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

Lehrstuhl für Informatik 10 (Systemsimulation)

Single Node Performance and Weak Scaling of waLBerla on current Supercomputing Platforms

Jan Götz, S. Donath, C. Feichtinger, U. Rüde

Technical Report 09-16

Single Node Performance and Weak Scaling of waLBerla on current Supercomputing Platforms

Jan Götz, S. Donath, C. Feichtinger, U. Rüde

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

jan.goetz@informatik.uni-erlangen.de

November 23, 2009

Abstract

High performance on current supercomputers is mainly obtained by highly optimized kernels, which handle one specific problem with given restrictions. waLBerla in contrast is a large C++ software framework and aims at high performance . It is our extensible implementation of a parallel fluid solver, and supports various applications. In this report we evaluate the perfomance of the fluid solver on current supercompute platforms and test its suitability for these machines. The single node performance and weak scaling is discussed and compared between the computers. Influences of the data layout, the compiler and different kernel routines are investigated.

1 Introduction

In the era of multicore CPUs, single processor and single node optimization is especially important to exploit modern supercomputers. This is a well explored task in our group [WZDH05, DIW$^+$08, STR08]. While utilizing processors efficiently for numerical kernel routines is complex and time-consuming, this is even more difficult and time consuming for large software frameworks, like the waLBerla project. waLBerla is our implementation of a massively parallel lattice Boltzmann fluid solver with various add-on applications. It will be covered in Chapter 3 of this report. For many applications like blood flow or the fluid structure interaction with moving objects incorporated in the flow, the fluid solver is the bottleneck in performance and should be optimized first. In this report the single node performance of the fluid solver is evaluated on current supercomputing platforms like the Woodcrest cluster at RRZE Erlangen, the hlrb 2 at LRZ Garching and the JuRoPA at FZ Jülich (see Chapter 4). Improvements of the data layout and optimization are discussed. Furthermore, parallel weak scaling on the architectures is presented on up to 4096 compute cores.

2 Numerical Method

2.1 Lattice Boltzmann Method

The lattice Boltzmann method (lbm) is an alternative to classical Navier–Stokes solvers for fluid flow. It uses an equidistant grid of lattice cells, which interact in each time step only with their direct neighbors. In this study we use the common three dimensional D3Q19 model originally developed by Qian, d'Humiéres and Lallemand [QDL92] with $N = 19$ particle distribution functions (PDFs) $f_\alpha : \Omega \times T \mapsto [0; 1)$, where $\Omega \subset \mathbb{R}^3$ and $T \subset \mathbb{R}$ are the physical and time domain, respectively. The corresponding dimensionless discrete velocity set is denoted by $\{\mathbf{e}_\alpha | \alpha = 0, \ldots, N-1\}$. This model has been shown to be both stable and efficient [MSYL02]. For the work presented in this paper, we adopt a lattice Boltzmann collision scheme proposed by Bhatnagar, Gross and Krook (LBGK) [BGK54, QDL92]

$$f_\alpha(\mathbf{x}_i + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}_i, t) - \frac{1}{\tau}[f_\alpha(\mathbf{x}_i, t) - f_\alpha^{(\mathrm{eq})}(\mathbf{x}_i, t)]\,, \tag{1}$$

where $\mathbf{x}_i$ is a cell in the discretized simulation domain, τ is the relaxation time, t is the current time step whereas $t + \Delta t$ is the next time step, and $f_\alpha^{(\mathrm{eq})}$ represents the equilibrium distribution. For an incompressible LBGK scheme this leads to [HL97]

$$f_\alpha^{(\mathrm{eq})}(\mathbf{x}_i, t) = w_\alpha \left[\rho(\mathbf{x}_i, t) + \rho_0 \left(3\mathbf{e}_\alpha \mathbf{u}(\mathbf{x}_i, t) + \frac{9}{2}(\mathbf{e}_\alpha \mathbf{u}(\mathbf{x}_i, t))^2 - \frac{3}{2}\mathbf{u}(\mathbf{x}_i, t)^2\right)\right]. \tag{2}$$

Here, we choose $\rho_0 = 1$. The weighting factors w_α are depending on the discretization scheme and are chosen as in Succi et al. [Suc01]. The macroscopic fluid density ρ and velocity $\mathbf{u}$ is calculated from the first two moments of the distributions as

$$\rho(\mathbf{x}_i, t) = \rho_0 + \delta\rho(\mathbf{x}_i, t) = \sum_\alpha f_\alpha(\mathbf{x}_i, t) \tag{3}$$

and

$$\mathbf{u}(\mathbf{x}_i, t) = \frac{1}{\rho_0} \sum_\alpha \mathbf{e}_\alpha f_\alpha(\mathbf{x}_i, t)\,. \tag{4}$$

Equation (1) is separated into two steps, known as the collision step and the streaming step, respectively

$$\tilde{f}_\alpha(\mathbf{x}_i, t) = f_\alpha(\mathbf{x}_i, t) - \frac{1}{\tau}[f_\alpha(\mathbf{x}_i, t) - f_\alpha^{(\mathrm{eq})}(\mathbf{x}_i, t)]\,, \tag{5}$$

$$f_\alpha(\mathbf{x}_i + \mathbf{e}_\alpha \Delta t, t + \Delta t) = \tilde{f}_\alpha(\mathbf{x}_i, t)\,, \tag{6}$$

where $\tilde{f}_\alpha$ denotes the post-collision state of the distribution function. The collision step is a local single time relaxation towards equilibrium. While this is compute intensive, the streaming step advects all PDFs except f_0 to their neighboring lattice site depending on the velocity, which is a memory intensive operation.

As a first order no-slip boundary condition often a simple bounce-back scheme is used, where distribution functions pointing to a neighboring wall are just reflected such that both normal and tangential velocities vanish

$$f_{\bar{\alpha}}(\mathbf{x_f}, t) = \tilde{f}_\alpha(\mathbf{x_f}, t)\,, \tag{7}$$

with $\bar{\alpha}$ representing the index of the opposite direction of α, $\mathbf{e}_{\bar{\alpha}} = -\mathbf{e}_\alpha$, and $\mathbf{x_f}$ explicitly denoting the fluid cell. More details on the lattice Boltzmann algorithm and its derivation can be found in Succi et al. [Suc01] or Chen et al. [CD98].

2.2 Parallelization of the LBM Solver

Our parallelization is based on a subdomain partitioning for the LBM flow solver. Optimizing and parallelizing the LBM has been studied intensively see e.g. [PTD+04, WZDH05, KPR+05, WCO+08] and for cache and memory performance in particular [PKW+03, FS07, DIW+08].

However, the implementation of a flexible parallelization required by a high performance multi purpose simulation framework raises additional problems.

In the waLBerla framework we rely on a patch data structure described in [FGD+07] in combination with generic MPI communication. Generally, the flow domain is subdivided into patches of equal size, but several patches can be assigned to each process. The patches are surrounded by a ghost layer which is updated before each time step. The message sent from one process to another is composed from several smaller messages. To reduce the startup time overhead, data is accumulated in buffers before being sent to neighboring processes. The buffers store raw bytes in order to be able to communicate floating point data as well as integer values at the same time.

Each patch may contain structured data like the data for the PDFs, velocities and cell state data (Flags), as well as unstructured data like the moving objects, which is communicated to the neighboring processes in each time step. However, for the unstructured data messages of variable size have to be send. The resulting variable size of messages is supported by sending the data of one process via `MPI_ISend()` and by probing the message for its actual size with a `MPI_Probe()` on the receiving process. Finally, the data is received using `MPI_Receive()`.

3 Software

3.1 waLBerla

waLBerla (widely applicable lattice Boltzmann from Erlangen) is our C++ implementation of a lattice Boltzmann fluid solver for various applications like blood flow, moving objects, free surfaces and mixtures. It is incorporated in a flexible environment which supports extensions of the functionallity by using existing routines for data access, IO, visualization, etc. waLBerla is implemented to support different standard compute platforms and current supercomputers. An efficient parallelization based on a domain partitioning is embedded, which is described in detatil in [FGD+07].

3.2 Data Layouts

The choice of the data layout is important for the performance of the solver. For lattice Boltzmann we need four-dimensional arrays, three dimensions for the 3D coordinate system and one for the PDFs. Since the PDFs of the last time step are needed to calculate the current time step, either two data fields, or a compressed-grid storage [Igl03] are needed. In waLBerla two independent data fields are stored and two different data layouts are currently supported:

- The structure of arrays (SoA) layout, where the first dimension is used to store the PDFs and the remaining dimensions for the 3D coordinate directions.
- The array of structures layout (AoS), where the first three dimensions are used for the 3D coordinate system, followed by the PDFs.

The AoS layout provides spatial locality for the PDFs, which improves the performance of the part in the lattice Boltzmann kernel that performs local operations for one cell, that is the collision step. The SoA layout is optimized for streaming [WZDH05], but is more prone to cache thrashing effects.

3.3 Variable Length Arrays

Accessing one-dimensional arrays in C is simple, but getting more complicated for multi-dimensional arrays. In waLBerla the data class (i.e the DataLayout) is holding the data in

a one dimensional array and simplifies the access for the user by operator functions. These operator functions allow to vary between the different data layouts and calculate the index of the data entry depending on the sizes in each dimension. This involves expensive integer computations. The DataLayout class also hides the currently used data layout from the user. In the kernel routines the data is accessed via intermediate field classes, which additionally hinders the compiler to optimize the code efficiently.
In C99, the bounds of an array can be a run-time expression. Such arrays are called variable length arrays or VLAs for short [Mey01b] . The VLAs allow to use the classical array convention (i.e. array[i][j][k]) even when the sizes are not known at compile time. In waLBerla we added a VLA version of the fluid solver kernel routine, which accesses the data via VLA pointers, but is still using the common DataLayout class for holding the data. The routines are declared as external C functions in the C++ framework and compiled with a C99 compiler. With this approach the index calculation of array elements can be efficiently optimized by the compiler. For a performance comparison of the original code and the VLA code see Chapter 4. For more information on VLAs see the users journal of Randy Meyers [Mey01b, Mey01a, Mey02].

4 Performance Results

In this chapter we present performance and scalability results for the Woodcrest cluster of the Regional Computing Center of Erlangen (RRZE) [Woo], the hlrb 2, an sgi Altix system located at the Leibniz Supercomputing Centre in Garching [HLR], and the JuRoPA [Jur] computer located at Jülich Supercomputing Centre. For the simulations, on Woodcrest the Intel ICC compiler version 11.1.056 and GNU compiler version 4.3.3 were used. On the hlrb 2 the same version of ICC, but version 4.2.0 of GNU were used and on JuRoPA only ICC in version 11.0.74 was available. To compare the performance values, the results are given in terms of *million fluid lattice updates per second* (MFlups) [ZGS08]. This allows for an estimate of the runtime for a given problem size. Following the methodology of Zeiser et al. [ZGS08] we compare the LBM solver to the STREAM vector-triad benchmark [McC08]. Here the maximal achievable bandwidth of a parallel system is evaluated using the STREAM values obtained per node. The aggregated application memory performance is evaluated relative to the STREAM performance which is considered as a realistic achievable optimum. On architectures that perform a read for ownership before a write, waLBerla transfers 524 Bytes per cell update (see [FGD+07]) neglecting cache effects.

4.1 Machines

4.1.1 Woodcrest

The Woodcrest cluster at RRZE is IA32-based. It consists of 217 2-socket nodes (HP DL140G3) with dual-core 64-bit enabled Intel Xeon 5160 CPUs (codename Woodcrest) and Infiniband interconnection organized in a fat-tree topology. The rough overall peak performance of the system is about 10.3 TFlops. Each dual core processor on a two socket node is attached to 4 GB of main memory and 4 MB shared L2 cache and delivers 48 GFlop/s.

4.1.2 HLRB 2

This computer features an overall main memory size of 39 TB and a peak performance of 62.3 TFlops, delivered by 4864 dual-core 1.6 GHz Itanium 2 CPUs of Montecito type. The system is organized in 5 partitions with so-called high-density blades (4 cores per memory channel) and 13 partitions with so-called high-bandwidth blades (2 cores per memory channel).

The CPUs are interconnected by a NUMAlink 4 network with a hierarchical topology and a nominal bandwidth of 6.4 GB/s per CPU. Each processor is attached to 4 GB of local main memory, 512 kB of L1 data cache and 18 MB of L2 cache. One processor has a peak performance of 12.8 GFlops/s.

4.1.3 JuRoPA

The JuRoPA system at FZ Jülich consists of 2208 compute nodes, each with two Intel Xeon X5570 (Nehalem-EP) quad-core processors running at 2.93 GHz and 8 MB Intel Smart Cache. One node has a peak performance of 93.76 GFlops and is attached to 24GB of main memory. The system is connected via Infiniband QDR with non-blocking fat tree topology and has a peak performance of 207 TFlops.

4.2 Serial Performance of the LBM Code

When using the Intel compiler on HLRB 2, which is the common one on this system, a straight forward implementation of the LBM in waLBerla shows a relatively low serial performance of up to 0.98 MFlups on one core using array of structures layout, which can be seen from Table 1. To reduce main memory access, streaming and collision steps are merged into a single loop, which results in a performance of 1.42 MFlups. Rewriting the LBM stream collide core in C99 using variable length arrays simplifies the analysis of data dependencies and data streams for the compiler and reduces the number of integer calculations. This is crucial for achieving high performance on the IA-64 architecture, as shown in [STR08]. Changing the data layout of the LBM from AoS to SoA further improved the performance up to 5.8 MFlups on one single core. Additionally incorporating the boundary treatment in the stream collide loop enhances the performance to a maximum of 6.03 MFlups. Overall these tuning techniques have led to an improvement of the performance by more than a factor of 6.

Type of optimization	MFlups with AoS layout	MFlups with SoA layout
unfused	0.98	0.84
original (fused)	1.42	1.28
VLA	3.5	5.8
VLA opt BC	3.5	6.03

Table 1: Performance of serial LBM solver on HLRB 2 compiled with ICC with different data layouts and different type of optimizations.

4.3 Single Node Performance of the LBM Code

On the Woodcrest, the Intel and the GNU compiler produce nearly equivalently fast code and thus the differences in performance are quite low (see Figures 1 and 2). On this machine, the SoA data layout is preferable and results in better performance. But for domain sizes of 30, 70, 110 and 150 cells per direction the performance decreases, which could result from cache thrashing effects. When running in cache for a domain size of 10^3, the code delivers up to 23.87 MFlups on one node. When running out of cache, waLBerla obtains a maximum of 12.17 MFlups for a domain size of 160^3 cells with the Intel compiler and the SoA layout, which corresponds to a bandwidth usage of 6.38 GB/s. On the Woodcrest, the theoretical memory bandwidth of one node is 21.3 GB/s. However, the maximum achievable bandwidth is approximately 6.4 GB/s for the STREAM triad per node. Compared to the STREAM triad this results in 99.6% of the maximum achievable memory bandwidth for the waLBerla code neglecting cache effects.

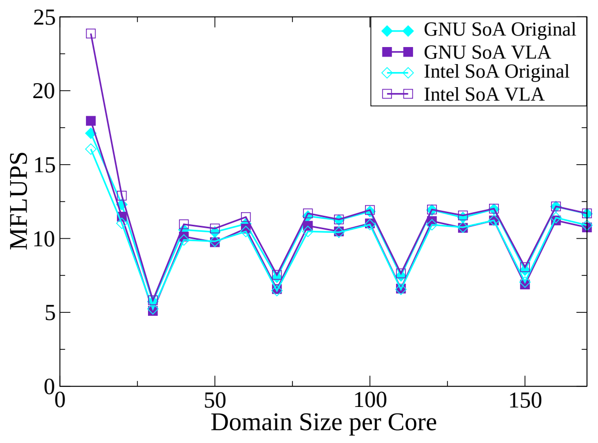

Figure 1: Single node performance on Woodcrest with structure of arrays layout and GNU and Intel compilers for different domain sizes.

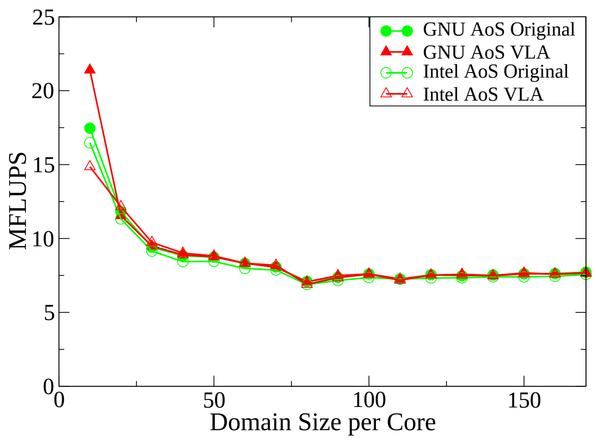

Figure 2: Single node performance on Woodcrest with array of structures layout and GNU and Intel compilers for different domain sizes.

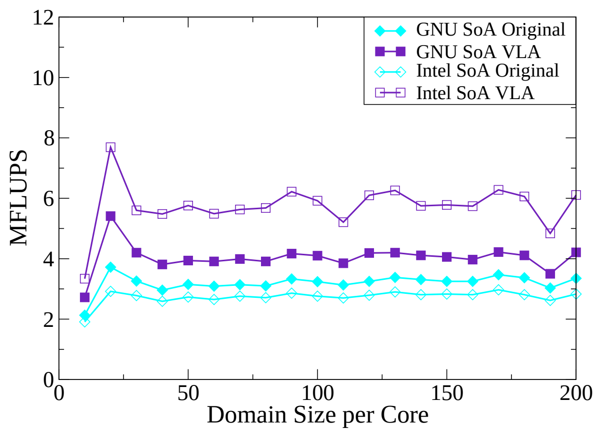

Figure 3: Single node performance on hlrb 2 with structure of arrays layout and GNU and Intel compilers for different domain sizes.

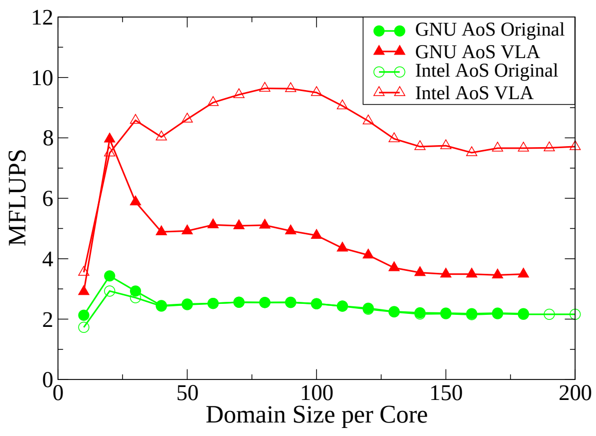

Figure 4: Single node performance on hlrb 2 with array of structures layout and GNU and Intel compilers for different domain sizes.

On the hlrb 2, the Intel compiler outperforms the GNU compiler using the variable length arrays by around 50% with the structure of arrays layout (see Figure 3). When running in cache, the maximum performance is 7.69 MFlups. The original code is comparatively slow with both compilers. The performance can be improved by holding the data in an array of structures, which is shown in Figure 4. Here the maximum performance of around 9.5 MFlups is reached when running domain sizes of 70^3 to 100^3 cells. Measurements of the hardware counters using the performance monitoring interface Perfmon [Per] show a high number of integer loads and integer operations for the original code with both compilers and data layouts. These result from index calculations, which are necessary for accessing the data in the data holding classes. As an example, on one core of the hlrb 2, the number of total loads and stores for the original code is around three times higher as the code using VLA. The number of floating point loads are nearly the same, but the number of integer loads are around five times higher. Additionally, the Itanium architecture is not optimized for integer arithmetics. All integer operations have explicitly be executed on the floating point unit, which is costly. Using the VLA constructs, the compiler can reduce the integer loads and integer operations significantly, which pays off by a factor of three in the performance comparing the most efficient original code with the best VLA variant. To reduce the overhead of integer operations in the original version, the basis index of the current cells can be precalculated and reused. Thereby the performance can be improved.

Measurements have shown a maximum achievable bandwidth of approximately 6.8 GB/s for the STREAM triad per node [Joh]. Comparing the highest LBM performance (which is 9.64 MFlups for a domain size of 80^3 and the VLA version with the AoS layout compiled with Intel) in terms of aggregate memory bandwidth to the STREAM triad, we maintain close to 75% of the peak STREAM triad bandwidth. In terms of peak GFlops our simulations achieve a still respectable 16%, considering that the LBM performance is typically limited by memory bandwidth.

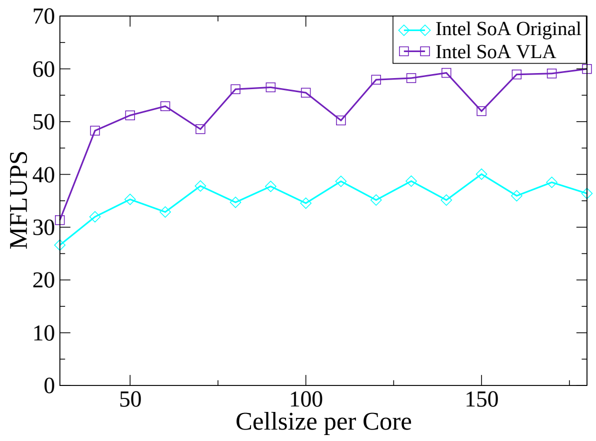

Figure 5: Single node performance on JuRoPA with structure of arrays layout and Intel compiler for different domain sizes.

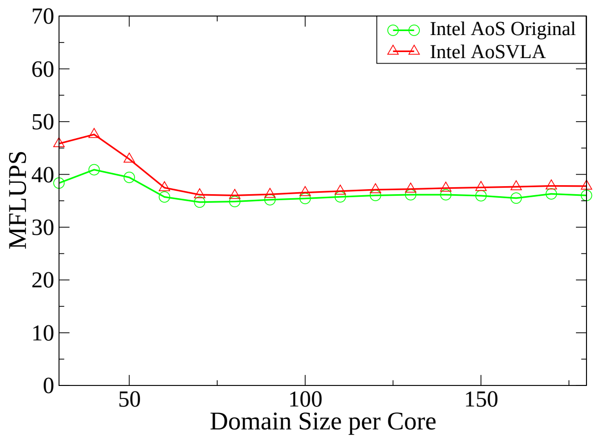

Figure 6: Single node performance on JuRoPA with array of structures layout and Intel compiler for different domain sizes.

On the JuRoPA again the SoA data layout is more efficient when running long loops, which can be seen from Figures 5 and 6. As on the Woodcrest, for domain sizes of 30, 70, 110 and 150 cells per direction, the performance decreases, which again could result from cache thrashing effects. For the SoA layout the VLA version is around 60% faster than the original version. Measurements of the hardware counters with the Likwid tool [Lik] again show many more integer operations for the original version compared to the VLA version, that come from address calculations. As on the hlrb 2 this decreases the performance significantly, but the problem can be adressed and reduced by precalculating and reusing the basis index of the current cell. The maximum achievable performance per node on this system is 59.98 MFlups when using SoA layout and the VLA version. This corresponds to 93.3% of the bandwidth compared to the STREAM triad.

A summary of all characteristics of the most efficient variant of the code on each machine is shown in Table 2. On the Woodcrest and the JuRoPA we maintain close to the maximum available memory bandwidth. Note that the Itanium node and the Nehalem node are well balanced in terms of usable memory bandwidth to peak performance. The Woodcrest processor on the other hand is not well balanced, which results in a low ratio of LBM performance to node peak performance.

	WOODCREST	HLRB 2	JUROPA
Domain size	160^3	80^3	180^3
Compiler	Intel	Intel	Intel
Data layout	SoA	AoS	SoA
STREAM memory bandwidth (GB/s)	6.4	6.8	33.67
LBM memory bandwidth (GB/s)	6.38	5.05	31.43
LBM bandwidth to STREAM bandwidth (in %)	99.6	74.3	93.3
Node peak (GFlop/s)	48	12.8	93.76
Performance LBM (GFlop/s)	2.59	2.05	12.78
LBM performance to node peak performance (in %)	5.4	16.04	13.63
STREAM bandwidth to node peak performance (GB/GFlop)	0.133	0.53	0.36

Table 2: Characteristics of most efficient version of the code on one node of the three different compute platforms.

4.4 Weak Scaling of the LBM Code

Figures 7 and 8 show the weak scaling of waLBerla on the three architectures each with the most efficient optimization and data layout with domain sizes of 100^3 and 140^3 per core respectively. A good scaling on all machines up to the maximum available number of nodes in the local queueing is achieved.

On the WOODCREST the SoA layout was the most efficient one. The maximum available number of cores is 256 in the standard queueing. Up to this number, the efficiency is still greater than 90% for both domain sizes.
For the HLRB 2 the simulations were performed with the AoS layout and the scaling is nearly linear up to 128 nodes, which are located in one rack and connected with 1.6 GB/s. For the jobs with 256 nodes, the slowest connection is 0.8 GB/s, which shows off in a decrease of efficiency in the graph. The simulations with 512 and 1024 nodes only have a network bandwidth of 0.4 GB/s and 0.2 GB/s respectively due to the fat tree topology, which again decreases the overall performance. But for 2048 MPI processes, the efficiency is still respectable 90% for the domain size of 140^3.

On the JUROPA the SoA layout was used and the scaling is nearly linear up to 64 nodes, which correspond to 512 cores. Then the efficiency decreases statically, which indicates a lower network bandwidth beyond 512 cores. Using 4096 cores (512 nodes), the efficiency is 68%. Note that the simulation on 512 nodes with a domain size of 140^3 is missing. This is due to memory problems when using many MPI processes. On this machine the MPI system allocates at runtime an amount of memory for each connection. This results in lack of memory for the user data and the program aborts. It can be fixed by switching off the automatic allocation of the MPI system. This will be investigated in the future.

5 Conclusion

This report shows that high performance can be obtained on current supercomputing platform from a huge C++ software framework like waLBerla, which is built to supply various applica-

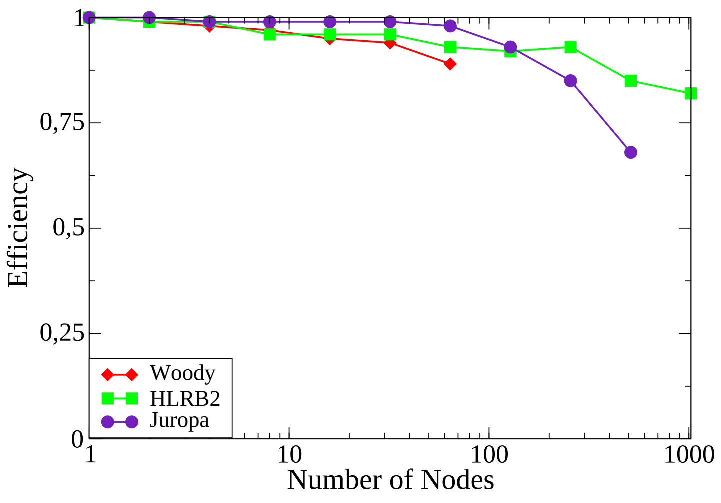

Figure 7: Weak scaling on WOODCREST, HLRB 2 and JUROPA with 100^3 cells per core using the most efficient data layout and the Intel compiler.

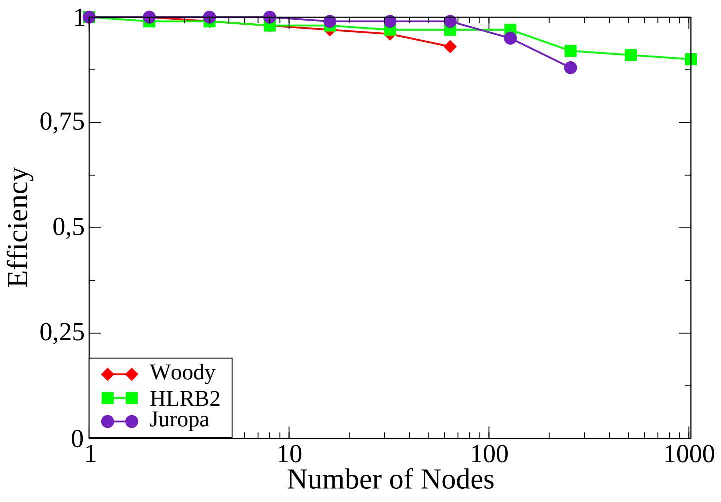

Figure 8: Weak scaling on WOODCREST, HLRB 2 and JUROPA with 140^3 cells per core using the most efficient data layout and the Intel compiler.

tions. However, optimizations of the kernel routine can improve the performance tremendously, which especially holds for the Itanium architecture where a reduction of integer calculations improves the efficiency by a factor of three. But also the latest processors from Intel with Nehalem core suffer from the overhead of index calculations and benefit from using the variable length array optimization. Here the performance improvements are still in the order of 50%. Using the appropriate data layout can additionally enhance the speed of the simulations. waLBerla supports both, choosing the most efficient data layout and kernel optimizations. The weak scaling on all platforms show good results up to 4096 cores, which illustrates the ability of the software to run large parallel simulations on current supercomputers.

In a next step, the performance and scaling of waLBerla on other platforms, like the Cray XT4 and the BlueGene, will be evaluated and the code will be optimized to support these machines efficiently.

Acknowledgments: The work was supported by the Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern (KONWIHR) under project waLBerlaMC and the Bundesministerium für Bildung und Forschung under the SKALB project no. 01IH08003A.

We would like to thank for their support: Thomas Zeiser, Jan Treibig, Johannes Habich and Gerhard Wellein, Regional Computing Center of Erlangen (RRZE), Erlangen, Germany; Hans-Georg Kleinhenz, Reinhold Bader and Matthias Brehm, Leibniz-Rechenzentrum (LRZ), Garching, Germany; Stefanie Meier, Jülich Supercomputing Centre (JSC) , Germany; Markus Stürmer, System Simulation Group, Erlangen.

References

[BGK54] P. L. Bhatnagar, E. P. Gross, and M. Krook, *A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems*, Phys. Rev. **94** (1954), no. 3, 511–525.

[CD98] S. Chen and G. D. Doolen, *Lattice Boltzmann method for fluid flows*, Annu. Rev. Fluid Mech. **30** (1998), 329–364.

[DIW+08] S. Donath, K. Iglberger, G. Wellein, T. Zeiser, and A. Nitsure, *Performance Comparison of Different Parallel Lattice Boltzmann Implementations on Multi-core Multi-socket Systems*, Int. J. Sci. Eng. **4** (2008), no. 1, 3–11.

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

[FS07] M. Frigo and V. Strumpen, *The memory behavior of cache oblivious stencil computations*, J. Supercomput. **39** (2007), no. 2, 93–112.

[HL97] X. He and L.-S. Luo, *Lattice Boltzmann model for the incompressible Navier-Stokes equation*, J. Stat. Phys. **88** (1997), 927–944.

[HLR] *Information on the HLRB 2*, http://www.lrz-muenchen.de/services/compute/hlrb/.

[Igl03] K. Iglberger, *Performance Analysis and Optimization of the Lattice Boltzmann Method in 3D*, Master's thesis, Computer Science Department 10 (System Simulation), University of Erlangen-Nuremberg, 2003.

[Joh] *Johannes Habich, Regionales Rechenzentrum Erlangen (RRZE)*, Private communication.

[Jur] *Information on the JuRoPA*, http://www.fz-juelich.de/jsc/juropa/.

[KPR$^+$05] C. Körner, T. Pohl, U. Rüde, N. Thürey, and T. Zeiser, *Parallel Lattice Boltzmann Methods for CFD Applications*, Numerical Solution of Partial Differential Equations on Parallel Computers (A.M. Bruaset and A. Tveito, eds.), Lecture Notes for Computational Science and Engineering, vol. 51, Springer, 2005, pp. 439–465.

[Lik] *Information on Likwid*, http://code.google.com/p/likwid/.

[McC08] J. D. McCalpin, *STREAM: Sustainable memory bandwidth in high performance computers*, http://www.cs.virginia.edu/stream/, 1991-2008.

[Mey01a] R. Meyers, *The New C: Declarations and Initializations*, C/C++ Users Journal, 2001.

[Mey01b] ______, *The New C: Why Variable Length Arrays?*, C/C++ Users Journal, 2001.

[Mey02] ______, *The New C: VLA typedefs and Flexible Array Members*, C/C++ Users Journal, 2002.

[MSYL02] R. Mei, W. Shyy, D. Yu, and L.-S. Luo, *Lattice Boltzmann Method for 3-D Flows with Curved Boundary*, J. Comput. Phys. **161** (2002), 680–699.

[Per] *Information on Perfmon*, http://perfmon2.sourceforge.net/.

[PKW$^+$03] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rüde, *Optimization and Profiling of the Cache Performance of Parallel Lattice Boltzmann Codes*, Parallel Process. Lett. **13** (2003), no. 4, 549–560.

[PTD$^+$04] T. Pohl, N. Thürey, F. Deserno, U. Rüde, P. Lammers, G. Wellein, and T. Zeiser, *Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures*, 2004, Supercomputing Conference 04.

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

[STR08] M. Stürmer, J. Treibig, and U. Rüde, *Optimising a 3D multigrid algorithm for the IA-64 architecture*, Int. J. Comp. Sci. Eng. **4** (2008), no. 1.

[Suc01] S. Succi, *The Lattice Boltzmann Equation - For Fluid Dynamics and Beyond*, Clarendon Press, 2001.

[WCO$^+$08] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick, *Lattice boltzmann simulation optimization on leading multicore platforms*, Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on (2008), 1–14.

[Woo] *Information on the Woodcrest Cluster*, http://www.rrze.uni-erlangen.de/dienste/arbeiten-rechnen/hpc/systeme/woodcrest-cluster.shtml.

[WZDH05] G. Wellein, T. Zeiser, S. Donath, and G. Hager, *On the single processor performance of simple lattice Boltzmann kernels*, Computer & Fluids **35** (2005), no. 8–9, 910–919.

[ZGS08] T. Zeiser, J. Götz, and M. Stürmer, *On Performance and Accuracy of Lattice Boltzmann Approaches for Single Phase Flow in Porous Media: A Toy Became an Accepted Tool - How to Maintain Its Features Despite More and More Complex (Physical) Models and Changing Trends in High Performance Computing!?*, Computational Science and High Performance Computing III, Springer, 2008, pp. 165–183.