full Tsubame 2.0. This corresponds to around 60% of the total sustainable bandwidth of the cluster. The best results are achieved with the heterogeneous parallelization approach (Fig. 6(e)). Here, it is possible to preserve the 10% speed up obtained for the single compute node experiments.

Strong Scaling. In the strong scaling experiments the pure-MPI parallelization approach achieves the best results (Fig. 6(b), (d) and (f)). For an increasing number of GPUs the strong scaling performance is hampered by two mayor effects. First, the small domain sizes occurring in strong scaling scenarios decrease the kernel performance. This can be seen from the deviation of linear scaling and kernel performance in Fig. 6(b). Second, the difference between the measured and kernel performance shows the reduction of the performance by the communication overhead.

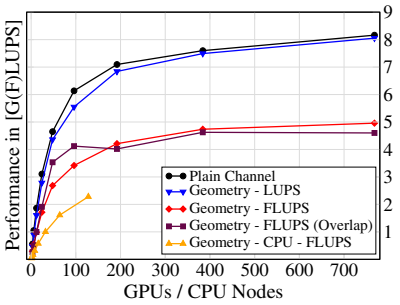
The use of the hybrid and the heterogeneous parallelization approach results in lower performance due to the thread synchronization overhead, which decreases the kernel performance and increases the communication overhead. In addition, the kernel performance of heterogeneous parallelization is further reduced compared to the hybrid approach, as here the domain sizes simulated on the GPUs are even smaller than for the hybrid strong scaling scenarios.

6.3. Parallel performance for flows through a porous media

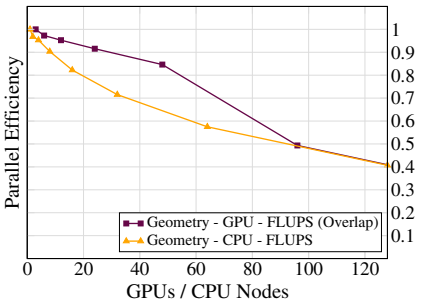
As show case for the applicability of the presented implementation the flow through a packed bed of hollow cylinders (see Fig. 7(a)) is presented in this Section. The purpose of this investigation is to carry out simulations to optimize the structural composition of the granular medium to minimize the pressure drop, to maximize the surface of the catalytically active



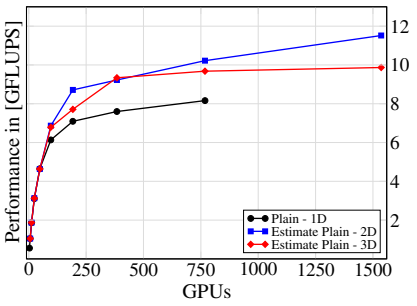
(a) Packed bed of hollow cylinders.



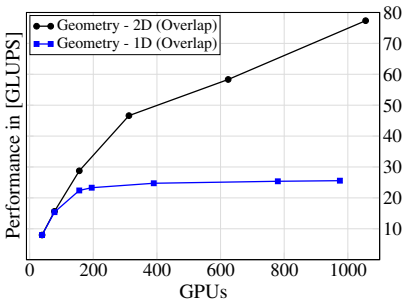
(b) Pure-MPI, 1D, $100^2 \times 1536$



(c) Pure-MPI, 1D, $100^2 \times 1536$



(d) Pure-MPI, 1D + 2D + 3D, $100^2 \times 1536$



(e) Pure-MPI, 1D + 2D, $256^2 \times 3600$

Fig. 7. Strong scaling performance of porous media simulations using the pure-MPI parallelization for two domain sizes: $100 \times 100 \times 1536$ and $256 \times 256 \times 3600$ lattice cells. Performance results for CPUs and GPUs are depicted for 1D, 2D, and 3D domain decomposition. Additionally, the parallel efficiency for CPUs and GPUs for the coarser resolution is given.

granular media, and to achieve an optimal residence time of the fluid. As for the determination of the optimal layout many simulations are required, the time to solution of one simulation is highly relevant. Hence, the performance that can be achieved with the LB method at maximum, i.e. for the simplest LB collision model available is evaluated. In Fig. 7(b) the performance for a coarse discretization of the geometry with $100 \times 100 \times 1536$ lattice cells, and a one-dimensional domain decomposition is depicted. The pure-MPI parallelization approach including overlap has been used, as it has been the fastest implementation for strong scaling scenarios. In total, the geometry included around 9.2 M fluid cells corresponding to a volume fraction of 60% fluid. A maximum of around 8 GFLUPS is achieved for the simulation of the plain channel containing no obstacles, which corresponds to 520 time steps per second. The simulations including the geometry achieved a maximum of 8 GLUPS (giga lattice updates per second). This shows that the additionally required BC treatment has no major impact on performance. However, as independent of their BC state all lattice cells are streamed and collided, the resulting maximal number of updated fluid cells is reduced to 5 GFLUPS. The CPU implementation achieved a maximum of 2.3 GFLUPS on 128 compute nodes. A comparison of the parallel efficiency of the CPU and the GPU implementation is depicted in Fig. 7(c). It can be seen that the GPU implementation with the help overlapping can obtain a higher parallel efficiency up to 48 GPUs. To investigate the performance for two- and three-dimensional domain decompositions, estimates predicted by the performance model for the communication are depicted in Fig. 7(d). It can be seen that better scaling can be obtained, but also that a three-dimensional domain decomposition does not necessarily achieve higher performance. For the two-dimensional domain decomposition the domain size in x-dimension is not reduced during the strong scaling, which results in higher kernel performance compared to the three-dimensional decomposition. The performance for a finer resolution of $256 \times 256 \times 3600$ lattice cells is depicted in Fig. 7(e).

7. Conclusions

We have introduced software design concepts for multi-GPU and heterogeneous simulations on CPUs and GPUs. Further, we have reported our solutions for the challenges of efficient heterogeneous LB simulations. With the help of our performance model it was possible to validate the single compute node performance and to precisely predict the performance of parallel simulations. The reported performance analysis shows that our software design and implementation is efficient and scales well on Tsubame 2.0 up to more than 1000 GPUs. In addition, we were able to successfully predict the performance of an industrial relevant geometry representing a porous structure.

Summarizing, the performance results on Tsubame 2.0 cluster show that

- for weak scaling experiments good scalability can be achieved for multi-GPU simulations,
- for strong scaling experiments not only the communication overhead has to be minimized, but also the kernel performance for small domain sizes is important,
- the communication overhead is significant for GPUs,
- the heterogeneous parallelization approach can achieve 28% higher performance using one GPU and one CPU compute node than GPU-only simulations,
- for simulations on a single compute node and for weak scaling experiments the performance of the heterogeneous parallelization approach is fastest compared to the pure-MPI and hybrid parallelization approach,
- in strong scaling experiments the pure-MPI parallelization approach is fastest, as the hybrid and heterogeneous parallelization approach have the additional overhead for thread synchronization.

In the future clusters are expected to become more and more heterogeneous such that our design concepts are already suitable for the next generation of supercomputers. One of the next major steps within waLBerla will be the support of local refinement strategies that will require dynamic load balancing for the parallel simulations.

Acknowledgments

This work is partially funded by the Bundesministerium für Bildung und Forschung under the *SKALB* project, No. 01IH08003A, as well as by the "Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern" (*KONWIHR*) via *waLBerlaMC*.

References

[1] Information on the Tsubame 2.0, <http://www.gsic.titech.ac.jp/en/tsubame2>.
[2] T. Zeiser, G. Hager, G. Wellein, Benchmark analysis and application results for lattice Boltzmann simulations on NEC SX vector and intel Nehalem systems, Parallel Process. Lett. 19 (4) (2009) 491–511.
[3] C. Aidun, J. Clausen, Lattice–Boltzmann method for complex flows, Annu. Rev. Fluid Mech. 42 (1) (2010) 439–472.
[4] C. Feichtinger, J. Götz, S. Donath, K. Iglberger, U. Rüde, WaLBerla: exploiting massively parallel systems for lattice Boltzmann simulations, in: Parallel Computing. Numerics, Applications, and Trends, Springer, 2009, pp. 240–259.
[5] J. Tölke, M. Krafczyk, Teraflop computing on a desktop PC with GPUs for 3D CFD, Int. J. Comput. Fluid Dyn. 22 (7) (2008) 443–456.
[6] J. Habich, C. Feichtinger, H. Köstler, G. Hager, G. Wellein, Performance engineering for the lattice Boltzmann method on GPGPUs: architectural requirements and performance results, Comput. Fluids 80 (2013) 276–282.
[7] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, A new approach to the lattice Boltzmann method for graphics processing units, Comput. Math. Appl. 61 (12) (2011) 3628–3638.
[8] X. Wang, T. Aoki, Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster, Parallel Comput. 37 (9) (2011) 521–535.
[9] Q. Xiong, B. Li, J. Xu, X. Fang, X. Wang, L. Wang, X. He, W. Ge, Efficient parallel implementation of the lattice Boltzmann method on large clusters of graphic processing units, Chin. Sci. Bull. 57 (2012) 707–715.
[10] C. Feichtinger, Design and performance evaluation of a software framework for multi-physics simulations on heterogeneous supercomputers (Ph.D. thesis), 2012.
[11] S. Donath, K. Mecke, S. Rabha, V. Buwa, U. Rüde, Verification of surface tension in the parallel free surface lattice Boltzmann method in walberla, Comput. Fluids 45 (1) (2011) 177–186.
[12] J. Götz, K. Iglberger, M. Stürmer, U. Rüde, Direct numerical simulation of particulate flows on 294912 processor cores, in: ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10), IEEE Computer Society, New Orleans, Louisiana, USA, 2010, pp. 1–11.
[13] K. Pickl, J. Götz, K. Iglberger, J. Pande, K. Mecke, A.-S. Smith, U. Rüde, All good things come in threesthree beads learn to swim with lattice Boltzmann and a rigid body solver, J. Comput. Sci. 3 (5) (2012) 374–387.
[14] C. Feichtinger, J. Habich, H. Köstler, G. Hager, U. Rüde, G. Wellein, A flexible patch-based lattice Boltzmann parallelization approach for heterogeneous GPU–CPU clusters, Parallel Comput. 37 (9) (2011) 536–549.
[15] S. Chen, G.D. Doolen, Lattice Boltzmann method for fluid flows, Annu. Rev. Fluid Mech. 30 (1) (1998) 329–364.
[16] S. Succi, The lattice Boltzmann Equation for Fluid Dynamics and Beyond (Numerical Mathematics and Scientific Computation), Oxford University Press, USA, 2001.
[17] X. He, L.-S. Luo, Lattice Boltzmann model for the incompressible Navier Stokes equation, Stat. Phys. 88 (3–4) (1997) 927–944.
[18] J. Tölke, M. Krafczyk, Teraflop computing on a desktop PC with GPUs for 3D CFD, Int. J. Comput. Fluid Dyn. 22 (7) (2008) 443–456.
[19] M. Wittmann, T. Zeiser, G. Hager, G. Wellein, Comparison of different propagation steps for lattice boltzmann methods, Comput. Math. Appl. 65 (6) (2013) 924–935.
[20] C. Feichtinger, S. Donath, H. Köstler, J. Götz, U. Rüde, WaLBerla: HPC software design for computational engineering simulations, J. Comput. Sci. 2 (2) (2011) 105–112.
[21] Information on the C++ Library Boost, <http://www.boost.org/>.

[22] G. Hager, G. Wellein, Introduction to High Performance Computing for Scientists and Engineers, CRC Press Inc, 2010.
[23] S. Goedecker, A. Hoisie, Performance optimization of numerically intensive codes, in: SIAM Conference on Computational Science and Engineering, Philadelphia, USA, 2001.
[24] W. Schönauer, Scientific supercomputing: architecture and use of shared and distributed memory parallel computers, Karlsruhe, 2000.
[25] J. McCalpin, Memory bandwidth and machine balance in current high performance computers, IEEE Computer Society Technical Committee on Computer Architecture Newsletter, 1995, pp. 19–25.
[26] G. Wellein, T. Zeiser, G. Hager, S. Donath, On the single processor performance of simple lattice Boltzmann kernels, Comput. Fluids 5 (8–9) (2006) 910–919.
[27] G. Wellein, J. Habich, G. Hager, T. Zeiser, Node-level performance of the lattice Boltzmann method on recent multicore CPUs, Extended abstract ParCFD, 2011, <http://parcfd2011.bsc.es/sites/default/files/abstracts/id141-wellein.pdf>.