Computers and Mathematics with Applications 58 (2009) 1062–1070

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

Fluid flow simulation on the Cell Broadband Engine using the lattice Boltzmann method

Markus Stürmer [a,*], Jan Götz [a], Gregor Richter [b], Arnd Dörfler [b], Ulrich Rüde [a]

[a] Department of Computer Science 10 – System Simulation, University of Erlangen-Nuremberg, 91058 Erlangen, Germany

[b] Department of Neuroradiology, University of Erlangen-Nuremberg, 91054 Erlangen, Germany

ARTICLE INFO

Keywords:
Cell processor
CBEA
Blood flow
Lattice Boltzmann
Hemodynamics
Aneurysm
High performance computing

ABSTRACT

In this paper we present a fast lattice Boltzmann fluid solver that has been performance optimized and tailored for the Cell Broadband Engine Architecture. Many design decisions were motivated by the long range objective to simulate blood flow in human blood vessels, especially in aneurysms, but have proven to be much more generally applicable. After explaining implementation details and how they were influenced by the target platform, the performance and memory requirements of this prototype solver are evaluated.

1. Introduction

Cardiovascular disease is the major cause of death in developed nations [1]. Local artery vessel wall irregularities, the so-called aneurysms, are frequent findings in healthy population. We focus on aneurysms of the cerebral arteries, which typically occur in artery bifurcations or in curved vessel segments. Intracranial hemorrhage after rupture leads to death or enduring impairments in more than 50% of the cases.

In the past years numerous in vivo, in vitro and numerical experiments showed a connection between hemodynamic factors and aneurysm initiation, growth and rupture (see e. g. [2–5]). Unfortunately, it is hardly possible to measure reliable in vivo hemodynamic parameters, at least for human intracranial arteries. Fluid simulation, however, may help us to simulate the hemodynamics in real geometries (see e.g. [6]), to understand the growth and rupture mechanisms of aneurysms and also support planning their therapy, which is also explored in the @neurIST project [7]. The question we deal with here is how specialized hardware and optimized software can cope with this computationally very demanding task and make this evolution feasible in the near future.

The rest of this paper is organized as follows: First the lattice Boltzmann model we use in our approach and the concept of the Cell Broadband Engine Architecture are introduced. After describing the preliminary work we outline our implementation. Then the single and multicore performance for a channel flow and a real-world geometry is evaluated and compared to another performance optimized implementation for standard CPUs. The paper ends with conclusions and an overview of future work.

1.1. The lattice Boltzmann method

The lattice Boltzmann method is an alternative to classical Navier–Stokes solvers and works on an equidistant grid of cells, the so-called lattice cells, which only interact with their direct neighbors. In this study we use the common three-dimensional D3Q19 model originally developed by Qian, d'Humiéres and Lallemand [8] with $N = 19$ the so-called

* Corresponding author.
E-mail address: markus.stuermer@informatik.uni-erlangen.de (M. Stürmer).

0898-1221/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2009.04.006

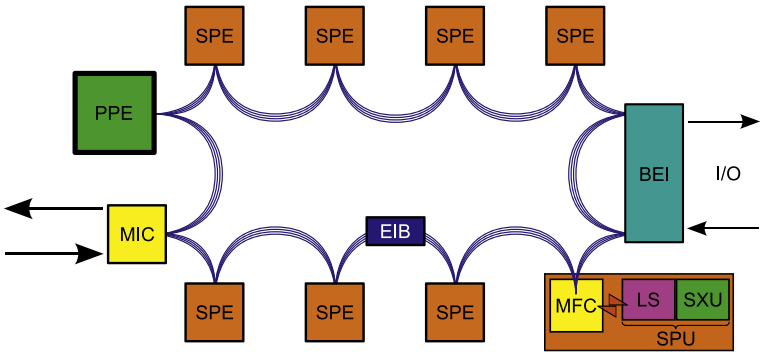

Fig. 1. Diagram of the Cell Broadband Engine Architecture.

distribution functions (DFs) f_α and the corresponding dimensionless discrete velocity set $\{e_\alpha | \alpha = 0, \ldots, N-1\}$. This model was shown to be both stable and efficient (see [9]). The macroscopic fluid moments' density ρ and mass flux $\rho\mathbf{u}$ can be calculated from the distributions using:

$$\rho = \sum_\alpha f_\alpha \tag{1}$$

and

$$\rho\mathbf{u} = \sum_\alpha \mathbf{e}_\alpha f_\alpha. \tag{2}$$

For the work presented in this paper, we adopt a lattice-Boltzmann–Bhatnagar–Gross–Krook (LBGK) scheme (see [10,8]):

$$f_\alpha(\mathbf{x}_i + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}_i, t) - \frac{1}{\tau}[f_\alpha(\mathbf{x}_i, t) - f_\alpha^{(eq)}(\mathbf{x}_i, t)], \tag{3}$$

where $\mathbf{x}_i$ is a cell in the discretized simulation domain, $f_\alpha^{(eq)}$ represents the equilibrium distribution, τ is the relaxation time, t is the current time step and $t + \Delta t$ is the next time step. Eq. (3) can be separated into two steps, known as the collision step (4) and the streaming step (5), respectively:

$$\tilde{f}_\alpha(\mathbf{x}_i, t) = f_\alpha(\mathbf{x}_i, t) - \frac{1}{\tau}[f_\alpha(\mathbf{x}_i, t) - f_\alpha^{(eq)}(\mathbf{x}_i, t)], \tag{4}$$

$$f_\alpha(\mathbf{x}_i + \mathbf{e}_\alpha \Delta t, t + \Delta t) = \tilde{f}_\alpha(\mathbf{x}_i, t), \tag{5}$$

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

More details on the lattice Boltzmann algorithm and its derivation can be found in [11,12].

1.2. The STI cell processor

Over the last decade the performance increase of mainstream CPUs has slowed down whereas power requirements have continued to grow. As increasing clock frequency is hitting technical limits, more cores with reduced clock rate are now integrated on a chip — leading to both a reduced power consumption and an increased overall performance. Not following other designs, which integrate multiple cores of the same type on a single die, Sony, Toshiba and IBM (STI) introduced the heterogeneous Cell Broadband Engine Architecture (CBEA) [13]. The current Cell implementation, depicted in Fig. 1, combines a general purpose core with eight cores optimized for memory throughput and data processing.

The so-called Power Processor Element (PPE) is a simplified, PowerPC compliant 64-bit RISC core executing the operating system and controlling program execution. The workhorses, however, are the Synergistic Processor Elements (SPEs), each consisting of the Synergistic Execution Unit (SXU), 256 kB of fast Local Store (LS) holding the program the SXU executes and its operands, and the Memory Flow Controller (MFC) that is used for communication and copying data between LS and main memory with asynchronous DMA transfer. SXU and LS constitute the Synergistic Processor Unit (SPU).

The Element Interconnect Bus (EIB) connects all nine cores and has also the Memory Interface Controller (MIC) and Broadband Engine Interface (BEI) attached. The MIC supports Rambus Extreme Data Rate (XDR) memory, providing a theoretical peak bandwidth of 25.6 GB/s. The BEI is used to access IO devices or create a cache-coherent connection to another processor. In the IBM QS20 or QS21 blade servers, e. g., two Cell processors, each with a local main memory of 512 MB or 1 GB respectively, are combined to a cache-coherent non-uniform memory access (ccNUMA) system. Sony's Playstation 3

Table 1
Fraction of patches that actually has to be allocated using the domain decoupling method for five different real and preprocessed brain aneurysm geometries.

Geometry	Size	Patches
A	250×250×220	1388 of 28 672
B	250×250×130	515 of 17 408
C	250×250×200	1515 of 25 600
D	220×190×129	995 of 11 424
E	330×330×330	2501 of 74 088

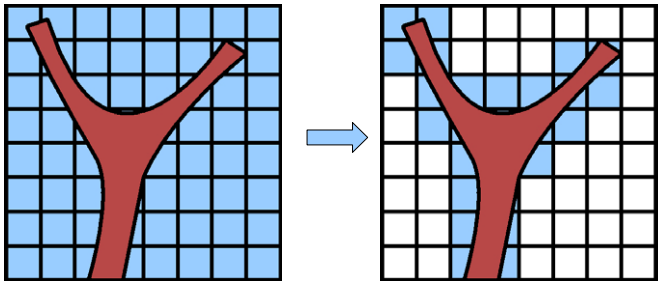

Fig. 2. Example of domain decoupling, white patches need not to be stored.

is also based on the CBEA, but for Linux running as a guest system on it only six SPEs are available, reducing the maximal performance accordingly.

Programming the PPE is basically like programming any common general purpose CPU. To optimize programs for CBEA, however, it is necessary to identify tasks suitable for moving on the SPEs and to create the corresponding code in small programs, performing the necessary communication, DMA orchestration and synchronization. Depending on complexity and desired performance, it is possible to use single source compilers, employ frameworks or control everything manually in C or C++ using language extensions and library support.

2. Preliminary work

Diagnostic assessment of vessel diseases requires modern imaging modalities including magnetic resonance imaging, computed tomography–angiography and digital subtraction angiography (DSA). Handling the large 3D datasets is especially challenging.

Domain decoupling

The capability of the lattice Boltzmann method has been demonstrated recently for various applications, especially for complex geometries [14–16]. Furthermore, the data structure introduced in [17] reduces the memory consumption for highly complex sparse geometries like vessels in the human brain significantly. This is achieved using a domain decoupling and elimination of areas that consist of solid only and where therefore no computations take place. In Fig. 2 an example bifurcation is divided into equally sized so-called patches, where only patches need to be stored which contain the fluid parts of the vessel. As shown in Table 1, this reduces the memory consumption in 3D for real patient geometries by 90% and more.

Improving performance

The geometry data for the simulation is acquired using a rotational, flat detector equipped C-arm DSA scanner and preprocessed by a segmentation algorithm. The resulting datasets are characterized by excellent spatial resolution and a low artifact load. These modern imaging techniques generate huge data sets with 512×512×512 voxels and more, which lead to large memory consumption but results in sufficiently fine resolution of the vessels. A simulation with this resolution would need more than 40 GB using a standard LBM implementation with two full grids and double precision. Even after reducing memory consumption with the technique introduced above, hemodynamic simulations are still computationally very intensive. Besides choosing a different approach, there are the following options to reduce the computational time:

- A parallel program on a dedicated high performance computer on site (e. g. in the hospital)
- A parallel program on a general purpose high performance computer off site, where the data is sent to
- A system built with special purpose processors (e. g. Cell processor, FPGA, or GPU).

As a high performance computer in the hospital is expensive, needs to be maintained, and might not be used for further computations, and since it is problematic to send highly sensitive patient data to a general computing center, we expect sooner acceptance of the third option, especially if included in the clinical environment with relatively low costs.

In particular graphics cards and the Cell processor are suggestive for this purpose, as both are produced cost-effectively in large volumes. In this article we have chosen the Cell processor. For a perspective on a lattice Boltzmann solver in 2D using graphics hardware see e. g. [18].

Simplifications

In general blood is a non-Newtonian fluid, i. e. the viscosity is not constant. However, experiments have shown that in large vessels the non-Newtonian effects of blood are of minor importance and the viscosity of blood asymptotes to a value of 3–4 mPa s at a temperature of 37 °C. Thus, blood is considered to be a Newtonian, isotropic, incompressible and homogeneous fluid with a viscosity of 4 mPa s here. Furthermore, the walls of the vessel are treated as rigid and the maximum inflow velocity is taken from the literature (see [19]) depending on the simulated vessel. For simplicity, the time dependency of the real inflow velocity is approximated using a combination of a $\sin^2$ function and a constant (see [17]). In principle also patient specific measurement values of the cardiac cycle or profiles from the literature (see e. g. [20]) can be used as input parameters, but are actually not implemented in the prototype solver.

3. Efficient implementation on the CBEA

An efficient implementation has to exploit the strengths and obey the restrictions of the CBEA. For a good introduction into Cell programming we suggest [21]. The most important aspects are:

- As Cell hardware with more than 2 GB of main memory is currently not available, it is a scarce resource and must not be wasted.
- Most computation should take place on the SPEs, as they have more computational power and better bandwidth to main memory than the PPE. Optimal bandwidth is reached if multiple blocks of 128 B – corresponding to cache lines of the PPE – are transferred to an equally aligned address in LS.
- Further computation should be done in SIMD mode, as scalar operations are not available and must be emulated. Since loading or writing naturally aligned 16 B vectors are the only memory operations to the LS the SPU supports natively, as many SIMD operands as possible should be aligned that way. Otherwise up to two aligned loads and a so-called shuffle command are necessary to generate an unaligned vector operand, leading to drastically reduced performance.
- A good memory layout must support parallel processing in a ccNUMA environment, e. g. to utilize 16 SPEs and two distinct memory buses on the IBM QS20 and QS21 blades.
- As SPUs have long penalties for branch misses and only support static prediction in software, branches should be avoided wherever possible.
- The current implementation of the Cell processor can perform 204.8 GFlop/s in single precision, with truncation as the only rounding mode supported. Double precision will reach only about 14.3 GFlop/s, but with the usual round-to-nearest mode. Its successor with enhanced double precision, expected to debut in 2008, will increase this to 102.4 GFlop/s. We therefore restrict ourselves to single precision at the moment, but expect those results to be valid for double precision in the near future.

Memory layout

Our main concern is the memory layout, since it is the key to efficiency. Our implementation divides the domain into patches of $8\times8\times8$ lattice cells and only patches containing fluid are actually allocated and processed, as described in Section 2. This approach combines the good performance of structured domains with the ability to process irregular geometries efficiently. The necessary data to update a patch fits easily into the 256 kB LS.

Inside each patch, data is organized as structure-of-arrays. A line of a distribution function contains 8 lattice cells and therefore covers two SIMD vectors—containing four single precision floating point values each. When aligned appropriately, a plane containing 8 lines will reside in two cache line blocks and can be optimally transferred by DMAs.

To process a patch, data is also needed from 18 neighboring patches in the D3Q19 model. To better support this data exchange, copies of those distribution functions are reordered contiguously on the SPE, stored after each update and collected from neighboring patches again before an update begins. For each patch we need 30 "planes" of individual distribution functions corresponding to 5 distribution functions at each face, and analogously 12 "lines" of distribution functions corresponding to one distribution function at the 12 edges of the cube.

While only one version of the inner patch is necessary, two copies of these exchange structures are required, one holding the distribution functions from the last time step, and one holding the results of the current time step. This enables processing patches in any order and thus also a parallel execution. While this supports easy and efficient data transfer, streaming becomes much more tricky on the downside, but not much slower.

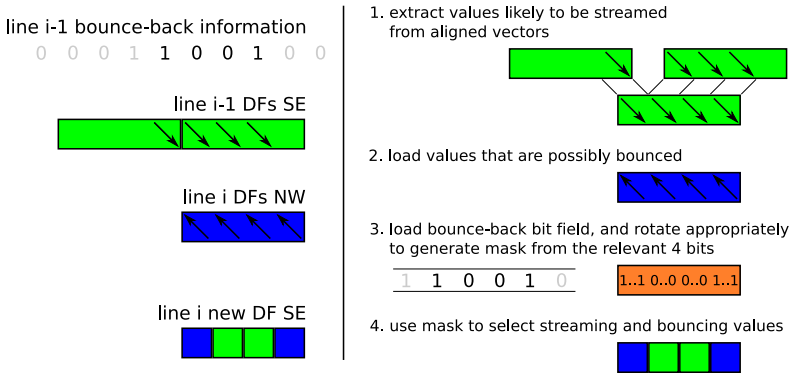

Fig. 3. Bounce-back using SIMD operations. Overview of data structures involved on the left. On the right: Operations taken to perform the bounce-back, exemplified by calculating a vector of new south-east (SE) distribution functions in line *i*.

Boundary conditions

Solid lattice cells that require bounce-back to represent no-slip conditions at obstacles or domain boundaries can occur anywhere, their location is described by a bit field stored with each patch. Streaming is done by loading an SIMD vector containing the distribution functions likely to be streamed as well as one containing those that might be reflected. By using mask generation and select operations provided by the SPU, the correct values are chosen according to the bits of bounce-back information. This is depicted in more detail in Fig. 3. Performance does not depend on the structure inside a patch and is further enhanced by register blocking. In this study simple bounce-back conditions are used. To improve accuracy, second order boundary conditions (see [22,9]) can be implemented, generating a more complicated and expensive algorithm for the lattice cells near the wall.

Further boundary conditions currently implemented are source and sink cells, which are calculated using the equilibrium distribution and lead to an inflow and outflow when placed at boundaries, and a pressure outflow condition based on [16]. Source and sink conditions have little overhead, but patches affected by pressure outflow are more expensive to handle, both in terms of memory bandwidth and computation. Pressure boundaries are treated as solid lattice cells during streaming, but in the neighboring cells the reflected distribution functions must be adapted before collision can take place. For this correction, the velocity and pressure field from the last time step are required and therefore have to be stored after and fetched before processing such a patch.

Collision optimized for SIMD units

As SIMD operations do not throw exceptions, collision can calculate fake values for solid lattice cells and its implementation is trivial. The collision step has been implemented to exploit common subexpressions and to take advantage of the fused multiply–add operations. It performs 167 floating point operations coded into 115 assembly instructions. Since the collision already contains the calculation of macroscopic velocity and density values, they can also be stored to memory during this step and later written by a PPE to a data file for visualization.

Parallelization

SPEs negotiate about which is processing which patch by counting with means of atomic increments. Running on a ccNUMA system, an equal number of patches can be statically allocated on each memory location and processed by the corresponding SPEs. Inter-node traffic only occurs to exchange data with neighboring patches allocated in remote memory and to synchronize between time steps.

4. Results

Typical means of expressing the performance of LBM codes are the number of lattice updates per second (LUPS) or of fluid lattice updates per second (FLUPS). As every patch contains 512 lattice cells, the lattice update rate depends mainly on how many patches can be processed per time. The fluid lattice updates are then calculated by multiplication by the fraction of fluid lattices.

Depending on the implementation and the abilities of the hardware, streaming may take as much or even more time than the collision, but no single floating point operation (Flop) is performed. Although they can be easily calculated, Flop rates are therefore less common to describe LBM performance.

Table 2
Performance of a straightforward single precision LBM implementation in C on an Intel Xeon 5160 at 3.0 GHz, a standard 3.2 GHz PPE and SPU, compared with the optimized SPU-kernel for a 8^3 fluid lattice cells' channel flow.

	Straightforward C			Optimized
CPU	Xeon	PPE	SPU	SPU
MFLUPS	10.2	4.8	2.0	49.0

Table 3
Cell MLUPS performance for a 96^3 channel flow; MFLUPS = MLUPS $\cdot \frac{94^3}{96^3}$.

CPUs	PS3	QS 20		
	One	One	Both	Both
Memory	Local	Local	Interleaved	NUMA-aware
1 SPE/CPU	42	40	73	70
2 SPEs/CPU	81	79	129	136
3 SPEs/CPU	93	107	156	189
4 SPEs/CPU	94	110	166	204
6 SPEs/CPU	95	110	174	205
8 SPEs/CPU	N/A	109	173	200

SPU performance

Table 2 compares the performance of a lattice Boltzmann implementation in C and our SIMD optimized implementation that is mainly written in SPU assembly language. To get an impression of the performance of the simple implementation, we include results obtained on a single core of an Intel Core2 Duo processor. To make the results comparable, we make the problem fitting into the general purpose CPUs' level 2 caches or a SPE's LS. Besides having to add instructions to transfer data between main memory and LS, we would primarily measure the different memory bandwidths. Instead we measure the performance of the SPU part (SXU and LS) of a SPE only. It can be seen, that the PPE cannot keep up with a modern server processor. However, performance on the SPE is worst due to the huge overhead of performing scalar operations on the SIMD units. While advanced compilers can vectorize simpler scalar descriptions, they cannot exploit SIMD units in the LBM program automatically.

As shown by our optimized implementation, the poor performance is not due to the inadequacy of the SPU architecture for the LBM algorithm: As described in Section 3, the optimized version uses carefully aligned buffers to exploit SIMD units. Above all we use select operations instead of branches except for loops, as conditional jumps are inherently scalar operations.

Channel flow

To examine the performance of the complete solver, we first choose to benchmark a standard test application in CFD: the channel flow. Table 3 presents performance on the Sony Playstation 3 and an IBM QS20 blade server, with different number of SPEs utilized. When only one Cell processor of the blade is used, all data is allocated in the main memory local to that CPU. For utilizing both processors, two approaches were evaluated: In the first one, patches were not assigned to a certain node, but memory pages were allocated alternating on both memory locations (interleaved). The second method is aware of the NUMA architecture and assigns half of the patches explicitly to each Cell processor and its attached memory.

The results show that at most 4 SPEs are sufficient to saturate the local memory bus and using more SPEs might even decrease the efficiency due to increased synchronization overhead. Only for the interleaved case employing more SPEs is advantageous, as it helps saturating the limiting interconnect.

We can currently only guess why the Playstation 3 is less than half as fast as the whole QS20 blade with two processors. First, the Linux operation system is only running in a virtualized environment on the Playstation 3, and different kernel versions were in use. Second, probably both PPEs can serve page faults on the blade simultaneously, and we assume that this is especially the reason for the very good single CPU performance there. Unsurprising, memory benchmarks have shown a great influence of page size and the number of page faults, as a linear memory stream at 20 GB/s will lead to a page fault about every 0.2 μs for the 4 kB pages used by default.

We see that allocating memory in an interleaved way enables acceptable performance of a NUMA-unaware program when both CPUs are used. But regarding the ccNUMA architecture by diligently assigning data the work is worthwhile.

Blood vessel geometry

Finally, we want to evaluate the performance for simulating blood flow in a real-world geometry depicted in Fig. 4. The data set of $250\times250\times220$ voxels (geometry A in Table 1) was acquired using digital subtraction angiography and shows a large saccular aneurysm in front of a vessel bifurcation. In this geometry only about 2.5% of the voxels represent fluid.

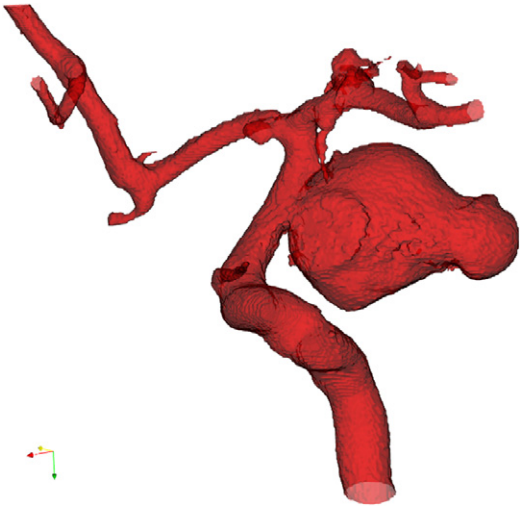

Fig. 4. Real intracranial vessel geometry with a saccular aneurysm.

Table 4
Aneurysm geometry MFLUPS performance.

	HP DL140G3	Sony PS3	IBM QS20
1 core or SPE	9.9	21.1	19.5
1 CPU	11.7	43.8	49.1
SMP node	21.1	N/A	90.0

We compare our simulation with a performance optimized code for complex irregular geometries developed within the *International Lattice Boltzmann Development Consortium* (LB–DC) [23]. The resulting data structure in this code is a 1D list containing the distribution functions' values of all fluid lattice cells (for the current and the next time step) as well as a second 1D list with information about the adjacency of the lattice cells used during propagation. This allows us to process the fluid lattice cells in any order, since adjacency of lattice cells is explicitly stored and need no longer be obtained by index arithmetics. All benchmarks of the LB–DC code ran with a propagation optimized structure-of-arrays data layout and a combined collide-propagate algorithm on a HP DL140G3 cluster node of the "Woody" cluster at Erlangen. It provides a UMA architecture that connects two dual-core Intel Xeon 5160 processors at 3.0 GHz with two memory buses providing a theoretical peak bandwidth of 10.6 GB/s each. The memory requirements of this implementation for the given geometry are about 128 MB.

For our implementation about 48% of all allocated lattice cells represent fluid, so only every second update counts as FLUP. Around 116 MB main memory are occupied. Performance reached by both implementations in single precision is shown in Table 4 for various usages of cores or SPEs respectively. Most notable is the performance a single SPE can achieve compared to a full-featured server CPU core. However, both implementations do the computation very efficiently, as using multiple cores to increase performance is limited on both machines: Using the second Xeon core or more than three SPEs (compare Table 3) on a node can increase the performance only slightly. One advantage of our Cell implementation here is the higher bandwidth of the XDR memory.

Our optimized code is not portable on the one hand, on the other hand it does not make sense to benchmark the code on CBEA hardware that does not exploit its features and only runs on the PPE. So we have to compare different implementations on different platforms, which does not allow more conclusions.

5. Conclusions and future work

We have shown the impressive potential of the CBEA for fluid flow simulation using the lattice Boltzmann method. To illustrate the performance measured in Section 4: The Sony Playstation 3, currently selling at less than 400 EUR, outperforms a modern Intel Xeon 5160 processor running at 3.0 GHz by a factor of 3.7 for the simulation in the given geometry. While getting this performance out of this novel design needs a lot of expertise and hand-tuning – at least at this early stage – the labor pays off.

The energy- and cost-effective design of this architecture is especially advantageous, since an excellent performance could be obtained by the Cell processor, which could be integrated directly into clinical equipment or put into a "super computer under the desk".

Further we could show that blockwise structured data layouts can be competitive also to adapt unstructured description for quite complex domains—in memory consumption as well as in performance.

In the future we plan to improve the performance further, but we would also like to enhance accuracy and applicability. Performance cannot be increased by code optimization, as currently less than half the memory bandwidth is available of what could be processed. The straightforward solution of this issue is to extend the solver for parallel execution on a CBEA-based cluster.

Another approach is to adopt strategies from cache blocking techniques [24] to the explicit data management on the SPEs. This might allow us to perform multiple time steps in a single sweep with little overhead. One of the hardest challenges to realize this efficiently comes from the small size of LS that can hold only a small number of lattice cells at the same time—this problem is becoming worse by the dependency on SIMD processing. However, as the gap between memory bandwidth and single-die performance is expected to widen further, such techniques may become increasingly important in the future.

It should be examined if calculations in single precision with truncation as rounding mode deliver sufficient precise results. First numerical experiments have shown that we can expect a slight loss of mass in the current implementation, with even less change in velocity, but different approaches exist to increase the accuracy and stability. Eventually, using double precision with the common round-to-nearest rounding mode will be an option in the next generation of Cell hardware.

In our simulation we used the popular single-relaxation time LBGK model, which is the simplest model for particle collisions and is limited in its general application due to a coupled derivation of all moments. This model is easy to implement and easy to parallelize, but on the other hand not as accurate as two-relaxation time (TRT) or multi-relaxation time (MRT) models, which promise to repair the shortcomings of BGK. The MRT model offers complete flexibility to specify simulation parameters, but is much more computationally expensive than BGK, whereas the TRT is a compromise between the extremes. To improve the accuracy, one of these models could perhaps be included in our simulation without losing performance, since we are actually memory bounded and have additional free computing resources.

Besides pressure and velocity, shear stresses play an important role in aneurysm initiation and growth. In contrast to Navier–Stokes solvers, where the stresses are calculated by computing the derivative of the velocity field, the stresses in LBM can be computed from distribution functions during collision. The computation of the stresses would further improve the applicability of the code and give the surgeon additional information. Furthermore, oscillatory shear stresses could be computed, which are the most commonly used indicator to quantify the sensibility of endothelial cells to temporal fluctuations in wall shear stress [25].

Acknowledgments

We would like to thank Thomas Zeiser at the Regional Computing Center Erlangen (RRZE) for benchmarking the LB–DC code on the "Woody" cluster and Marcus Prümmer and Eva Kollorz at the Institute of Pattern Recognition, University of Erlangen-Nuremberg, for help with segmenting the data sets. Our thanks also go to Dr. Wohlmuth and his group at IBM Lab Böblingen for permitting us to explore the Cell processor on development hardware. Finally we thank Wilhelm Homberg at the Research Centre Jülich for providing us access to their JUICE Cell blade cluster.

The work was supported by the Deutsche Forschungsgemeinschaft DFG (project Ru 422/7-5) and by the Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern (KONWIHR).

References

[1] Copenhagen: WHO Regional Office for Europe, Highlights on health in Germany 2004, Tech. Rep., World Health Organization, 2006. URL http://www.euro.who.int/document/e88527.pdf.
[2] A. Burleson, V. Turitto, Identification of quantifiable hemodynamic factors in the assessment of cerebral aneurysm behavior. On behalf of the subcommittee on biorheology of the scientific and standardization committee of the ISTH, in: Thrombosis and Haemostasis, vol. 76, 1996, pp. 118–123.
[3] K. Kayembe, M. Sasahara, F. Hazama, Cerebral aneurysms and variations in the circle of Willis, Stroke 15 (5) (1984) 846–850.
[4] M. Castro, C. Putman, J. Cebral, Computational Modeling of Cerebral Aneurysms in Arterial Networks Reconstructed from Multiple 3D Rotational Angiography Images, in: A. Amini, A. Manduca (Eds.), Medical Imaging 2005: Physiology, Function, and Structure from Medical Images, vol. 5746, number 1, SPIE, 2005, pp. 233–244.
[5] T.-M. Liou, H.-N. Liou, A review on in vitro studies of hemodynamic characteristics in terminal and lateral aneurysm models, in: Proceedings of the National Science Council, Republic of China, vol. 23, number 4, National Science Council, Republic of China, 1999, pp. 133–148.
[6] A. Artoli, A. Hoekstra, P. Sloot, Mesoscopic simulations of systolic flow in the human abdominal aorta, J. Biomech. 39 (5) (2006) 873–884.
[7] @neurIST - integrated biomedical informatics for the management of cerebral aneurysms, 2008. URL http://www.aneurist.org.
[8] Y. Qian, D. D'Humières, P. Lallemand, Lattice BGK models for Navier–Stokes equation, Europhys. Lett. (EPL) 17 (6) (1992) 479–484.
[9] R. Mei, W. Shyy, D. Yu, L.-S. Luo, Lattice Boltzmann method for 3-D flows with curved boundary, J. Comp. Phys. 161 (2002) 680–699.
[10] P. Bhatnagar, E. Gross, M. Krook, A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems, Phys. Rev. 94 (3) (1954) 511–525.
[11] S. Succi, The Lattice Boltzmann Equation — For Fluid Dynamics and Beyond, Clarendon Press, 2001.
[12] S. Chen, G. Doolen, Lattice Boltzmann method for fluid flows, Annu. Rev. Fluid Mech. 30 (1998) 329–364.
[13] IBM, Cell Broadband Engine Architecture, Oct. 2007. URL http://www.ibm.com/developerworks/power/cell/.
[14] A. Artoli, Mesoscopic computational haemodynamics, Ph.D. Thesis, University of Amsterdam, 2003.
[15] K. Iglberger, Performance analysis and optimization of the lattice Boltzmann method in 3D, Bachelor Thesis, University of Erlangen-Nuremberg, Computer Science 10 — Systemsimulation, 2003. URL http://www.10.informatik.uni-erlangen.de/Publications/Theses/2003/Iglberger_BA.ps.gz.
[16] N. Thürey, Physically based animation of free surface flows with the lattice Boltzmann method, Ph.D. Thesis, University of Erlangen-Nuremberg, Computer Science 10 — Systemsimulation, 2007.
[17] J. Götz, Simulation of bloodflow in aneurysms using the Lattice Boltzmann method and an adapted data structure, Tech. Rep., University of Erlangen-Nuremberg, Computer Science 10 — Systemsimulation, 2006. URL http://www10.informatik.uni-erlangen.de/Publications/TechnicalReports/TechRep06-6.pdf.

[18] J. Tölke, Implementation of a lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA, Comp. Vis. Sci. 2008. URL http://dx.doi.org/10.1007/s00791-008-0120-2.
[19] C. Ruge, Altersabhängigkeit der Hirndurchblutung im Erwachsenenalter: Eine farbduplexsonographische Studie, Ph.D. Thesis, Eberhard-Karls-Universität zu Tübingen, 2003.
[20] D. Holdsworth, C. Norley, R. Fraynek, D. Steinman, B. Rutt, Characterization of common carotid artery blood-flow waveforms in normal human subjects, Phys. Meas. 20 (1999) 219–240.
[21] IBM, Cell Broadband Engine Programming Tutorial, 2007. URL http://www.ibm.com/developerworks/power/cell/.
[22] M. Bouzidi, M. Firdaouss, P. Lallemand, Momentum transfer of a Boltzmann-lattice fluid with boundaries, Phys. Fluids 13 (11) (2001) 3452–3459.
[23] L. Axner, J. Bernsdorf, T. Zeiser, P. Lammers, J. Linxweiler, A. Hoekstra, Performance evaluation of a parallel sparse lattice Boltzmann solver, J. Comp. Phys. 227 (10) (2008) 4895–4911.
[24] M. Kowarschik, Data locality optimizations for iterative numerical algorithms and cellular automata on hierarchical memory architectures, in: Advances in Simulation, vol. 13, SCS Publishing House, 2004.
[25] D. Ku, D. Giddens, D. Philips, D. Strandness, Hemodynamics of the normal human carotid bifurcation: In vitro and in vivo studies, Ultrasound Med. Biol. 11 (1985) 13–26.