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

Lehrstuhl für Informatik 10 (Systemsimulation)

Blood Flow Simulation on the Cell Broadband Engine using the Lattice Boltzmann Method

Markus Stürmer, Jan Götz, Gregor Richter, Ulrich Rüde

Short Report 07-9

Abstract

Cardiovascular diseases are a major issue in modern health care systems. We demonstrate possibilities to efficiently simulate the flow in blood vessels; our long range objective is to help understanding the development of local vessel dilatations, so-called aneurysms, and to support planning their treatment. The fluid dynamics are modeled using the lattice Boltzmann method, since it is well-suited for handling complex domains. To produce results in acceptable time we propose to use highly tuned software for special purpose hardware rather than conventional high performance clusters. To demonstrate the potential of this approach and its suitability for the application, we describe and evaluate a performance optimized prototype solver on the Cell broadband engine architecture.

1 Introduction

This chapter is organized as follows: First, we motivate the simulation of fluid flow in diseased vessels and describe the geometries. Then, the simulation method, the so-called lattice Boltzmann method, is introduced. The section is ended by an overview of the Cell processor, a hybrid multi-core architecture which is exploited to simulate hemodynamics in aneurysms.

1.1 Aneurysms

Cardiovascular disease is the major cause of death in developed nations [7]. Local cerebral artery vessel wall irregularities, so-called aneurysms, are frequent findings in healthy population. The prevalence amounts to about 2% [22]. Aneurysms of the cerebral arteries typically occur in artery bifurcations or in curved vessel segments and may lead to perilous intracranial hemorrhage in case of rupture. According to a systematic literature review (see [26]) the annual risk of cerebral aneurysm rupture for people harbouring at least one aneurysm is specified with approximately 1%. Subgroup analysis revealed female gender, age > 60 years and aneurysm diameter > 5 mm as significant risk factors for higher incidence of aneurysm rupture.

Diagnostic assessment of vessel diseases requires modern imaging modalities including magnetic resonance imaging, computed tomography-angiography and digital subtraction angiography (DSA). In cerebral DSA, x-ray fluoroscopy is used to navigate a diagnostic catheter into the cervical arteries supplying the cerebral arteries with blood. After acquisition of a plain radiography, another radiography is acquired after contrast-dye injection. The contrast-dye image subtracted of the blank x-ray image results in a DSA-image. Acquisition of multiple x-ray images (rotational C-arm) allows assessment of 3D-reconstruction images of intracranial vessels in high spatial solution. This 3D-imaging is currently the gold standard for planning the treatment of ruptured and unruptured cerebral artery aneurysms.

Different types of aneurysms are shown in Figure 1, the most frequent ones in the human brain are berry and saccular aneurysms.

Treatment options for cerebral artery aneurysms are neurosurgical aneurysm clipping or endovascular aneurysm coil embolization. Aneurysm clipping requires neurosurgical skull trepanation. Endovascular aneurysm coil embolization means packing of the aneurysm with platinum microcoils via a microcatheter introduced in the aneurysm sac. The international subarachnoid aneurysm trial showed that the treatment of intracranial aneurysms with endovascular coiling was preferable to neurosurgical clipping if both treatments were possible because of the significantly better outcome [20].

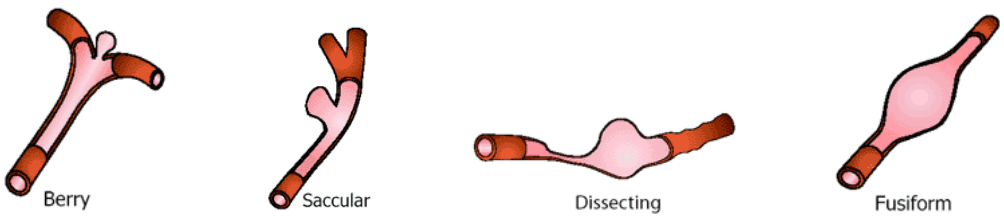


Figure 1: Different types of aneurysms (from [4]).

1.2 The Lattice Boltzmann Method

The lattice Boltzmann method is an alternative to classical Navier-Stokes solvers and works on an equidistant grid of cells, so-called lattice cells which only interact with their direct neighbors. Each lattice cell contains a fixed number of distribution functions f_α representing an amount of fluid moving in a given velocity direction $e_\alpha, \alpha = 0, \dots N-1$. We use the common three-dimensional D3Q19 model (see Figure 2) originally developed by Qian and d'Humiéres [21] with N=19 distribution functions (DFs), which has shown to be both stable and efficient (see [19]). In this model, particles are only allowed to propagate along Cartesian directions $e_{1\dots6}$ with speed 1, in directions $e_{7\dots10} = (\pm1, \pm1, 0)$, $e_{11\dots14} = (0, \pm1, \pm1)$, $e_{15\dots18} = (\pm1, 0, \pm1)$ with speed $\sqrt{2}$, or to stay at rest (e_0). In the following all lattice values are characterized by the superscript *. The macroscopic

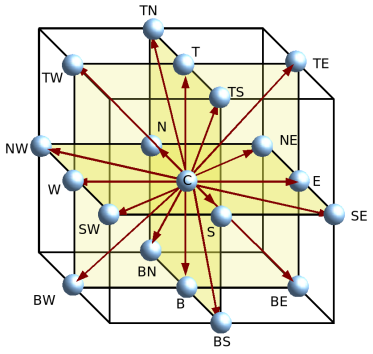


Figure 2: D3Q19 model for three dimensions.

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 [3]). The equilibrium distribution $f_\alpha^{(eq)}$ represents the state of the fluid, where the number of particles of a lattice cell in a discrete direction after collision is the same as the number which left the lattice cell in the time step before. This does not mean that the fluid is not moving, but that the DFs do not change during a time step. For the D3Q19 model the equilibrium distribution functions can be calculated from

$$f_\alpha^{(eq)} = \rho^* \cdot w_\alpha \cdot [1 + \frac{3}{c^2}\mathbf{e}_\alpha \cdot \mathbf{u}^* + \frac{9}{2c^4}(\mathbf{e}_\alpha \cdot \mathbf{u}^*)^2 - \frac{3}{2c^2} \cdot \mathbf{u}^* \cdot \mathbf{u}^*] \quad , \alpha = 0, \dots N-1 \,, \tag{3}$$

where $w_0 = 1/3$, $w_{1,\dots6} = 1/18$ and $w_{7,\dots18} = 1/36$. DFs of the next time step $t + \Delta t^*$ are acquired in two steps:

Collide

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

Streaming

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

where τ is the relaxation time, which can be calculated from

$$\tau = 3\nu^* + \frac{1}{2} \,, \tag{6}$$

using

$$\nu^* = \nu \frac{\Delta t}{\Delta x^2} \tag{7}$$

with ν being the kinematic viscosity of the fluid.

In the collide step which can be interpreted as many particle collisions, new DFs are calculated according to the old time step. During streaming all DFs besides f_0 are advected to their neighboring lattice cells depending on their velocities.
More details on the lattice Boltzmann algorithm and its derivation can be found in [9, 27, 15].

1.3 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 have multiple cores of the same type on a single die, Sony, Toshiba and IBM (STI) introduced the Cell Broadband Engine Architecture (CBEA) [11], which is a hybrid multi-core architecture. The current Cell implementation, depicted in Figure 3, offers a revolutionary design, combining one Power processor element (PPE) and eight simple single instruction multiple data (SIMD) cores, so-called *synergistic processor elements* (SPEs). This architecture places the Cell processor somewhere between classical general purpose CPUs and special purpose systems like graphics processing units (GPUs), whose application becomes more and more popular in high performance computing, or e. g. the Clearspeed Advance accelerator boards.

Besides being integrated in IBM's QS20 blades, the Cell processor is the heart of Sony's Playstation 3 gaming system and has also been put into other accelerator boards and servers.

The PPE is a PowerPC compliant, 64-bit RISC general purpose processor core. Providing 32 kiB L1 data, 32 kiB L1 instruction and 512 kiB L2 cache, it supports symmetric multithreading (SMT) but executes only in-order. It is primarily responsible to run the operating system and for program control.

The SPEs are optimized for efficient data processing. Each consists of the following three units: The synergistic processor unit (SPU) is a simple but fast SIMD-only co-processor. Its 256 kiB large local store (LS) is a fast cache-like memory that stores the SPU's instruction code and the data it processes. To transfer data between main memory and local store, DMA commands have to be issued explicitly to the associated memory flow controller (MFC). The MFC can process multiple DMA transfers concurrently, SPE programs can check or wait for their completion. Each SPE reaches 25.6 GFlop/s using fused multiply-adds in single precision at 3.2 GHz.

The memory interface controller (MIC) supports Rambus extreme data rate memory (XDR) which can deliver up to 25.6 GB/s theoretical bandwidth. The Broadband Engine Interface (BEI) uses Rambus Flex I/O technology to support I/O devices or to create a coherent connection to another Cell processor, forming a powerful ccNUMA system in e. g. IBM's QS20 servers. All components are connected by the element interconnect bus (EIB), a ring bus which provides a sustained maximum bandwidth of up to 204.8 GB/s.

In addition to a high raw performance, the chip is highly power efficient due to a low operating voltage and an advanced power management. However, the current SPUs are optimized for single precision floating point instructions, double precision (DP) stalls the pipeline for 6 cycles. But the next generation of the processor, which is manufactured using 65 nm SOI technology instead of 90 nm, will support DP fully pipelined. The single precision implementation is further not fully IEEE 755 compliant, particularly truncation (rounding to zero) is the only rounding mode currently available.

It should also be noted that only six SPEs are available for Linux running as a guest system on the Sony Playstation 3, reducing the maximum performance there accordingly.

Programming the PPU is basically like programming any common general purpose CPU. However code fragments to run on a SPE must be implemented as small programs and assembled separately into a specific byte code. A PPU program uses library functions to upload that program

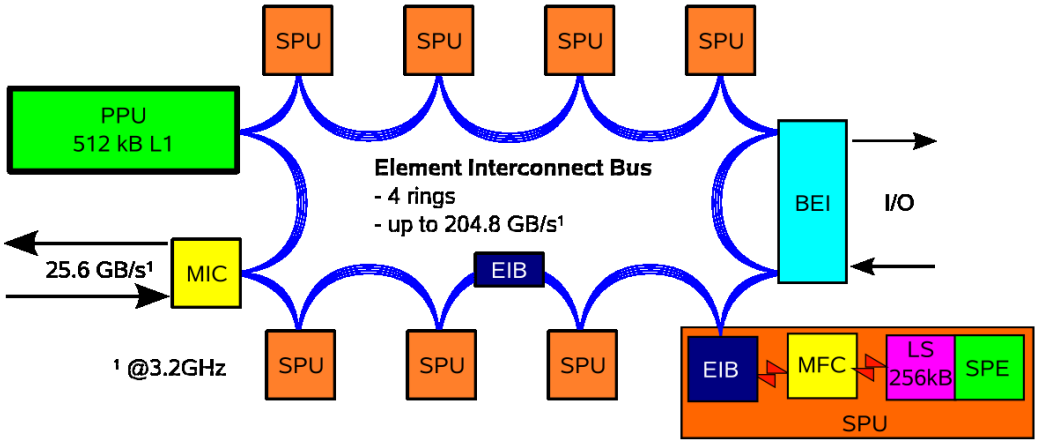


Figure 3: Diagram of the Cell Broadband Engine Architecture.

onto a SPE and starts it by issuing a synchronous system call. As a consequence, programs that should execute on multiple SPUs or at the PPU at the same time must be thread parallelized.

The best performance can often be achieved when the SPU program is implemented in C, C++ or assembly and DMA data transfers and communication are orchestrated by hand using language extensions. Tools can also generate the necessary stub functions from interface description language (IDL) to mimic remote procedure calls (RPCs). There are also different single source compilers under development that outsource work onto SPEs automatically or user directed. Since at the moment only early versions of these compilers are available and they cannot perform complex optimizations (e.g. memory optimizations), in this article we distribute the tasks by hand to achieve maximal performance.

2 Preliminary Work

In the last years numerous in vivo, in vitro and numerical experiments showed a connection between hemodynamic factors and aneurysm initiation, growth and rupture (see e. g. [5, 14, 6, 18]). However, it is hardly possible to measure reliable in-vivo hemodynamic parameters at least for human intracranial arteries. But computational based models established the feasibility to simulate hemodynamics in real aneurysm geometries, using high-quality 3D-datasets from e. g. computed tomography or DSA. These simulations enhance the understanding of aneurysm growth and rupture mechanisms. It may be one of the intentions to improve the predictability of aneurysm rupture risk, based on the clinically established risk factors with simultaneous consideration of flow dynamic risk factors.

Domain Decoupling

The capability of the lattice Boltzmann method has been demonstrated recently for various applications, especially for complex geometries [1, 13, 24]. Furthermore, the data structure introduced in [8] reduced the memory consumption for highly complex sparse geometries like vessels in the human brain significantly. This was achieved using a domain decoupling and elimination of unnecessary areas. In Figure 4 an example bifurcation is divided into equally sized so-called patches, where only patches need to be stored, which contain fluid parts of the vessel. This reduces the memory consumption in 3D for real patient geometries by about 90%.

Improving Performance

The geometry data for the simulation is acquired using modern imaging techniques like computed tomography, magnetic resonance imaging, or angiography and preprocessed by a segmentation algorithm. Modern scanners generate huge data sets with 512×512×512 voxels and more, which leads to large memory consumption but results in sufficiently fine resolutions 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 but could be reduced to about 3 GB with the technique introduced above. However, hemodynamic simulations are still computationally very intensive. Besides faster algorithms, the following options are possible to reduce the computational time during surgery:

- 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 processor (e. g. Cell processor, FPGA, GPU, Co-processor)

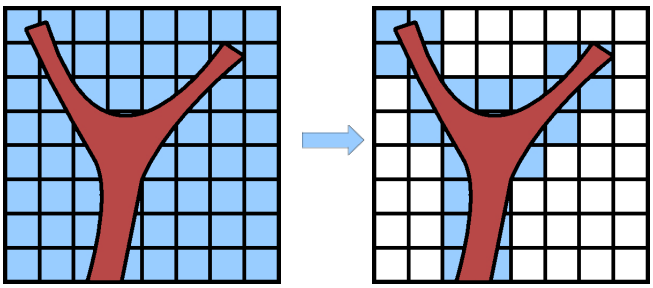


Figure 4: Example of domain decoupling, white patches need not to be stored.

As a high performance computer in the hospital is expensive, needs to be maintained, and might not be used for further computations, the first option is not taken into consideration. The second option is problematic, since it is critical to send highly sensitive patient data to a general computing center. Thus, the third option seems to be the most promising one since such a solution could be included in the clinical environment and leads to a relatively cost-effective solution. Two possibilities for this option are graphics cards and the Cell processor, which are both 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. [25].

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 elastic 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 literature (see [23]) depending on the simulated vessel (see Table 1). The time dependency of a real inflow is approximated using a combination of a $\sin^2$ function and a fixed velocity at the inflow (see Figure 5).

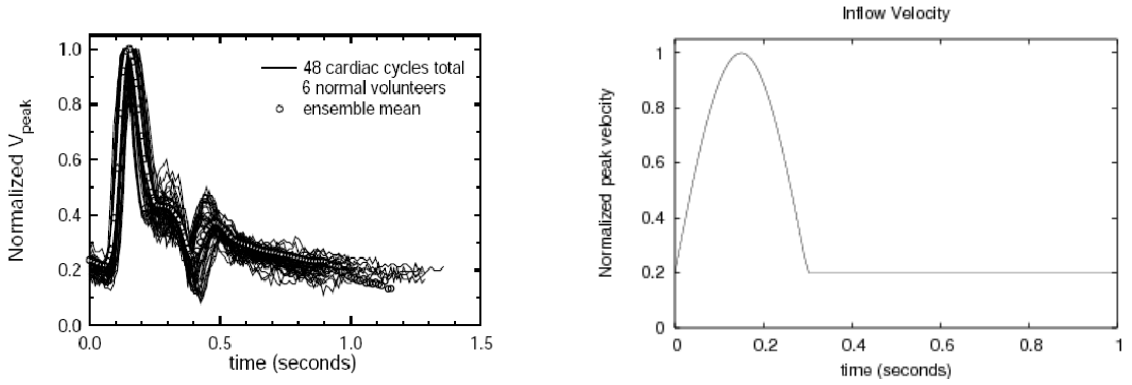
3 Efficient Implementation on the CBEA

An efficient implementation has to exploit the strengths and obey the restrictions of the Cell Broadband Engine architecture. For a good introduction into Cell programming we suggest [12]. The most important aspects are:

- As Cell hardware with more than 1 GiB of main memory is currently not available, it is a scarce resource and must not be wasted.

Vessel	Vs (cm/s)	TAV (cm/s)
common carotid artery (CCA)	96 ± 22	23 ± 5
external carotid artery (ECA)	84 ± 22	20 ± 6
internal carotid artery (ICA)	64 ± 14	24 ± 5
vertebral artery (VA)	48 ± 10	14 ± 4

Table 1: Velocities including standard deviation of different arteries: Velocities averaged from measurements with 156 persons of $20 - 85$ years age taken with Laser-Doppler-anemometry (TAV means time-averaged mean velocity).



(a) Measured blood flow in carotid artery (from [10]).

(b) Used flow profile.

Figure 5: Measured and used flow velocity profiles for the inflow.

- Most computation should take place on the SPEs, as they have more computational power and better bandwidth to main memory than the PPU. Optimal bandwidth is reached, if multiple blocks of 128 B — corresponding to cache lines of the PPU — are transfered 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 cache coherent non-uniform memory access (ccNUMA) environment, e. g. to utilize 16 SPEs and two distinct memory buses on the QS20 blades.
- As SPUs have long penalties for branch misses and only support static prediction in software, branches should be avoided wherever possible.

Memory Layout

Our main concern is the memory layout, since it is the key to efficiency.

Our implementation divides the domain into patches of $8 \times 8 \times 8$ lattice cells and only patches containing fluid are actually allocated and processed, like described in Section 2. This approach combines the good performance of structured domains with the ability to process irregular geometries efficiently. Data to update a patch fits easily into the 256 kiB 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 transfered 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 the 6 faces of the cubical patch and for each face 5 distribution functions and analogously 12 "lines" of distribution functions corresponding to the 12 edges of the cube.

Two copies are necessary, one holding the distribution functions that have been calculated in the last time step and one currently being written as input for the next 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.

Boundary Conditions

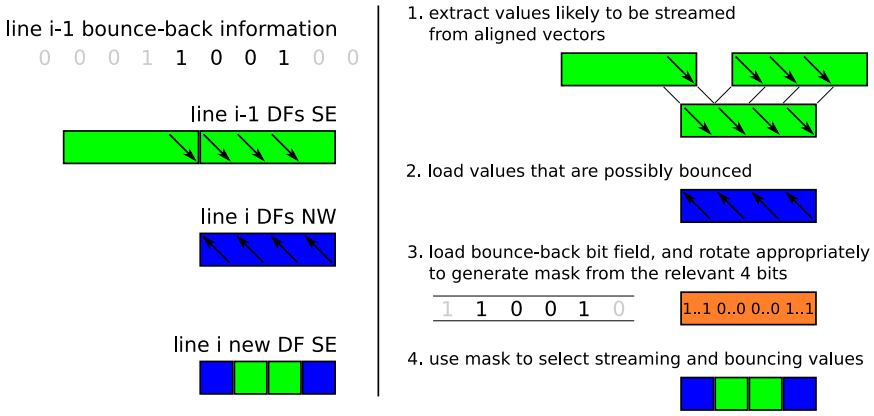


Figure 6: Bounce-back using SIMD operations. Overview of data structures involved on the left. On the right: Operations taken to perform the bounce-back, exemplified by calculating a vector of new south-east (SE) distribution functions in line i.

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 depictured in more detail in Figure 6. Performance does not depend on the structure inside a patch and is further enhanced by register blocking.

Further boundary conditions currently implemented are source and sink which are calculated using the equilibrium distribution and lead to an in- and outflow when placed at boundaries and a pressure outflow condition based on [24]. 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. During streaming, pressure boundaries are treated as solid lattice cells. Before collision can take place, the reflected distribution functions have to be adapted. To correct their value, the velocity and pressure field from the last time step are necessary 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

	straight-forward C			optimized
CPU	Xeon	PPE	SPE	SPE
MFLUPS	10.2	4.8	2.0	49.0

Table 2: Performance of a straight-forward single precision LBM implementation in C on an Intel Xeon 5160 at 3.0 GHz, a standard 3.2 GHz PPU and SPE, compared with the optimized SPE-implementation for a 8^3 fluid lattice cells channel flow.

calculation of macroscopic velocity and density values, they can also be stored to memory during this step and later written by a PPU 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 is statically allocated on each node and processed by the node's SPEs. Inter-node traffic only occurs to exchange data with neighboring patches allocated on another node and to synchronize between time steps.

4 Results

Performance Numbers

Typical means of expressing the performance of LBM codes are the number of lattice updates per second (LUPS) or of fluid lattice updates (FLUPS). As only patches containing 512 lattices are used, the lattice update rate depends mainly on the number of patches. 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.

SPE Performance

Table 2 compares performance of a lattice Boltzmann implementation in C and our SIMD-optimized implementation that is mainly written in SPU assembly. To get an impression of performance of the simple implementation, we included results obtained on a single Intel Core2 Duo core. In all cases performance is independent of main memory bandwidth, as the problem fits well into the general purpose CPUs' level 2 caches or a SPE's LS.

It can be seen, that the PPU 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 is 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 on one or both SMP nodes of an IBM QS20 blade server, with different numbers of SPEs utilized. For using both CPUs, two approaches were evaluated: In the first case, patches were not assigned to a certain node, but memory was allocated page-wise alternating on both memory locations. The second approach assigns half of the patches to each SMP node, so that SPEs process patches allocated in their local memory only.

	PS3	QS 20		
CPUs	N/A	node 0	both	both
memory	N/A	node 0	interleaved	distributed
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
5 SPEs/CPU	94	110	171	199
6 SPEs/CPU	95	110	174	205
7 SPEs/CPU	N/A	109	174	200
8 SPEs/CPU	N/A	109	173	200

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

The results show that at most 4 SPEs are sufficient to saturate the local memory bus. Using more SPEs might even decrease efficiency due to increased synchronization overhead. We can currently only guess why the Playstation 3 is not even half as fast as the QS20 blade: The Linux operation system is only running in a virtualized environment on the Playstation, further different kernel versions were in use.

Memory benchmarks have shown that the memory page size has quite some impact on the actual bandwidth. This is not surprising as a linear memory stream at 20 GiB/s will lead to a page fault about every 0.2 μs for the 4 kiB pages used by default! Most probably both PPUs can serve page faults on the blade simultaneously. We assume this is the reason for the very good single node performance on the QS20.

We see that allocating memory in an interleaved way improves performance of NUMA-unaware programs with few changes, but regarding the ccNUMA architecture by diligently assigning data and work is worthwhile.

Blood Vessel Geometry

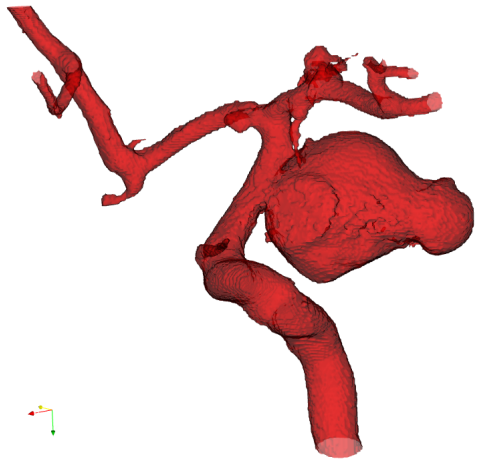


Figure 7: Real intracranial vessel geometry with a saccular aneurysm.

Finally we want to evaluate the performance for simulating blood flow in a real-world geometry depicted in Figure 7. The data set of $250 \times 250 \times 220$ voxels 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.

We compare our simulation with a performance optimized code for complex irregular geometries

	HP DL140G3	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

Table 4: Aneurysm geometry MFLUPS performance.

developed within the *International Lattice Boltzmann Development Consortium* (LB–DC) [2]. The resulting data-structure in this code is a 1-D list containing the density distribution values of all fluid lattice cells (for the current and the next time step) as well as a second 1-D list with information about the adjacency of the lattice cells used during propagation. This allows 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 ccNUMA architecture that connects two dual-core Intel Xeon 5160 processors at 3 GHz, each one having a theoretical peak bandwidth of 10.6 GB/s through its local memory bus. The memory requirements of this implementation for the given geometry are about 128 MiB.

Our implementation needs 1365 patches to represent the structure. About 48% of all allocated lattice cells represent fluid, so only every second update counts as FLUP. Around 116 MiB 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 SPU 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 performance only slightly. One advantage of our Cell implementation here is the higher bandwidth of the XDR memory. But even if normalized to the same bandwidth, the overhead of adjacency information in the unstructured domain description of the LB–DC code is larger than using simple structures and "wasting" half of the assigned lattice cells for such a moderately sparse structure.

Our optimized code is not portable on the one hand, and benchmarking code on CBEA hardware that does not exploit its features and only runs on the PPU makes no sense on the other. So we have to compare different implementations on different platforms, which does not allow more conclusions.

5 Conclusions and Future Work

Conclusions

We have shown the impressive potential of the Cell broadband architecture for blood flow simulation. To illustrate the performance measured in Section 4: The Sony Playstation 3, currently selling at less than 600 EUR, takes only about 11 minutes to simulate two heartbeats through that 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.

Future Work

In the future we plan to improve 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 straight-forward solution of this issue is to extend the solver for parallel execution on a cluster, which can easily be built using QS20 blades and Infiniband interconnects.

Another approach is to adopt strategies from cache blocking techniques [16] to the explicit data management on the SPUs. This might allow 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.

To improve accuracy, it is necessary to examine the consequence of the truncating rounding on the SPUs. Numerical experiments have shown that we can expect a slight loss of mass in the current implementation, but the change in velocity is even less. For the current application this seems acceptable, as fluid traverses the geometry usually fast and we only want to simulate short durations, but not for general application. We've found that simple approaches lead to a much better stability, but to find an optimal solution more detailed investigations are necessary.

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 is the most commonly used indicator to quantify the sensibility of endothelial cells to temporal fluctuations in wall shear stress [17].

6 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, further Professor Dörfler at the Department of Neuroradiology, University of Erlangen-Nuremberg, for advice and providing clinical data sets, and finally 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 giving us access to their JUICE Cell blade cluster.

References

[1] A. Artoli. *Mesoscopic Computational Haemodynamics*. PhD thesis, University of Amsterdam, 2003.

[2] L. Axner, J. Bernsdorf, T. Zeiser, P. Lammers, J. Linxweiler, and A. G. Hoekstra. Performance evaluation of a parallel sparse lattice Boltzmann solver. In *not yet published, submitted to proceedings of ICMMES-2007*, 2007.

[3] P. Bhatnagar, E. 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(3):511–525, May 1954.

[4] Cerebral (Brain) Aneurysms, 2006. www.brainavm.uhnres.utoronto.ca/malformations/cerebral_aneurysms_index.htm.

[5] A. Burleson and 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*, volume 76, pages 118–123, 1996.

[6] M. Castro, C. Putman, and J. Cebral. Computational Modeling of Cerebral Aneurysms in Arterial Networks Reconstructed from Multiple 3D Rotational Angiography Images. In A. Amini and A. Manduca, editors, *Medical Imaging 2005: Physiology, Function, and Structure from Medical Images*, volume 5746, number 1, pages 233–244. SPIE, 2005.

[7] Copenhagen: WHO Regional Office for Europe. Highlights on health in Germany 2004. Technical report, World Health Organization, 2006.

[8] J. Goetz. Simulation of bloodflow in aneurysms using the Lattice Boltzmann method and an adapted data structure. Technical report, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2006.

[9] X. He and L.-S. Luo. Lattice Boltzmann model for the incompressible Navier-Stokes equations. *J. of Stat. Phys.*, 88:927–944, 1997.

[10] D. Holdsworth, C. Norley, R. Fraynek, D. Steinman, and B. Rutt. Characterization of common carotid artery blood-flow waveforms in normal human subjects. *Physiological Measurement*, 20:219–240, 1999.

[11] IBM. *Cell Broadband Engine Architecture*, Oct. 2006.

[12] IBM. *Cell Broadband Engine Programming Tutorial*, May 2007.

[13] K. Iglberger. Performance Analysis and Optimization of the Lattice Boltzmann Method in 3D. Bachelor thesis, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2003.

[14] K. Kayembe, M. Sasahara, and F. Hazama. Cerebral aneurysms and variations in the circle of Willis. *Stroke*, 15(5):846–850, 1984.

[15] C. Körner, T. Pohl, U. Rüde, N. Thürey, and T. Zeiser. Parallel Lattice Boltzmann Methods for CFD Applications. In A. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes for Computational Science and Engineering*, chapter 5, pages 439–465. Springer Verlag, 2005.

[16] M. Kowarschik. *Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchical Memory Architectures*. Number 13 in Advances in Simulation. SCS Publishing House, 2004.

[17] D. Ku, D. Giddens, D. Philips, and D. Strandness. Hemodynamics of the normal human carotid bifurcation: in vitro and in vivo studies. *Ultrasound Med Biol*, 11:13–26, 1985.

[18] T.-M. Liou and 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*, volume 23, number 4, pages 133–148. National Science Council, Republic of China, 1999.

[19] R. Mei, W. Shyy, D. Yu, and L.-S. Luo. Lattice Boltzmann Method for 3-D Flows with Curved Boundary. *Journal of Computational Physics*, 161:680–699, 2000.

[20] A. Molyneux, R. S. Kerr, L.-M. Yu, M. Clarke, M. Sneade, J. Yarnold, and P. Sandercock. International subarachnoid aneurysm trial (ISAT) of neurosurgical clipping versus endovascular coiling in 2143 patients with ruptured intracranial aneurysms: a randomised comparison of effects on survival, dependency, seizures, rebleeding, subgroups, and aneurysm occlusion. *The Lancet*, 366:809–817, 2005.

[21] Y. H. Qian, D. D'Humières, and P. Lallemand. Lattice BGK Models for Navier-Stokes Equation. *Europhysics Letters (EPL)*, 17(6):479–484, 1992.

[22] G. Rinkel, M. Djibuti, A. Algra, and J. van Gijn. Prevalence and Risk of Rupture of Intracranial Aneurysms : A Systematic Review. *Stroke*, 29(1):251–256, 1998.

[23] C. Ruge. *Altersabhängigkeit der Hirndurchblutung im Erwachsenenalter: Eine farbduplexsonographische Studie*. PhD thesis, Eberhard-Karls-Universität zu Tübingen, 2003.

[24] N. Thürey. *Physically based Animation of Free Surface Flows with the Lattice Boltzmann Method*. PhD thesis, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation, 2007.

[25] J. Toelke. Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture. *Computing and Visualization in Science*, 2007.

[26] M. Wermer, I. van der Schaaf, A. Algra, and G. Rinkel. Risk of Rupture of Unruptured Intracranial Aneurysms in Relation to Patient and Aneurysm Characteristics: An Updated Meta-Analysis. *Stroke*, 38(4):1404–1410, 2007.

[27] D. Yu, R. Mei, L.-S. Luo, and W. Shyy. Viscous flow computations with the method of lattice Boltzmann equation. *Progress in Aerospace Sciences*, 39:5, 2003.