

# Towards Optimal Performance for Lattice Boltzmann Applications on Terascale Computers

G. Wellein<sup>a</sup>, P. Lammers<sup>b</sup>, G. Hager<sup>a</sup>, S. Donath<sup>a</sup>, and T. Zeiser<sup>a</sup>

<sup>a</sup>Regionales Rechenzentrum Erlangen (RRZE)  
Martensstraße 1, D-91058 Erlangen, Germany

<sup>b</sup>Höchstleistungsrechenzentrum Stuttgart (HLRS)  
Allmandring 30, D-70550 Stuttgart, Germany

On popular cluster computers the performance of many CFD applications can fall far short of the impressive peak performance numbers. Using a large scale LBM application, we demonstrate the different performance characteristics of modern supercomputers. Classical vector systems (NEC SX8) still combine excellent performance with a well established optimization approach and can break the TFlop/s application performance barrier at a very low processor count. Although, modern microprocessors offer impressive peak performance numbers and extensive code tuning has been performed, they only achieve 10% or less of the vector performance for the LBM application. Clusters try to fill this gap by massive parallelization but network capabilities can impose severe performance restrictions if the problem size is not arbitrarily scalable. Tailored HPC servers like the SGI Altix series can relieve these restrictions.

## 1. Introduction

In the past decade the lattice Boltzmann method (LBM) [1–3] has been established as an alternative for the numerical simulation of (time-dependent) incompressible flows. One major reason for the success of LBM is that the simplicity of its core algorithm allows both easy adaption to complex application scenarios as well as extension to additional physical or chemical effects. Since LBM is a direct method, the use of extensive computer resources is often mandatory. Thus, LBM has attracted a lot of attention in the High Performance Computing community [4–6]. An important feature of many LBM codes is that the core algorithm can be reduced to a few manageable subroutines, facilitating deep performance analysis followed by precise code and data layout optimization [6–8]. Extensive performance evaluation and optimization of LBM for terascale computer architectures is required to find an appropriate computer architecture for the various LBM applications and to make best use of computer resources. It also allows us to better understand the performance characteristics of modern computer architectures and to develop appropriate optimization strategies which can also be applied to other applications.

In this report we discuss LBM performance on commodity “off-the-shelf” (COTS) clusters (GBit, Infiniband) with Intel Xeon processors, tailored HPC systems (Cray XD1,

SGI Altix) and a NEC SX8 vector system. We briefly describe the main architectural differences and comment on single processor performance as well as optimization strategies. Finally, we evaluate and present the parallel performance of a large scale simulation running on up to 2000 processors, providing 2 TFlop/s of sustained performance. For those studies we use the LBM solver *BEST* (Boltzmann Equation Solver Tool), which is written in FORTRAN90 and parallelized with MPI using domain decomposition [9]. As a test case we run simulations of flow in a long channel with square cross-section, which is a typical application in turbulence research.

## 2. Basics of the Lattice Boltzmann Method

The widely used class of lattice Boltzmann models with BGK approximation of the collision process [1–3] is based on the evolution equation

$$f_i(\vec{x} + \vec{e}_i \delta t, t + \delta t) = f_i(\vec{x}, t) - \frac{1}{\tau} [f_i(\vec{x}, t) - f_i^{\text{eq}}(\rho, \vec{u})], \quad i = 0 \dots N. \quad (1)$$

Here,  $f_i$  denotes the particle distribution function which represents the fraction of particles located in timestep  $t$  at position  $\vec{x}$  and moving with the microscopic velocity  $\vec{e}_i$ . The relaxation time  $\tau$  determines the rate of approach to local equilibrium and is related to the kinematic viscosity of the fluid. The equilibrium state  $f_i^{\text{eq}}$  itself is a low Mach number approximation of the Maxwell-Boltzmann equilibrium distribution function. It depends only on the macroscopic values of the fluid density  $\rho$  and the flow velocity  $\vec{u}$ . Both can be easily obtained as the first moments of the particle distribution function.

The discrete velocity vectors  $\vec{e}_i$  arise from the  $N + 1$  chosen collocation points of the velocity-discrete Boltzmann equation and determine the basic structure of the numerical grid. We chose the D3Q19 model [1] for the discretization in 3-D, which uses 19 discrete velocities (collocation points) and provides a computational domain with equidistant Cartesian cells (voxels). Each timestep ( $t \rightarrow t + \delta t$ ) consists of the following steps which are repeated for all cells:

- Calculation of the local macroscopic flow quantities  $\rho$  and  $\vec{u}$  from the distribution functions,  $\rho = \sum_{i=0}^N f_i$  and  $\vec{u} = \frac{1}{\rho} \sum_{i=0}^N f_i \vec{e}_i$ .
- Calculation of the equilibrium distribution  $f_i^{\text{eq}}$  from the macroscopic flow quantities (see [1] for the equation and parameters) and execution of the “collision” (relaxation) process,  $f_i^*(\vec{x}, t^*) = f_i(\vec{x}, t) - \frac{1}{\tau} [f_i(\vec{x}, t) - f_i^{\text{eq}}(\rho, \vec{u})]$ , where the superscript \* denotes the post-collision state.
- “Propagation” of the  $i = 0 \dots N$  post-collision states  $f_i^*(\vec{x}, t^*)$  to the appropriate neighboring cells according to the direction of  $\vec{e}_i$ , resulting in  $f_i(\vec{x} + \vec{e}_i \delta t, t + \delta t)$ , i.e. the values of the next timestep.

The first two steps are computationally intensive but involve only values of the local node while the third step is just a direction-dependent uniform shift of data in memory. A fourth step, the so called “bounce-back” rule [2,3], is incorporated as an additional routine and “reflects” the distribution functions at the interface between fluid and solid cells, resulting in an approximate no-slip boundary condition at walls.

### 3. Implementation and Optimization Strategies

The basic implementation of our code follows the guidelines described in [7]. For a better understanding of the subsequent performance evaluation we briefly highlight some important aspects:

- Removing contributions with zero components of  $\vec{e}_i$  and precomputing common subexpressions results in roughly  $F = 200$  floating point operations (Flops) per lattice site update. The actual number can vary because of compiler optimizations.
- Collision and propagation steps are done in the same loop (“collision-propagation”), which reduces the data transfer between main memory and processor for one time step to one read of the whole distribution function at time  $t$  and one store of the whole distribution function at time  $t^*$ . Thus,  $B = 2 \times 19 \times 8$  Bytes have to be transferred per lattice site update, at first sight.
- Two arrays are used which hold data of successive timesteps  $t$  and  $t + 1$ . The data layout was chosen as  $f(x, y, z, i, t)$  with the first index addressing consecutive memory locations due to Fortran’s column major order making spatial blocking techniques dispensable [7].

The achieved performance for this implementation on IA32 and IA64 systems was, however, only 25–45% of the theoretical maximum set by memory bandwidth. Inspection of compiler reports and performance counters has revealed two additional bottlenecks, which have not been addressed by the optimizations discussed so far:

- The large loop body prevents the Intel IA64 compiler from software pipelining, which is essential for Itanium 2, and causes massive register spills on all architectures.
- Concurrently writing to 19 different cache lines interferes with the limited number of write combine buffers on IA32 architectures (6 for Intel Xeon/Nocona).

In order to remove these problems the innermost loop (in  $x$ -direction) of our “collision-propagate” step is split up into 3 loops of length  $N_x$ , which equals the domain size in  $x$ -direction. However, additional auxiliary arrays of length  $N_x$  are required which hold intermediate results. As long as  $N_x$  is not too large ( $N_x < 1000$ ), these arrays can be kept in the on-chip caches and accessed quickly. This implementation increases the number of floating point operations per lattice site update ( $B \approx 230 - 250$ ) but usually provides best performance (see next section). Note that for vector systems no additional optimizations besides vectorization, are required and the number of floating point operations per lattice site update is rather small ( $B \approx 170$ ).

### 4. Single Node Specification and Performance

Most up-to-date HPC systems are configured as shared-memory nodes (e.g. dual Xeon) connected by an high speed interconnect (e.g. GBit or Infiniband) or a proprietary network (e.g. NUMALink in SGI systems). In order to clearly separate the contribution of processor and interconnect to the total performance of a terascale system, we first turn to the single node performance characteristics.

Table 1

Single processor specifications. Peak performance, maximum memory bandwidth and sizes of the largest on-chip cache levels are given in columns 2-4. Columns 5,6 show the maximum MLUPS rate as limited by peak performance or memory bandwidth. The last two columns contain performance achieved for original and tuned implementations at a domain size of  $128^3$ .

Platform	CPU specs			Limits		Versions	
	Peak GFlop/s	MemBW GB/s	Cache MB	Peak MLUPS	MemBW MLUPS	Orig. MLUPS	Tuned MLUPS
Intel Xeon DP (3.4 GHz)	6.8	5.3	1.0 (L2)	34.0	11.8	4.5	4.9
AMD Opteron848 (2.2 GHz)	4.4	6.4	1.0 (L2)	22.0	14.0	2.9	4.7
Intel Itanium 2 (1.4 GHz)	5.6	6.4	1.5 (L3)	28.0	14.0	7.9	8.5
NEC SX8 (2 GHz)	16.0	64.0	—	80.0	210.0	68.0	—

Since a lot of optimization was done at the CPU level, we have built a benchmark kernel including the routines for “collision-propagation” and “bounce-back” which mainly determine the performance of the LBM step. In order to test for limitations of the single node performance, a shared-memory parallelization using OpenMP was implemented as well.

#### 4.1. Architectural Specifications

In the left part of Table 1 we briefly sketch the most important single processor specifications<sup>1</sup> of the architectures examined. Concerning the memory architecture of COTS systems, we find a clear tendency towards large on-chip caches which run at processor speed, providing high bandwidth and low latency. The NEC vector system incorporates a different memory hierarchy and achieves substantially higher single processor peak performance and memory bandwidth.

##### Intel Xeon/Nocona and AMD Opteron

The Intel Xeon/Nocona and the AMD Opteron processors used in our benchmarks are 64-bit enabled versions of the well-known Intel Xeon and AMD Athlon designs, respectively, but maintain full IA32 compatibility. Both are capable of performing a maximum of two double precision floating point (FP) operations (one multiply and one add) per cycle. The most important difference is that in standard multi-processor configurations (2- or 4-way are in common use) all processors of the Intel based designs have to share one memory bus while in AMD based systems the aggregate memory bandwidth scales with processor count. The benchmarks were compiled using the Intel Fortran Compiler for Intel EM64T-based applications in version 8.1.023 and were run on a 2-way Intel and 4-way AMD system. The parallel benchmarks were run on a cluster with Intel Xeon nodes featuring both GBit and Infiniband interconnect. As an Opteron based parallel system

<sup>1</sup>The Intel Xeon processor supports a maximum bandwidth of 6.4 GB/s; the benchmark system was equipped with DDR-333 memory providing a bandwidth of 5.3 GB/s only.

we have chosen the Cray XD1 which uses a proprietary network (RapidArray) to connect 2-way Opteron nodes.

### Intel Itanium 2

The Intel Itanium 2 processor is a superscalar 64-bit CPU using the Explicitly Parallel Instruction Computing (EPIC) paradigm. The Itanium concept does not require any out-of-order execution hardware support but demands high quality compilers to identify instruction level parallelism at compile time. Today clock frequencies of up to 1.6 GHz and on-chip L3 cache sizes from 1.5 to 9 MB are available. Two Multiply-Add units are fed by a large set of 128 floating point (FP) registers, which is another important difference to standard microprocessors with typically 32 FP registers. The basic building blocks of systems used in scientific computing are 2-way nodes (e.g. SGI Altix, HP rx2600) sharing one bus with 6.4 GByte/s memory bandwidth.

The system of choice in this section is a 2-way HP zx6000 with 1.4 GHz CPUs. For compilation we used Intel's Fortran Itanium Compiler V9.0. The parallel benchmarks were run on SGI Altix systems with NUMALink interconnect and 1.5 GHz or 1.6 GHz CPUs but with the same memory bandwidth per CPU.

### NEC SX8

From a programmers' view, the NEC SX8 is a traditional vector processor with 4-track vector pipes running at 2 GHz. One multiply and one add instruction per cycle can be executed by the arithmetic pipes delivering a theoretical peak performance of 16 GFlop/s. The memory bandwidth of 64 GByte/s allows for one load or store per multiply-add instruction, providing a balance of 0.5 Word/Flop. The processor features 64 vector registers, each holding 256 64-bit words. Basic changes compared to its predecessor systems, as used e.g. in the Earth Simulator, are a separate hardware square root/divide unit and a "memory cache" which lifts stride 2 memory access patterns to the same performance as contiguous memory access. An SMP node comprises eight processors and provides a total memory bandwidth of 512 GByte/s, i. e. the aggregated single processor bandwidths can be saturated.

The benchmark results presented in this paper were measured on a 72-node NEC SX8 at the High Performance Computing Center Stuttgart (HLRS).

## 4.2. Single Node Performance: Theory and Reality

Before looking at the "pure" performance measurements an estimation of the maximum achievable performance numbers for the platforms under consideration may be helpful to understand the "experiments". All performance numbers are given in MLUPS (**M**ega **L**attice **S**ite **U**pdates **p**er **S**econd), which is a handy unit for measuring the performance of LBM.<sup>2</sup> In this section we report the update rate of the fluid cells only, i.e. we do not count obstacle cells at the boundaries where no computation is being done. Based on characteristic quantities of the benchmarked architectures (cf. Table 1), an estimate for a theoretical performance limit can be given. Performance is either limited by available memory bandwidth or peak performance. Therefore, the attainable maximum performance is either given as

---

<sup>2</sup>Note that 5 MLUPS are roughly equal to 1 GFlop/s of sustained performance.

Table 2

Single node performance for optimized LBM implementation using OpenMP parallelization and a fixed domain size of  $128^3$ . Arrays include one additional ghost layer in each direction, making cache trashing effects [7] negligible. For AMD, performance for naive and NUMA-aware OpenMP implementations are shown.

Platform	MLUPS		
	1 CPU	2 CPUs	4 CPUs
AMD Opteron - naive	4.6	5.8	5.8
AMD Opteron - NUMA	4.6	9.2	17.6
Intel Xeon	4.8	5.5	—
Intel Itanium 2	8.4	8.9	—

$$P = \frac{\text{MemBW}}{B} \quad \text{or} \quad P = \frac{\text{Peak Perf.}}{F} \quad (2)$$

The values of  $P$  are given in units of MLUPS, since  $B$  is the number of bytes per lattice site update to be transferred between CPU and main memory and  $F$  is the number of floating point operations to be performed per lattice site update. For both quantities, a typical number arising from the LBM implementation itself has been given in Section 3. As a peculiarity of most cache based architectures, read for ownership (RFO) transfers on write misses must be taken into account. In our implementation, this occurs when the updated values are propagated to adjacent cells. The corresponding cache line must be loaded from main memory before it can be modified and written back. Consequently, the value of  $B$  increases by 50%, further reducing the effective available bandwidth. The performance limits imposed by hardware and the performance numbers measured both for original and optimized implementations (additional inner loop splitting as described in Section 3) are presented in Table 1 for a computational domain size of  $128^3$ . A substantial effect of the additional optimization can be seen on the AMD system which is lifted from a poor performance level to the same performance as its Intel competitor.

The striking features of the vector machine are extremely high performance and very efficient use of the processor, e.g. our LBM code achieves 85% of the algorithm’s theoretical limit. Note that this limit is not set by the memory subsystem but by the peak performance of the processor.

The scalability of our simple LBM kernel using OpenMP within one node is depicted in Table 2. Although the Opteron architecture provides a separate path to the main memory for each processor, a naive implementation with “collision-propagation” and “bounce-back” routines being the only parallelized parts, shows poor scalability. The Opteron design is not a symmetric multiprocessing (SMP) node with flat memory (as the Intel systems are), but implements a Cache Coherent Non Uniform Memory (ccNUMA) architecture. Each processor has its own local memory and can access data on remote nodes through a network at lower bandwidth and higher latencies. For that reason, data placement becomes a major issue on ccNUMA machines. Since memory pages are mapped on the node of the processor which initializes the data (first-touch placement), all data is put into the memory of a single processor if the initialization routine is not parallelized. A

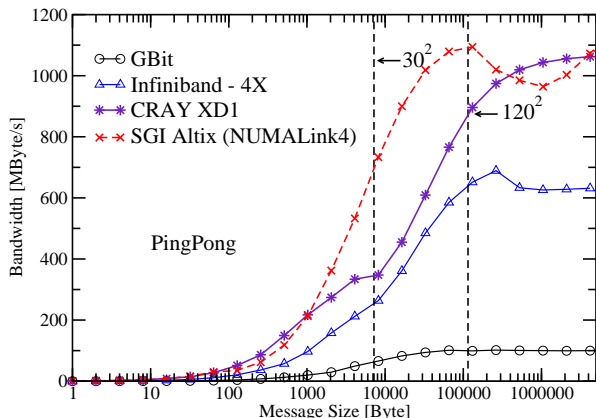


Figure 1. MPI bandwidth measured with PMB. Vertical lines denote message sizes with  $30^2$  and  $120^2$  double words.

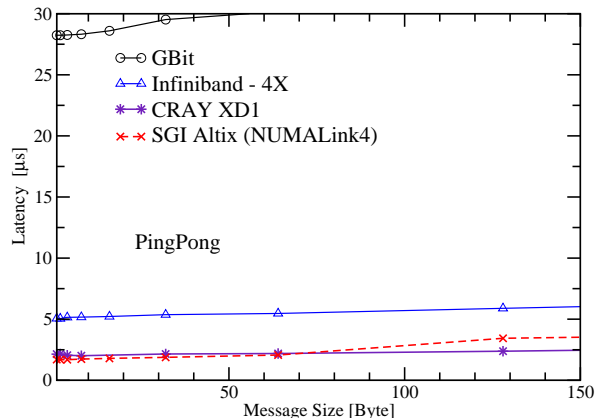


Figure 2. MPI latency (total time for message transfer) at small message sizes as measured with PMB.

NUMA-aware implementation uses parallelized data initialization in a manner compatible with the computational kernels and achieves very good scalability on the Opteron system, leading to a parallel efficiency of roughly 96% on 4 processors.<sup>3</sup>

Intel systems, on the other hand, do not benefit from parallel initialization as the memory model is flat. Consequently, scalability is poor due to the memory bottleneck, and gets even worse on 4-way SMP systems which we did not include in our evaluation.

## 5. Scalability and Parallel Performance

The lessons learned from [7] and Section 3 have been used to optimize the MPI parallel LBM application *BEST*. Because of additional program overhead, *BEST* performance may of course slightly deviate from the numbers presented so far.

Scalability beyond one node is determined by the ability of the network to meet the application's communication requirements. In our scalability analysis of *BEST*, we use several very different interconnects ranging from standard GBit to the NEC IXS. A good insight into basic network capabilities is provided by the Pallas MPI Benchmark (PMB, available as part of the Intel Cluster Toolkit). We have chosen the PingPong test to measure round trip times for sending messages between two arbitrary nodes. Bandwidth and latency (one way time) as a function of the message length are depicted in Figure 1 and Figure 2, respectively. Obviously, rather large message lengths of roughly 100 KBytes are required to get maximum bandwidth on all interconnects presented. Typical message sizes arising from simple domain decompositions of LBM on clusters (cf. vertical lines in Figure 1) can be too short to get maximum bandwidth. While GBit is not competitive in both categories, the Infiniband (IB) solution does much better but is still a factor of 2 behind the two proprietary interconnects used in the SGI Altix (NUMALink4) and the Cray XD1 (RapidArray). Although, the Cray XD1 technology seems to be related to IB there is a substantial technical difference which limits the capabilities of standard IB solutions as used in most clusters: The IB card is connected via an external PCI-X

<sup>3</sup>Parallel efficiency is defined as  $P(n)/(n \times P(1))$ , where  $P(n)$  is the performance on  $n$  CPUs.

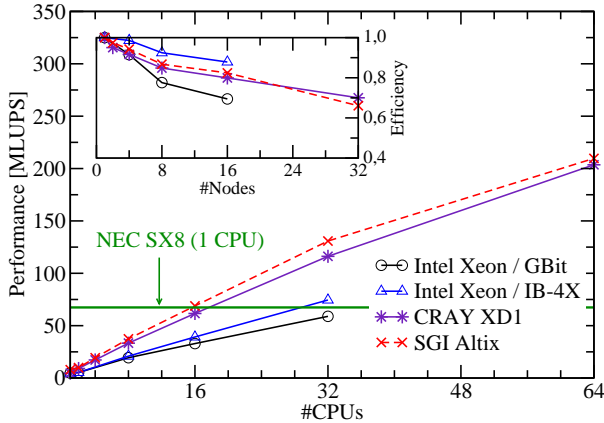


Figure 3. Strong scaling performance and parallel efficiency (inset) at a domain size of  $256 \times 128^2$ . Horizontal bold line denotes NEC SX8 single CPU performance.

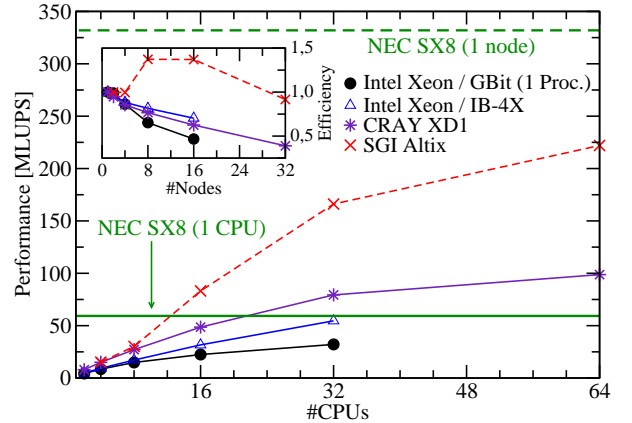


Figure 4. Strong scaling performance and parallel efficiency (inset) with a domain size of  $64^3$ . Horizontal bold (dashed) line denotes NEC SX8 single CPU (node) performance.

interface to the compute node. The RapidArray chip, on the other hand, is directly connected to the HyperTransport channel of one Opteron processor, avoiding additional overhead and limitations of PCI-X. For comparison, the IXS interconnect of the NEC SX8 achieves up to 28.75 GByte/s bandwidth and 3-7  $\mu$ s small message latency between two nodes [10].

For a final scalability analysis one must differentiate between *weak scaling* and *strong scaling* scenarios. In *weak-scaling* experiments the total problem size per processor is kept constant, i.e. total domain size increases linearly with the number of CPUs used. Consequently, the ratio of communication and computation remains constant and one can achieve a perfectly linear increase in total performance even on GBit [5] provided the domain size per node is large enough and the network capabilities scale with processor count. A typical area for the weak scaling LBM scenario is basic turbulence research, where the total domain size should be as large as possible.

In *strong scaling* experiments the total domain size remains constant, independent of CPU number. Here, scalability is dominated by interconnect features and two effects must be considered: First, with increasing processor count the domain size per processor decreases, raising the ratio of communication ( $\propto$  local domain surface) versus computation ( $\propto$  volume of the local domain). Second, smaller surface size is equivalent to smaller message size leading to substantially worse network bandwidth for small domains (cf. discussion of Figure 1). For large processor counts runtime will thus mainly be governed by communication and performance saturates, i.e. scalability breaks down. Strong scaling is a major issue if turnaround time for a fixed problem is important, e.g. for practical engineering problems or if computations would run for months.

In Figures 3 and 4 we present strong scaling experiments for two typical domain sizes with  $4.2 \times 10^6$  and  $0.26 \times 10^6$  lattice sites. Note that the baseline for parallel efficiency as given in the insets is the one-node performance  $P(2)$  in order to eliminate the different intranode performance characteristics. The deficiencies of GBit for decreasing message

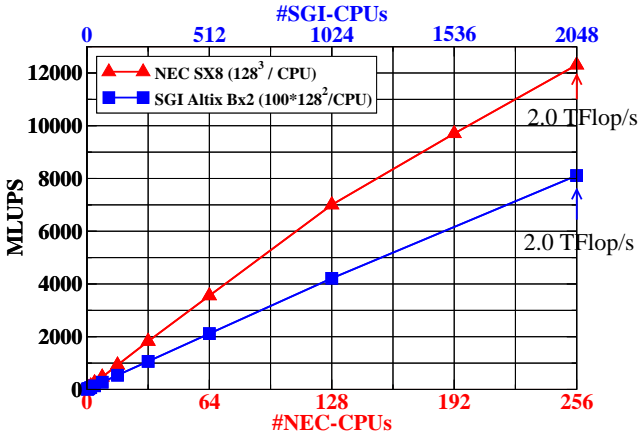


Figure 5. Weak scaling performance of *BEST* on SGI Altix Bx2 and NEC SX8. The number of NEC SX8 (SGI Altix Bx2) CPUs is given on the lower (upper) x-axis. The total domain size for the largest run was  $0.5 \times 10^9$  ( $3.3 \times 10^9$ ) on NEC SX8 (SGI Altix Bx2).

sizes become quite obvious. In both cases the GBit cluster with 32 CPUs (16 nodes) could not match the performance of a single vector CPU. Even more, in Figure 4 we only show results for one CPU per node because total performance drops significantly if both CPUs are used. The IB network provides higher total performance and better parallel efficiency, the latter even larger than on the Cray XD1, which is at first sight an inconsistency with our findings for the PMB benchmark. However, the substantially lower single node performance of the Intel Xeon node ( $\approx 5.5$  MLUPS as compared to  $\approx 9.2$  MLUPS for 2 Opteron processors) reduces the fraction of total time spent for communication and thus allows the slower network to “scale better”, albeit at lower total performance. A peculiarity of systems with large caches is presented in Figure 4 for an SGI Altix with Itanium 2 processors having 6 MB of L3 cache each. At and beyond 16 CPUs, the aggregate cache size is large enough to hold the whole data set, speeding up the computation part substantially as compared to the baseline result on two processors. This effect in combination with the fast NUMALink network allows us to achieve a parallel efficiency much larger than 1 and to get an even higher performance on 64 CPUs than for the larger domain used in Figure 3.

Concerning single CPU and single node performance, the NEC SX8 system is a class of its own for both domain sizes. Note that the NEC SX8 single node performance for Figure 3 is 435 MLUPS which exceeds the maximum y-scale of this figure by more than 50%.

In a final weak scaling experiment we have chosen NEC SX8 and SGI Altix Bx2 systems in Figure 5 to demonstrate that the *BEST* application is able to break the TFlop/s barrier on modern HPC systems. Both systems scale very well to a maximum sustained performance of roughly 2 TFlop/s for the largest simulations. However, the equivalent MLUPS rate, which is the important performance number for LBM applications, is 50% higher on the vector machine due to its computationally less intensive implementation (cf. discussion in Section 3). The striking feature of the vector system is, of course, that less than a tenth of the processors is required to reach the same performance level as on the SGI system. On clusters this ratio will be even worse. With only 128 processors the NEC machine achieves more than one TFlop/s of sustained application performance which is more than 50% of its peak.

## 6. Conclusion

The main focus of our report was to discuss single processor optimization strategies and scalability issues for large scale LBM applications. Although a lot of work was put into optimization, cache based microprocessors achieve only a fraction of the performance of a modern NEC SX8 vector CPU. In contrast to the microprocessors, LBM performance is not limited by the memory subsystem on vector systems. For our turbulence application code *BEST* we achieved a sustained performance of 2 TFlop/s on 256 NEC vector CPUs. Even though the SGI Altix system is best in the microprocessor class we require more than 2000 processors on this architecture to reach a comparable performance level.

## Acknowledgments

Part of this work is financially supported by the Competence Network for Technical, Scientific High Performance Computing in Bavaria KONWIHR and the High performance computer competence center Baden-Wuerttemberg. We gratefully acknowledge the support of R. Wolff (SGI), who provided access to the SGI Altix Bx2 and gave many helpful hints to make best use of the system. Benchmarks were also run on the SGI Altix system at CSAR Manchester.

## REFERENCES

1. Y. H. Qian, D. d’Humières, P. Lallemand, Lattice BGK models for Navier-Stokes equation, *Europhys. Lett.* 17 (6) (1992) 479–484.
2. D. A. Wolf-Gladrow, Lattice-Gas Cellular Automata and Lattice Boltzmann Models, Vol. 1725 of Lecture Notes in Mathematics, Springer, Berlin, 2000.
3. S. Succi, *The Lattice Boltzmann Equation – For Fluid Dynamics and Beyond*, Clarendon Press, 2001.
4. L. Oliker, J. C. A. Canning, J. Shalf, S. Ethier, Scientific computations on modern parallel vector systems, in: *Proceedings of SC2004*, CD-ROM, 2004.
5. T. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers, G. Wellein, T. Zeiser, Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures, in: *Proceedings of SC2004*, CD-ROM, 2004.
6. F. Massaioli, G. Amati, Achieving high performance in a LBM code using OpenMP, in: *EWOMP’02*, Roma, Italy, 2002.
7. G. Wellein, T. Zeiser, S. Donath, G. Hager, On the single processor performance of simple lattice Boltzmann kernels, *Computers & Fluids*.
8. T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, U. Rüde, Optimization and profiling of the cache performance of parallel lattice Boltzmann codes, *Par. Proc. Lett.* 13 (4) (2003) 549–560.
9. P. Lammers, K. Beronov, T. Zeiser, F. Durst, Testing of closure assumptions for fully developed turbulence channel flow with the aid of a lattice Boltzmann simulation, in: S. Wagner, W. Hanke, A. Bode, F. Durst (Eds.), *High Performance Computing in Science and Engineering, Munich 2004. Transactions of the Second Joint HLRB and KONWIHR Result and Reviewing Workshop*, Springer Verlag, 2004, pp. 77–92.
10. [http://icl.cs.utk.edu/hpcc/hpcc\\_results\\_lat\\_band.cgi?display=combo](http://icl.cs.utk.edu/hpcc/hpcc_results_lat_band.cgi?display=combo).