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

Lehrstuhl für Informatik 10 (Systemsimulation)

waLBerla: Exploiting Massively Parallel Systems for CFD

C. Feichtinger, J. Götz, S. Donath, K. Iglberger, U. Rüde

Technical Report 08-2

waLBerla: Exploiting Massively Parallel Systems for CFD

C. Feichtinger, J. Götz, S. Donath, K. Iglberger, U. Rüde

Lehrstuhl für Systemsimulation
Friedrich-Alexander University of Erlangen-Nuremberg
91058 Erlangen, Germany

{christian.feichtinger, jan.goetz, stefan.donath, klaus.iglberger, ruede}@informatik.uni-erlangen.de

February 14, 2008

The waLBerla project aims at the development of an efficient massively parallel lattice Boltzmann software library providing the necessary features for several computational fluid dynamics applications. In this article we focus on our parallelization of the framework, which is based on a domain partitioning scheme named patch concept. This concept also enables a seamless specialization of the partitions to the different application features as well as the possibility for further optimization such as memory reduction. It is discussed in detail how our design of the patches ensures an efficient and flexible implementation. Furthermore, the communication of the framework is introduced, which includes process-local and global communication. In order to discuss the suitability of the parallelization for massively parallel usage, various test scenarios have been investigated on different architectures. These tests include serial, weak and strong scaling experiments up to 810 cores and up to a domain size of 1530^3 lattice cells.

1 Motivation

In computational fluid dynamics (CFD) many applications of scientific interest share physical and computational aspects. In research environments the usual practice is one program for each application, leading to a reimplementation of the shared physics, the common data structures and also the parallelization, which often requires a considerable effort. Additionally, this replicated functionality has to be validated for each application, again leading to unnecessary work. Hence, there is a need for a fluid simulation library flexible enough to support research for several physical applications that cannot be simulated by existing software packages. The waLBerla software library has been designed to provide such a framework. It further aims at an efficient parallel implementation together with a good usability. For a detailed description of the features of waLBerla e.g. parallel simulation output or input descriptions see [FGD+07]. Most of today's flow simulations are based on numerical schemes that solve the Navier-Stokes (NS) equations directly. However, there exists an alternative approach named lattice Boltzmann method (LBM). This method is based on solving an approximation of the Boltzmann equation and thus is a kinetic-based approach. For the waLBerla software library the LBM has been chosen due to its advantages for the parallelization as well as its suitability for the scheduled applications. These applications cover moving charged colloids [HF01], fluid flow in blood vessels [SGR+07] and multiphase flows through micro porous media in fuel cells. In addition to a design flexible enough to include further applications, the framework has to be suitable for the simulation of large domains, e.g. a representative volume (REV) of the gas diffusion layer (GDL) in a polymer electrolyte fuel cell [Fue07]. The reason for the need of a large domain in the example above is, that the size of the REV[1] is about $0.45 \times 0.45 \times 0.1$ mm^3 and that the volume of a lattice cell has to be $\delta x^3 = 0.1^3 \mu\text{m}^3$, due to accuracy reasons and the limitation of the LBM to small Knudsen Numbers. With a 10% porosity this leads to $1.8 \cdot 10^{10}$ fluid cells, which results in a memory requirement of about 6.5 TB for the LBM (see following paragraph on LBM and [DGF+07] on resource requirements of waLBerla). Such a simulation is not feasible on a single CPU. Hence, the framework has to be adapted for massively parallel architectures.

Therefore one major target is the utilization of the HLRB II which is an SGI Altix 4700 [HLR07] featuring 39 TB memory. A rough estimation shows the power of this machine: With the performance of the current implementation of waLBerla (see Sec. 4.1) the above example results in a theoretical computation time of about 3 hours per time step, given a single core CPU with enough memory. Assuming a parallel efficiency of 70%, a time step would take about 1.5 seconds on the 4864 dual-core CPUs of the HLRB II. Thus

[1]Minimum size of REV based on findings from internal projects. No publications yet.

running fifty thousand time steps would require about 20 hours, instead of 17 years. Hence, only with an efficient parallelization it is possible to simulate the fluid flow in a GDL.

The remainder of this paper is organized as follows: In the subsequent paragraph a brief overview of the LBM is given, followed by the introduction of the waLBerla patch concept in Sec. 2. These patches are subdivisions of the fluid domain, which are the basic components for the parallelization, the optimization strategies, and the flexibility which is needed for the integration of further applications. In Sec. 3 the implementation of the process-local and global communication is explained in detail. Performance results are given in Sec. 4, where the serial performance as well as the parallel performance for various architectures has been evaluated. This first investigation discusses the suitability of the parallel concept for massively parallel usage in basic geometries. The article is concluded in Sec. 5 with a summary and outlook.

Brief Introduction to the Lattice Boltzmann Method

The lattice Boltzmann method is one possible approach to solve CFD problems numerically. It originates from the lattice gas cellular automata (LGCA), whereas McNamara and Zanetti were the first to introduce the Boltzmann collision operator to LGCA in 1988 [MZ88]. Further work [HL97] has shown that the LBM can be directly derived from the continuous Boltzmann equation. Hence, it is independent of the LGCA and based on kinetic theory. It can also be shown that the LBM is equivalent to an explicit finite difference scheme of the Navier-Stokes equations with second order spatial accuracy and first order temporal accuracy [JKL05]. Amongst others the LBM has been successfully applied to free surface flows [KPR$^+$05], multiphase flows [SC93], flows through porous media [ZFB$^+$02], fluid mixtures [LG03, Asi06], blood flows [AHS06] and metal foams [KPR$^+$05].

The advantages of the LBM are the explicit update rule, the fast mesh generation due to the Cartesian grids, and that many macroscopic and hydrodynamic effects result from mesoscopic quantities. A detailed description of the LBM can be found in [Hän04][WG00][YMLS03]. In the remainder of this Section an overview of the governing equations of the LBM is provided.

For the waLBerla software library the D3Q19 stencil [QDL92] and the LBGK [WG00] model are used. With the D3Q19 stencil the LBM is based on cubic cells with 19 unknowns, the particle distribution functions (PDF) $f_\alpha(x_i, t)$, which are defined as the expected amount of particles in the volume δx^3 located at the lattice position x_i with the lattice velocity $e_{\alpha,i}$. The lattice direction α points towards the neighboring cells (see Fig. 1 for an illustration). Discretized in time and space the LBGK model is given in tensor

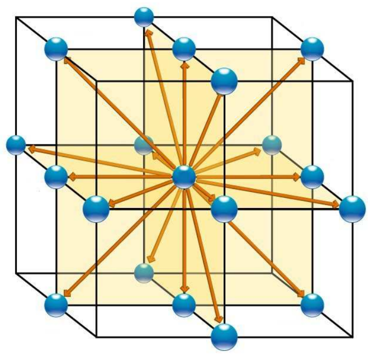

Figure 1: The D3Q19 stencil

notation by:

$$f_\alpha(x_i + e_{\alpha,i}\delta t, t + \delta t) - f_\alpha(x_i, t) = -\frac{\delta t}{\tau}\left[f_\alpha(x_i, t) - f_\alpha^{(eq)}(\rho(x_i, t), u_i(x_i, t))\right]. \tag{1}$$

Due to simplicity quantities depending on x_i and t will be written without their dependencies, e.g. $f_\alpha = f_\alpha(x_i, t)$. Further the relaxation time τ can be determined from the lattice viscosity (Eq. 7). The equilibrium distribution $f_\alpha^{(eq)}(\rho, u_i)$ (Eq. 2) depending on the macroscopic velocity u_i (Eq. 4) and the macroscopic density ρ (Eq. 5) for the isothermal case is given by the Maxwell-Boltzmann distribution function discretized for low mach numbers:

$$f_\alpha^{(eq)}(\rho, u_i) = \rho \cdot w_\alpha \cdot \left[1 + \frac{1}{c_s^2}(e_{\alpha,i} \cdot u_i) + \frac{1}{2c_s^4}(e_{\alpha,i} \cdot u_i)^2 - \frac{1}{2c_s^2}u_i^2\right]. \tag{2}$$

In the D3Q19 model the thermodynamic speed of sound is given by $c_s = \frac{1}{\sqrt{3}}$ and the lattice velocities $e_{\alpha,i}$ and lattice weights w_α are:

$$e_{\alpha,i} = \begin{cases} (0,0,0), \\ (\pm1,0,0), (0,\pm1,0), (0,0,\pm1), \\ (\pm1,\pm1,0), (0,\pm1,\pm1), (\pm1,0,\pm1) \end{cases} \quad w_\alpha = \begin{cases} 1/3, & \alpha = 0 \\ 1/18, & \alpha = 1, 2, 3, 4, 5, 6 \\ 1/36, & \alpha = 7, 8, 9, 10, 11, 12, \\ & \quad\; 13, 14, 15, 16, 17, 18 \end{cases}. \tag{3}$$

The macroscopic quantities of interest (ρ, p, u_i) are determined from the moments of the distribution functions:

$$\rho u_i = \textstyle\sum_{\alpha=0}^{18} e_{\alpha,i} \cdot f_\alpha = \displaystyle\sum_{\alpha=0}^{18} e_{\alpha,i} \cdot f_\alpha^{(eq)}(\rho, u_i), \tag{4}$$

$$\rho = \textstyle\sum_{\alpha=0}^{18} f_\alpha = \displaystyle\sum_{\alpha=0}^{18} f_\alpha^{(eq)}(\rho, u_i), \tag{5}$$

$$p = c_s^2 \rho, \tag{6}$$

$$\nu = (\tau - \tfrac{1}{2}) c_s^2 \delta t \ . \tag{7}$$

Due to the dependencies in Eq. 1 two grids are needed to store the PDFs. Hence, $2 \cdot 19$ *double* values are needed per lattice cell. Additionally, the update rule (Eq. 1) for each cell only depends on the neighboring cells. This locality can be exploited for an efficient parallelization. For further details on the actual implementation of the LBM in the waLBerla framework see [FGD+07].

2 Introduction of Patches

One of the goals of the waLBerla project is a highly scalable parallelization adequate for supercomputers. In order to create a parallelization suitable for several thousand cores it is essential to subdivide the global simulation domain into small, rectangular blocks that are independent of each other except for the necessary communication among the boundaries. In the waLBerla framework these blocks are called patches. In addition to the sole purpose of communication, patches are an adequate tool to realize different simulation requirements like free surfaces, moving rigid objects, etc in parts of the domain (see Fig. 2). This can be exploited to increase the performance of the simulation: special treatment for free surfaces or rigid objects is only enabled in patches where it is needed, whereas a pure fluid patch can be optimized for performance. Furthermore, the patches can also be distributed on a hybrid computer architecture. E.g. simple pure fluid patches could be calculated on the Cell processor and communicate to the more difficult patches via MPI, which are calculated on standard processors.

Since the waLBerla software library is written in C++, these thoughts can directly lead to the idea of a hierarchical patch design, which introduces specialized patches based on an abstract base class. Whereas the introduction of such a hierarchy seems to be the natural way in C++, a number of problems have been encountered with this hierarchy that challenged its application in the waLBerla framework (for more details see [FGD+07]). The most important of the problems is that a hierarchical patch design would in-

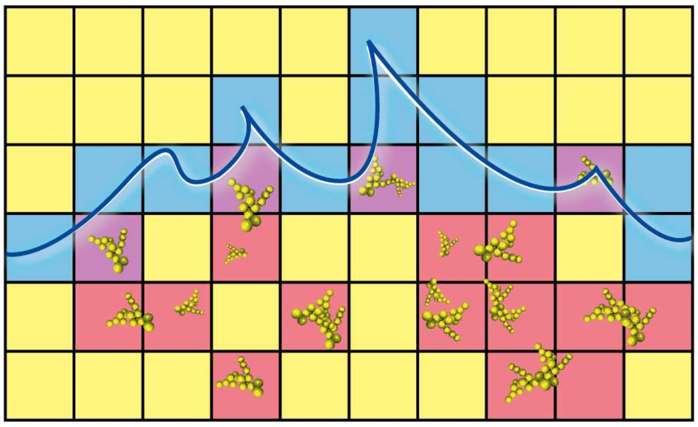

Figure 2: Two-dimensional sketch of differently skilled patches interacting with each other.

troduce multiple inheritance and therefore virtual inheritance to avoid duplicate data members. However, virtual inheritance introduces an additional indirection to the access of the data members of the common base class, which leads to a performance impact intolerable for a high performance implementation. Thus for the waLBerla framework we decided against a hierarchical patch design and use the approach illustrated in Fig. 3. Here, the single patch class *CalcPatch* handles all simulation aspects. Instead of creating

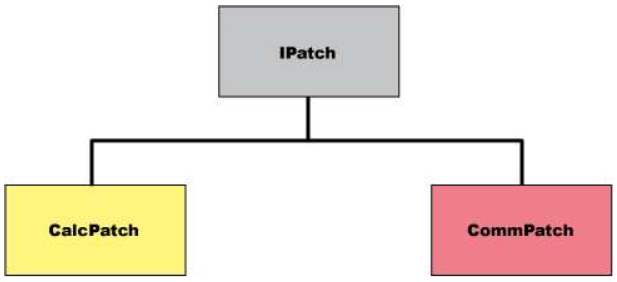

Figure 3: Implementation possibility of choice for patch class hierarchy.

different classes for the simulation features, a single *CalcPatch* class uses different functions, which can be activated or deactivated, depending on the current situation. If these are again composed by basic inline functions a high code reuse among the different applications can be ensured. In a pure fluid patch only the standard lattice Boltzmann treatment is enabled, which allows a fast processing. The only other patch class besides the *CalcPatch* is the *CommPatch*, which handles the communication across process boundaries (see Sec. 3). With this design the flexibility needed for the optimizations and integration of various applications is achieved.

Memory Reduction

Next to the primary purposes of building chunks of memory that can be distributed among several processes and to distinguish between different simulation requirements, the patch concept can also be used to reduce the overall amount of memory required for the simulation. The idea of this approach is described in detail in a previous work by Götz [Göt06] who is dealing with LBM blood flow simulations. Due to the complex, arbitrary geometry of blood vessels and the Cartesian grid setup of the LBM, a large fraction of LBM cells ends up as solid nodes. In order to save memory, the domain is subdivided into patches and then all patches with no fluid cells are removed (see Fig. 4). This approach can also be used efficiently for porous media. However, the downside of this strategy is the additional communication across the boundaries of the patches, when the domain is split into several patches on one process.

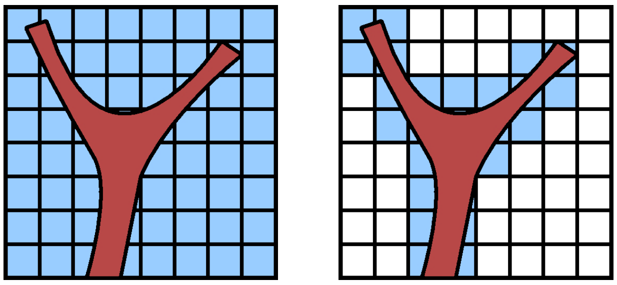
Figure 4: Two-dimensional sketch of a vessel bifurcation and patches that can be omitted (white) in order to save memory.

3 Communication Concept

For a scalable simulation on a large number of processors, an efficient communication concept is inevitable. With the introduction of patches in Sec. 2 the communication is divided into local and global communication. Patches on the same process exchange their data via local communication, whereas patches on different processes communicate by using MPI. For the D3Q19 model a patch has to communicate with at most 18 neighboring patches. Thereby, only the necessary PDFs have to be transferred: 5 for cells lying adjacent to boundary planes, 1 for edges and nothing for corners. Each patch stores a structure containing the neighborhood information, which is set up in the beginning of the simulation by first cutting the domain into *CalcPatch*es and then assigning the neighbors (either *CalcPatch*es or *CommPatch*es) to each patch. For the parallel case each process allocates a patch grid of the whole simulation domain including the neighborhood structures, but only allocates data fields in its own patches. The placement of the patches onto processes is either done via a Cartesian subdivision of the domain or by distributing an equal number of patches to the processes. In order to treat local and global communication in a similar way, the patch class *IPatch* holds a common interface for the communication routines. Thus a patch does not need to know if its neighbor is a *CalcPatch* or *CommPatch* and can just call the *Send()* routine of its neighboring patches. In the next two paragraphs the difference in process-local communication and communication across process boundaries is discussed.

3.1 Process-Local Communication

For the process-local communication a *CalcPatch* communicates with a neighboring *CalcPatch*, which is

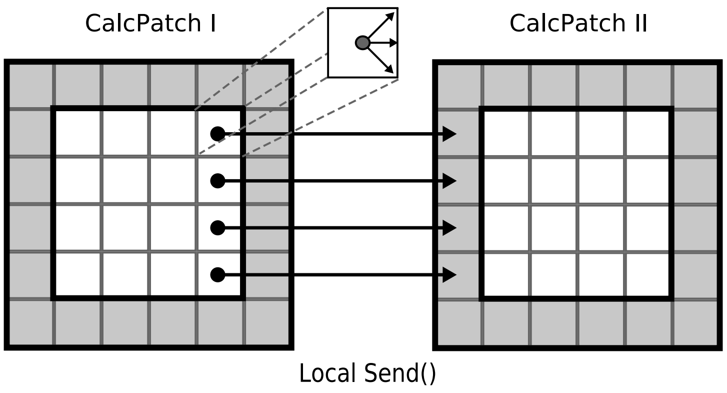

Figure 5: Process-local communication from CalcPatch I to CalcPatch II.

depicted in Fig. 5. The currently sending patch (CalcPatch I) calls the *Send()* function of the receiving patch (CalcPatch II). This function directly copies the data from the source data fields into the ghost nodes of the target data fields, without using any buffers in between.

3.2 Global Communication

The global communication concept is illustrated in Fig. 6. Here, the *CalcPatch I* has to transfer its data to a second *CalcPatch II* on a different process using MPI. In order to make this action transparent to the *CalcPath I*, it locally communicates with a *CommPatch I* as described in the previous paragraph. The *Send()* routine of the *CommPatch* then copies the data into its *SendBuffer* and sends it with the MPI command *MPI_Isend* to *CommPatch II.* Afterwards it issues an appropriate *MPI_Irecv* to receive data from *CommPatch II.* After the same procedure has been executed on the other site, the data sent by *CalcPatch I* is located in the *RecvBuffer* of *CommPatch II.* With an additional call of *CopyFromBuffer* the data is transferred to the ghost nodes of *CalcPatch II*. For each parallel communication a compatible send-receive pair is needed. We use the `tag` in MPI to match the messages on the processes by putting both IDs from the source and the target patch in the `tag`. Unfortunately the size of this identifier is limited and depends on the MPI implementation, which restricts the number of patches (see Sec. 4.2.1). Since non-blocking MPI commands are used for the global communication and all data transfer (process-local and global) is issued after the patches finished their calculations, both communications overlap in time.

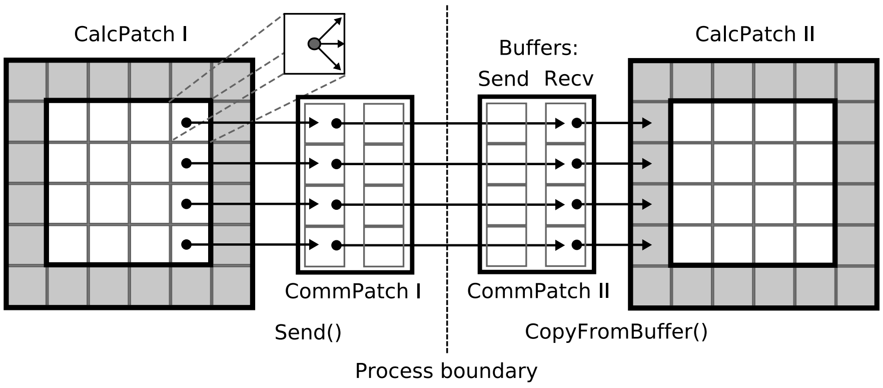

Figure 6: Global communication from CalcPatch I to CalcPatch II via two CommPatches.

4 Performance Studies

Before integrating more complex applications, the implementation of the parallelization has to be analyzed. Therefore, performance studies have been performed in order to quantify whether the concept is suitable for the use on massively parallel systems. The performance evaluation of the current implementation presented in this paper consists of serial experiments, weak and strong scaling measurements for the parallel performance as well as an examination of a multi-core implementation on the Cell Broadband Engine. For all studies a simple three-dimensional canal scenario with inflow and outflow conditions has been used. The results are given in terms of *million fluid lattice updates per second* (MFlups), which is an established performance measure in the lattice Boltzmann community since it allows for an estimation of the simulation runtime for a given problem size.

The later denoted parallel efficiency E is:

$$E(N,P) = \frac{S(N)}{P} = \frac{1}{P} \cdot \frac{\text{MFlups}(N,P)}{\text{MFlups}(N,1)} \cdot 100\%, \tag{8}$$

where S is the speedup gained by the use of P cores or nodes, and N is the problem size.
Two IA32-based clusters, namely the Woodcrest cluster at the Regional Computing Center of Erlangen (RRZE) and the Opteron cluster at the Chair for System Simulation (LSS) Erlangen, and an IA64-based supercomputer, the HLRB II [HLR07] at the Leibnitz Computing Center (LRZ) in Munich, are chosen for the evaluation. The experiments of Cell performance have been performed on the JUICE Cell Cluster [JUI07] at the Research Center Jülich.
The LSS cluster consists of 50 AMD Opteron processors resulting in a rough overall peak performance of about 220 GFlops. The nodes used for the benchmarks consist of four single-core CPUs with 4 GB dedicated memory each and are connected via Infiniband, providing a bandwidth of up to 10 GBit/s. On the Woodcrest cluster there are 217 2-socket nodes (HP DL140G3) with dual-core 64-bit enabled Intel Xeon 5160 CPUs (codename Woodcrest) and Infiniband interconnection. The rough overall peak performance of the system is about 10.3 TFlops. The HLRB II features 4846 dual-core Itanium 2 CPUs of Montecito type, each of which capable to address the whole shared memory of 39 TB capacity by non-uniform memory access (NUMA). The CPUs are interconnected by a hierarchically organized NUMAlink 4 network with a nominal bandwidth of 6.4 GB/s. This computer is listed as number 15 in the 30th TOP500 list [Top07] with an overall peak performance of 56.5 TFlops. The JUICE consists of 12 QS20 blades each equipped with 2 Cell processors and $2 \cdot 512$ MB memory.

4.1 Serial Experiments

The serial performance of a parallel code is important for the quantification of the quality of parallel scaling results, since the parallel efficiency depends on the serial performance (see Eq. 8). The presented results are restricted to one of the IA32 architectures and the IA64-based machine, namely the Woodcrest cluster and the HLRB II, whereby only for Woodcrest detailed benchmark comparisons are discussed. On the Woodcrest, the theoretical memory bandwidth of one node is 21.3 GB/s. However, in order to estimate the upper limit of the possible memory throughput for an LBM solver Zeiser et al. [ZGS07] suggest to compare to the STREAM [McC07] vector-triad benchmark. Table 1 shows that the maximum achievable data transfer rate is around 6 GiB/s only[2]. On architectures that perform a read for ownership before a write, waLBerla transfers 524 Bytes per cell update (for details see [FGD+07]). The single-core performance of waLBerla obtains 4.4 MFlups which corresponds to a bandwidth usage of 2.3 GiB/s. Compared to the STREAM triad, a single waLBerla process uses 70% of the available usable bandwidth.

[2]The STREAM benchmark uses a base of 1000 for the orders of magnitude, thus 1 GiB/s = 10^9 B/s

Configuration	1 Process		2 Processes 1 Socket		2 Processes 2 Sockets		4 Processes	
	MFlups	GiB/s	MFlups	GiB/s	MFlups	GiB/s	MFlups	GiB/s
waLBerla	4.4	2.31	6.4	3.35	8.6	4.51	11.4	5.97
STREAM Triad		3.32		3.30		6.09		6.04

Table 1: Performance and memory transfer rates of waLBerla and STREAM benchmark on one node of Woodcrest cluster with different number of processes and different placements. STREAM benchmark values by courtesy of [Zei07].

Contrary to IA32-based architectures, achieving high performance on IA64 machines is generally more difficult. The in-order architecture requires explicit vectorization and thus the performance often relies on the capabilities of the compiler and the use of appropriate pragmas in the code. Performance optimization of LBM for Itanium 2 is a well-explored task in our group [WZHD06]. However, many findings that enhance the performance of simple kernels cannot easily be applied to complicated programs like waLBerla. In order to feature a suitable framework for the complex algorithms of the real-life applications to be implemented, the class design contains complex structures that are not comparable to flat loops of simple kernels. Currently, the single-core performance of 1.68 MFlups represents only 39% of the performance on the Woodcrest cluster.

Influence of the Process-Local Communication

For an estimation of the influence the local communication has on the performance, a domain of 100^3 lattice cells has been simulated with an increasing number of patches. While the size of the domain remains constant, the number of patches is varied from 1 to 1000. In each simulation run, all patches have the same size, resulting in 100^3 to 10^3 lattice cells per patch. If the domain does not fit exactly into the patches, the remaining cells in the patches are marked as obstacles, not influencing the MFlups rate. The results (Fig. 7) show that the loss in performance compared to the situation without local communication is architecture dependent. On the LSS cluster the performance falls by about 13% for 64 patches and about 28% for 1000 patches, while on the Woodcrest cluster the loss is higher with 14% and 35%, respectively. On the HLRB II, the performance losses are smoother with 7% and 23%, respectively. However, this results from the small fraction of time which the copy operations of the local communication consume compared to the slow computations.

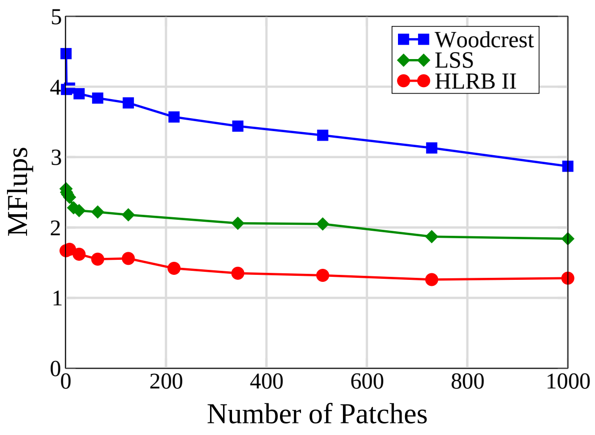

Figure 7: Influence of the local communication on the performance.

4.2 Parallel Experiments

The parallel efficiency is measured by different parallel experiments: Weak scaling tests on the Woodcrest cluster and the HLRB II show the influence of both process-local and MPI communication. Strong scaling experiments on all three clusters test the proportion of MPI communication to computational time.
As Tab. 1 shows, the memory architecture of the Woodcrest cluster has a tremendous effect on concurrently running processes. Apparently, the chipset cannot provide the full achievable memory bandwidth to a single socket, but only nearly the half. Employing the second core on the same socket does not improve the bandwidth usage. In the case of waLBerla one single process cannot utilize the full bandwidth one socket could achieve. Thus, employing the second core on the same socket can increase the usage by 45% until the machine's limit is reached. However, using two processes on separate sockets or two fully-employed sockets does not double the performance and bandwidth usage, similarly to the STREAM benchmark. Using four cores results in a performance of 11.4 MFlups and a data transfer rate of 5.9 GiB/s from memory, which equals the bandwidth usage of STREAM benchmark. The effect of getting not much more than half of the maximum achievable bandwidth on one socket is attributed to inherent limitations of the memory controller chipset. As a consequence, a comparison of the parallel scaling measurements based on fully-employed sockets instead of single cores is preferable. However, running less than four processes without explicit process pinning can let to different results depending on their distribution on the sockets (see cols. 2 and 3 of Tab. 1). Therefore, the parallel performance experiments on Woodcrest cluster in this paper always have been conducted on fully-employed nodes with four MPI processes, and the graphs are based on the number of nodes used. On the LSS cluster the performance scales well with

the number of cores because every core accesses its dedicated memory. For the HLRB II the term "node" is not applicable. The smallest unit in the hierarchy is a blade, featuring one (high-bandwidth blades) or two (high-density blades) dual-core CPUs that share one connection to the NUMAlink 4 network. This connection is able to feed each core with the full bandwidth. Thus, the graphs of the performance experiments of the latter two architectures base on the number of cores.

4.2.1 Weak Scaling

For the weak scaling test the overall size of the problem is not fixed, but scaled by the number of processes. Thus the problem size on each process is constant for all simulation runs. This test is especially well-suited to measure the maximal overall parallel performance of the code and to determine the runtime to be expected for real-world applications. The weak scaling benchmark in Fig. 8 has been carried out on the Woodcrest cluster. From 1 to 203 nodes each core processes one patch containing 100^3 lattice cells such that the largest system size for this weak scaling experiment is $1000 \times 900 \times 900$ consuming approx. 300 GB (measured value). Showing a nearly linear speed up, the parallel efficiency is 95.7% for 183 and 93.3% for 203 nodes. Since CFD applications often need large simulation domains up to and beyond 1000^3 lattice cells, the parallel efficiency of the code was tested in a single experiment with large memory consumption. This test resulted in 94.4% parallel efficiency on 729 cores for a simulated system size of 1530^3 and 1.2 TB. An investigation of the scaling behavior depending on the amount

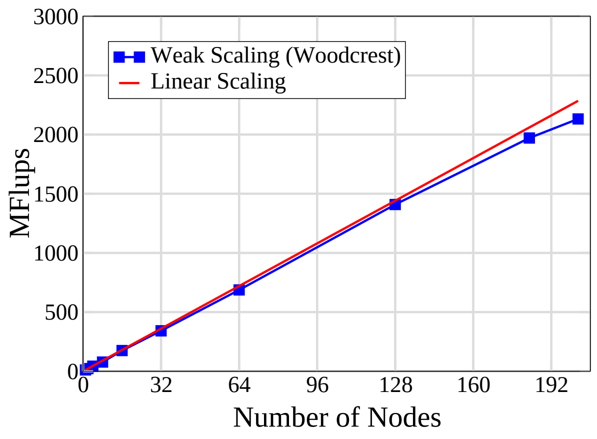

Figure 8: Weak scaling experiment of waLBerla on the Woodcrest cluster with one single patch per process in comparison to the ideal scaling.

of process-local communication (see Fig. 9 and Fig. 10) reveals that the MPI communication is not as expensive as expected due to the low latencies and high bandwidths of the Infiniband and NUMAlink4 interconnect on the one hand, and the overlapping of local and MPI communication in time on the other. In this weak scaling experiments again, every process computes a domain with 100^3 lattice cells, but with

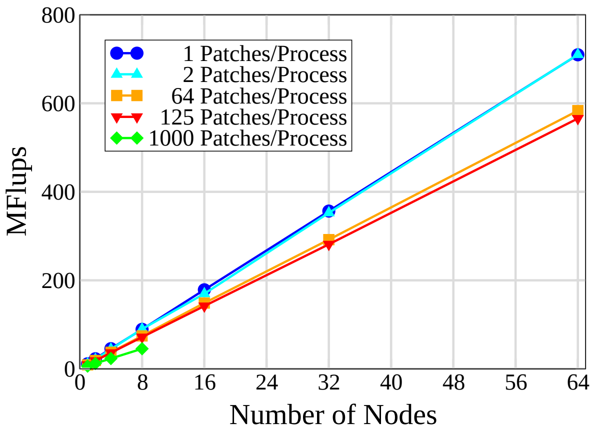

Figure 9: Weak scaling experiment of waLBerla on the Woodcrest cluster with different numbers of patches per process in comparison to the single-patch version.

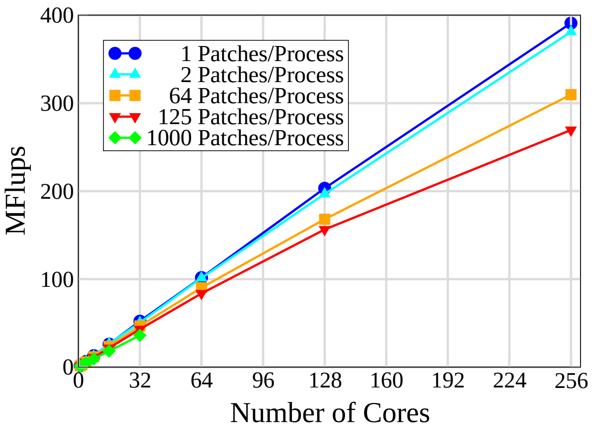

Figure 10: Weak scaling experiment of waLBerla on high-bandwidth blades of the HLRB II with different numbers of patches per process in comparison to the single-patch version.

different number of patches per process in each experiment. This does not only increase the effort of the local communication but also of the MPI communication, since every patch sends its data to its remote neighbors via a separate MPI call. Thus, a process with one patch that has neighbors to all sides sends 18 messages per time step. With two patches it sends 34 messages, with 125 patches 690, and with 1000 patches even 2880. The performance impact in comparison to the single-patch version is for each scenario relatively constant over the increasing process number. With 64 patches per process the performance loss is 18%, with 125 patches 21% and with 1000 patches around 50%. The single-core experiment above resulted in 14%, 16% and 35%, respectively. Thus, the overhead of MPI communication is low, which is attributed to the small latencies of the Infiniband network, but mainly to the fact that local and global communication overlap in time (as described in Sec. 3.2). In fact, a closer measurement with the Intel TraceAnalyzer shows that 5.6% of the total time are spent for MPI communication with 16 processes and 125 patches per process. Despite of the good results, further optimization of the MPI communication is inevitable: The weak scaling experiment with 1000 patches per process could not be performed with more than 8 nodes on the Woodcrest and 32 cores on the HLRB II, respectively, because of a limitation in the current implementation. The `tag` parameter for MPI communication is used to code the sending

and receiving patch by using a globally unique ID for the patches. When using more than 32768 patches, the 32-bit signed integer value overflows. Although for real-world applications the use of 1000 patches per process is very unlikely, simulations using more than 9000 cores—as planned on the HLRB II—can quickly reach the limit. Therefore, a future implementation will combine the messages of the patches of the same process in order to reduce the message count and evade the patch number limitation.
The NUMAlink 4 interconnect of HLRB II, also having low latencies and a high bandwidth, has a low impact on the performance in this scaling experiment, however the hierarchical structure influences the results (see Fig. 10). Up to 32 processes, the average performance losses of the parallel version of 10% for 64 patches, and 30% for 1000 patches compare well to the 7%, and 23% of the serial performance, respectively. From 64 to 128 cores as well as from 128 to 256 a decrease in parallel efficiency can be clearly determined. One compute partition of the HLRB II consists of 512 cores that are arranged in groups of 64 cores. Thus, messages between cores from different groups have to hop over routers of a higher hierarchy level. This effect can be neglected when the amount of communication is small, as in the case of one patch per process, while 64 and more patches per process experience a noticeable impact. This is another reason for changing the communication implementation in order to reduce the message count.

4.2.2 Strong Scaling

The strong scaling scenario has been performed with a size of 100^3 lattice cells. Here, the performance of the implementation is measured with an increasing number of cores, whereas the overall size of the problem remains fixed. This scenario enables to estimate the shortening of computation time when a higher number of processes is employed for solving the same problem. To neglect the effects of local communication one patch per MPI process has been used. The measurements for the LSS cluster are shown in Fig. 11. With 32 processes the parallel efficiency only drops to 75%. For the strong scaling on the Woodcrest cluster (see Fig. 12) up to 16 nodes have been used, each node running four MPI processes. The resulting parallel efficiency on 16 nodes is about 84.9%. It has to be noted that for this result the patch size of 25^3 does not fit into the cache.
Since for the same problem size the Itanium 2 based implementation is slower, while having the same amount of communication, the ratio between work and communication is larger than on IA32-based architectures. According to

$$S = \frac{1}{(1-p) + p/P}\,, \qquad \text{(Amdahl's Law)} \qquad (9)$$

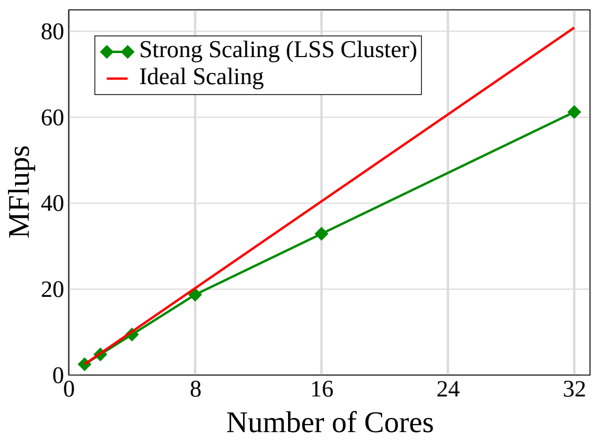

Figure 11: Strong scaling experiment of waLBerla on the LSS Cluster.

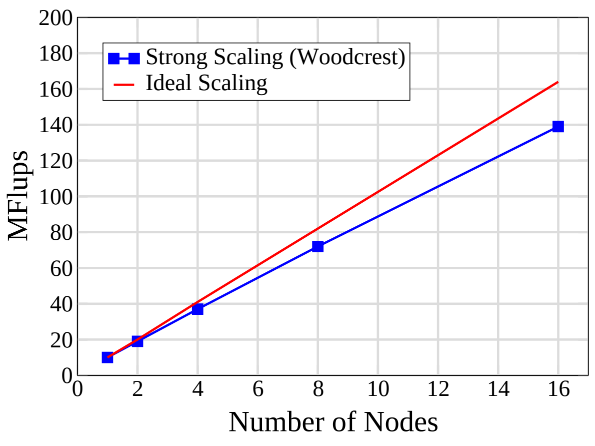

Figure 12: Strong scaling experiment of waLBerla on the Woodcrest Cluster.

an increase of the parallelizable part p results in a higher scalability on the same number of cores P. Therefore, the scaling behavior in a strong-scaling experiment on HLRB II shows a good efficiency: With 32 cores the efficiency of up to 93.5% is much better than on Woodcrest cluster, where 8 nodes (32 cores) reach only 87.8%. Figure 13 shows the scaling on high density and high bandwidth blades. From

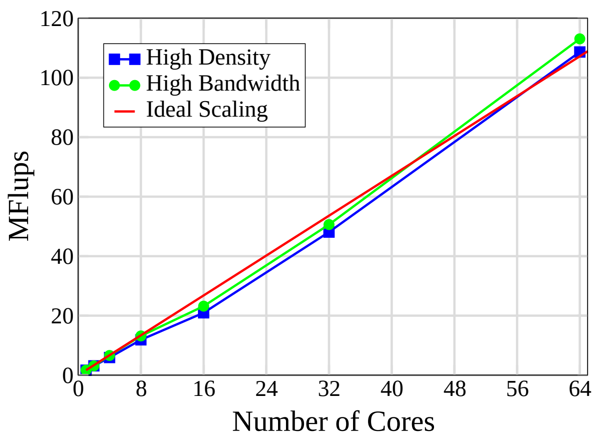

Figure 13: Strong scaling experiment on both high density and high bandwidth blades of HLRB II.

8 to 16 cores one can clearly determine the additional overhead induced by the communication across the boundaries of so-called building blocks (high bandwidth building blocks consist of 8 cores), which represent a lower part of the sophisticated hierarchy in HLRB II. Due to the large caches, for 64 cores the system size per process is small enough to fit completely into the cache, which results in super-linear scaling. Since the communication via the NUMAlink 4 network shares the bus to the memory controller,

it is obvious that the scaling on high density nodes is worse.

4.3 IBM Cell processor

In order to investigate optimization techniques for special hardware and multi-core systems our group's research includes the IBM Cell processor, which is the heart of Sony's Playstation III gaming console. This hybrid multi-core processor combines one Power processor element (PPE), which is a PowerPC compliant general purpose core, and eight simple single instruction multiple data (SIMD) cores, so-called synergistic processor elements (SPEs). The PPE is mainly responsible to run the operating system and for program control, whereas the SPEs are optimized for efficient data processing. In the Playstation III only six of the SPEs are available to be used for programming. Whether with Playstations or as blades, this processor can be a valuable asset to a hybrid cluster enabling high performance for appropriate codes. Using the Cell, our group implemented a blood flow simulation [Göt06] with similar concepts as in Sec. 2. With just compiling a straightforward implementation, one gains a meager performance of 2 MFlups on a single SPE, possibly summing up to 12 MFlups on a Playstation, after all. However, Stürmer et al. [SGR+07] showed that with architecture-related optimizations up to 95 MFlups are possible. Unfortunately, these results base on single-precision floating point operations. Since the code performance is limited by the memory bus, one can estimate that performance decreases by a factor of 2.5 for double precision, resulting in a sustained performance of around 40 MFlups.

5 Conclusion

In this work the parallelization concept for the waLBerla framework has been presented, which aims at the integration of various CFD applications together with an efficient massively parallel implementation. The key component for our realization of these aims is the patch concept, which supports the specialization to different applications as well as domain decomposition needed for parallelization. In order to verify the suitability of the parallelization for massively parallel usage the serial and the parallel performance have been investigated on different architectures.

For the serial performance it has been demonstrated that the sustained memory throughput is 70 % of the maximum throughput (STREAM triad) on the Woodcrest cluster which gives an indication of the quality of the serial performance. Additionally, the influence of the process-local communication has been discussed and it has been shown that the performance with 64 patches only drops by 14% on the Woodcrest cluster, 7% on the HLRB II and 13% on the LSS cluster.

In the parallel case weak scaling scenarios up to $1000 \times 900 \times 900$ lattice cells have been performed for process-local plus MPI communication and for pure MPI communication. The results show that the basic concept of the framework is suitable for massively parallel usage, as a parallel performance of about 94% has been achieved for 203(810) nodes(cores) on the Woodcrest cluster. Furthermore, the scenarios with several patches per process indicated that the MPI communication has a smaller impact on the performance than the local communication. The reason for that is the overlapping of local and MPI communication as well as the low latency of the interconnects. In addition to the weak scaling scenarios, strong scaling experiments have been performed. On the IA32-based architectures these result with 32 cores in 87.8% parallel efficiency compared to 93.5% on the IA64-based HLRB II.

In future work the efficiency of the framework will be tested on the HLRB II beyond 1000 cores. For this purpose the communication will be redesigned in order to reduce the overall message count, as the connection network of the HLRB II has a strong influence on the efficiency when dealing with a high amount of messages. With further work the CFD applications e.g. free surfaces and moving objects will be implemented together with a dynamic load balancing and the specialization of the patches.

References

[AHS06] A.M. Artoli, A.G. Hoekstra, and P.M.A. Sloot, *Mesoscopic simulations of systolic flow in the human abdominal aorta*, J. Biomech. **39** (2006), no. 5, 873 – 884.

[Asi06] P. Asinari, *Semi-implicit-linearized multiple-relaxation-time formulation of lattice Boltzmann schemes for mixture modeling*, Phys. Rev. E **73** (2006), no. 5, 056705.

[DGF$^+$07] S. Donath, J. Götz, C. Feichtinger, K. Iglberger, S. Bergler, and U. Rüde, *On the Resource Requirements of the Hyper-Scale waLBerla Project*, Tech. Report 07-13, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2007.

[FGD$^+$07] C. Feichtinger, J. Götz, S. Donath, K. Iglberger, and U. Rüde, *Concepts of waLBerla prototype 0.1*, Tech. Report 07–10, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2007.

[Fue07] *Information on fuel cells*, `http://www.fuelcells.org`, Sept. 2007.

[Göt06] J. Götz, *Numerical Simulation of Blood Flow with Lattice Boltzmann Methods*, Master's thesis, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2006.

[Hän04] D. Hänel, *Molekulare Gasdynamik*, Springer, 2004.

[HF01] J. Horbach and D. Frenkel, *Lattice-Boltzmann method for the simulation of transport phenomena in charged colloids*, Phys. Rev. E **64** (2001), no. 6, 061507.

[HL97] X. He and L.-S. Luo, *Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation*, Phys. Rev. E **56** (1997), no. 6, 6811–6817.

[HLR07] *Information on the HLRB II*, `http://www.lrz-muenchen.de/services/compute/hlrb/`, Nov. 2007.

[JKL05] M. Junk, A. Klar, and L.-S. Luo, *Asymptotic analysis of the lattice Boltzmann equation*, J. Comput. Phys. **210** (2005), no. 2, 676–704.

[JUI07] *Information on the Juelicher Initiative Cell Cluster (JUICE)*, `http://www.fz-juelich.de/jsc/service/juice`, Nov. 2007.

[KPR$^+$05] C. Körner, T. Pohl, U. Rüde, N. Thürey, and T. Hofmann, *FreeWIHR: Lattice Boltzmann Methods with Free Surfaces and their Application in Material Technology*, High Performance Computing in Science and Engineering, Garching 2004 (A. Bode and F. Durst, eds.), Springer, 2005, pp. 225–236.

[LG03] L.-S. Luo and S.S. Girimaji, *Theory of the lattice Boltzmann method: Two-fluid model for binary mixtures*, Phys. Rev. E **67** (2003), no. 3, 036302.

[McC07] J. D. McCalpin, *STREAM: Sustainable memory bandwidth in high performance computers*, `http://www.cs.virginia.edu/stream/`, 1991-2007.

[MZ88] G. McNamara and G. Zanetti, *Use of the Boltzman Equation to Simulate Lattice Gas Automata*, Phys. Rev. Lett. **61** (1988), no. 20, 2332 – 2335.

[QDL92] Y. H. Qian, D. D'Humires, and P. Lallemand, *Lattice BGK Models for Navier-Stokes Equation*, Europhys. Lett. **17** (1992), no. 6, 479–484.

[SC93] X. Shan and H. Chen, *Lattice Boltzmann model for simulating flows with multiple phases and components*, Phys. Rev. E **47** (1993), no. 3, 1815–1819.

[SGR$^+$07] M. Stürmer, J. Götz, G. Richter, A. Dörfler, and U. Rüde, *Blood flow simulation on the Cell Broadband Engine using the lattice Boltzmann method*, Tech. Report 07–9, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2007, submitted to the International Conference for Mesoscopic Methods in Engineering and Science, ICMMES.

[Top07] Top500, *The top 500 supercomputer sites, 30th list, released at Reno during SC07*, `http://www.top500.org`, 2007.

[WG00] D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models*, Springer, 2000.

[WZHD06] G. Wellein, T. Zeiser, G. Hager, and S. Donath, *On the single processor performance of simple Lattice Boltzmann kernels*, Computers & Fluids **35** (2006), no. 8–9, 910–919.

[YMLS03] D. Yu, R. Mei, L.-S. Luo, and W. Shyy, *Viscous flow computation with the method of lattice Boltzmann equation*, Prog. Aero. Sci. **39** (2003), no. 5, 329 – 367.

[Zei07] T. Zeiser, *private correspondence with Thomas Zeiser, Regional Computing Center Erlangen (RRZE)*, Nov. 2007.

[ZFB$^+$02] T. Zeiser, H.-J. Freund, J. Bernsdorf, P. Lammers, G. Brenner, and F. Durst, *Detailed Simulation of Transport Processes in Reacting Multi-Species Flows Through Complex Geometries by Means of the Lattice Boltzmann method*, In High Performance Computing in Science and Engineering '01, Transactions of the High Performance Computing Center Stuttgart (HLRS), Springer, 2002.

[ZGS07] T. Zeiser, J. Götz, and M. Stürmer, *On performance and accuracy of lattice Boltzmann approaches for single phase flow in porous media*, Proceedings of 3rd Russian-German Workshop on High Performance Computing, Novosibirsk, July 2007, to appear in a Springer book, 2007.