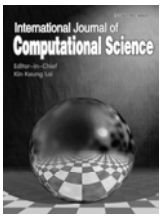
International Journal of Computational Science
1992-6669 (Print) 1992-6677 (Online) www.gip.hk/ijcs

2008, Vol. 2, No. 4, 437-449.



Parallel Lattice Boltzmann Flow Simulation on A Low-Cost PlayStation3 Cluster

Ken-ichi Nomura[1], Simon W. de Leeuw[2], Rajiv K. Kalia[1], Aiichiro Nakano[1]*, Liu Peng[1], Richard Seymour[1], Lin H. Yang[3], Priya Vashishta[1]

[1] Collaboratory for Advanced Computing and Simulations, Department of Computer Science, Department of Physics & Astronomy, Department of Chemical Engineering & Materials Science, University of Southern California, Los Angeles, CA 90089-0242, USA
{knomura,rkalia,anakano,liupeng,rseymour,priyav}@usc.edu

[2] Department of Chemistry, University of London, UK, and Theoretical Chemistry, LIC, University of Leiden, The Netherlands
leeuwdes@chem.leidenuniv.nl

[3] Physics/H Division, Lawrence Livermore National Laboratory, Livermore, CA 94551, USA
lyang@llnl.gov

Abstract. A scalable parallel algorithm has been designed to perform large-scale flow simulations based on the lattice Boltzmann method. The algorithm combines hierarchical spatial decomposition and a critical section-free, dual representation to expose maximal concurrency and data locality, thereby achieving isogranular parallel efficiency of 0.977 on 65,536 IBM BlueGene/L nodes. A hybrid thread + message passing programming is employed to implement the algorithm on a low-cost (~5,000 US dollars) Linux cluster consisting of 9 PlayStation3 consoles (based on the Cell Broadband Engine architecture) connected via a Gigabit Ethernet switch. The program achieves high multithreading parallel efficiency (0.882) on 6 Synergistic Processing Elements (SPEs) and performance improvement of factor 13.2 over a conventional PowerPC processor within each PlayStation3 console. Despite the limited bandwidth of the low-cost Ethernet switch, the program achieves reasonable (0.704) inter-console parallel efficiency.

* Corresponding Author. Email: anakano@usc.edu.

Keywords: Lattice Boltzmann method, flow simulation, parallel computing, hybrid thread + message passing programming, spatial decomposition, critical section-free dual representation, PlayStation3 cluster, Cell Broadband Engine architecture.

1 Introduction

Gaming consoles such as PlayStation3 can potentially be used as a low-cost parallel scientific computing platform [1]. PlayStation3 provides powerful computation capacity with a central processing unit (CPU) based on the Cell Broadband Engine (CBE) architecture designed by Sony, Toshiba and IBM [2], and a customized NVIDIA graphics processing unit (GPU). The CBE consists of a traditional microprocessor called Power Processor Element (PPE) and eight single-instruction multiple-data (SIMD) co-processors called Synergistic Processor Elements (SPEs) in a single chip. At a low-cost of ~$500, the peak performance of a PlayStation3 is 230 single-precision gigaflops on CPU (1 PPE + 8 SPEs) and 1.8 single-precision teraflops on CPU + GPU. We have built a Linux cluster consisting of 9 PlayStation3 connected via a Gigabit Ethernet switch at a price of less than $5,000. The peak performance of the cluster is 2.1 teraflops on CPUs and 16.2 teraflops on CPUs + GPUs. The PlayStation3 cluster is thus a low price/performance computing platform for a small research group.

PlayStation3 is also an ideal application test bed to prepare for the coming many-core era. Computer industry is facing a historical shift, in which Moore's law due to ever increasing clock speeds has been subsumed by increasing numbers of cores per microchip [3, 4]. Intel has already demonstrated an experimental 80-core microchip that achieved a teraflops with only 62W of power, and the number of cores per microchip is expected to double at each generation, reaching a thousand in 10 years. With a PPE and 8 SPEs on a microchip, CBE provides a glimpse of the coming era of many-core processors.

The many-core revolution will mark the end of the free-ride era (i.e., legacy software will run faster on newer chips), resulting in a dichotomy—subsiding speed-up of conventional software and exponential speed-up of scalable parallel applications [5]. Recent progresses in high-performance technical computing have identified key technologies for parallel computing with portable scalability. An example is an embedded divide-and-conquer (EDC) algorithmic framework to design linear-scaling algorithms for broad scientific and engineering applications based on spatiotemporal locality principles [6]. The EDC framework maximally exposes concurrency and data locality, thereby achieving reusable "design once, scale on new architectures" (or metascalable) applications. It is expected that such metascalable algorithms will continue to scale on future many-core architectures.

Despite the promise of PlayStation3 clusters and metascalable algorithms, their viability for real-life scientific applications is yet to be tested. This paper presents an efficient parallel implementation of a flow simulation based on the lattice Boltzmann method (LBM) [7] on our PlayStation3 cluster. The simulation features data locality, a large number of floating-point operations

per memory copy, and ease of parallelization, while handling complex geometry and multiphase flow.

A number of parallel implementations of LBM have been reported [8-10], incorporating complex solid boundaries [11] and inhomogeneous geometry [12]. The LBM has also been implemented on a Grid of distributed parallel computers [13] and GPU [14], but not on CBE. Here, we design a scalable parallel LBM (pLBM) algorithm that combines hierarchical spatial decomposition and a critical section-free, dual representation. Our LBM implementation adopting double buffering and thread interleaving has a great advantage in reducing data communication and achieving load balancing to make maximal use of the CBE architecture. Using hybrid thread + message passing programming, the pLBM algorithm is further implemented on a cluster of multicore processors (PlayStation3 consoles).

This paper is organized as follows. Section 2 describes the pLBM algorithm and its implementation on the PlayStation3 cluster, and benchmark results are presented in Section 3. Conclusions and future directions are contained in Section 4.

2 Parallel Lattice Boltzmann Method for Flow Simulation

2.1 Lattice Boltzmann Method

The essential quantity in the LBM [7] is a density function (DF) $f_i(\vec{x},t)$ on a discrete lattice $\vec{x}=(j\Delta x,k\Delta y,l\Delta z)$ ($j\in[1,N_x],k\in[1,N_y],l\in[1,N_z]$) with discrete velocity values $\vec{e}_i$ ($i\in[0,N_v-1]$) at time t. Here, each $\vec{e}_i$ points from a lattice site to one of its N_v near-neighbor sites. N_x, N_y and N_z are the numbers of lattice sites in the x, y and z directions, respectively, with Δx, Δy and Δz being the corresponding lattice spacings, and N_v (= 18) is the number of discrete velocity values. From the DF, we can calculate various physical quantities such as fluid density $\rho(\vec{x},t)$ and velocity $\vec{u}(\vec{x},t)$:

$$\rho(\vec{x},t)=\sum_i f_i(\vec{x},t)\,, \tag{1}$$

$$\rho(\vec{x},t)\vec{u}(\vec{x},t)=\sum_i \vec{e}_i f_i(\vec{x},t)\,. \tag{2}$$

The time evolution of the DF is governed by the Boltzmann equation in the Bhatnagar-Gross-Krook (BGK) model. The LBM simulation thus consists of a time-stepping iteration, in which collision and streaming operations are performed as time is incremented by Δt at each iteration step:

$$\text{Collision: } f_i(\vec{x},t^+)\leftarrow f_i(\vec{x},t)-\frac{1}{\tau}\left(f_i(\vec{x},t)-f_i^{\text{eq}}(\rho(\vec{x}),\vec{u}(\vec{x}))\right), \tag{3}$$

$$\text{Streaming: } f_i(\vec{x}+\vec{e}_i, t+\Delta t) \leftarrow f_i(\vec{x}, t^+). \tag{4}$$

In Eq. (4), the equilibrium DF is defined as

$$f_i^{\text{eq}}(\rho, \vec{u}) = \rho(A + B(\vec{e}_i \cdot \vec{u}) + C(\vec{e}_i \cdot \vec{u})^2 + D\vec{u}^2) \tag{5}$$

where A, B, C and D are constants, and the time constant τ is related to the kinematic viscosity ν through, $\nu = (\tau - 1/2)/3$.

It should be noted that the collision step involves a large number of floating-point operations that are strictly local to each lattice site, while the streaming step contains no floating-point operation but solely memory copies between nearest-neighbor lattice sites.

2.2 Parallel Lattice Boltzmann Method (pLBM) Algorithm on PlayStation3

We have designed a parallel lattice Boltzmann method (pLBM) algorithm for a cluster of multicore processors, such as a PlayStation3 cluster. The pLBM algorithm combines hierarchical spatial decomposition and a critical section-free, dual representation, and it is implemented using hybrid thread + message passing programming.

As a specific example, we use a Linux cluster consisting of PlayStation3 consoles. Within each PlayStation3 console, a main program runs on the PPE which spawns POSIX threads that run on multiple SPEs. Direct memory access (DMA) commands are used for data transfer between the main memory of the PPE and the local storage of the SPEs, since there is no access from SPEs to main memory. (Either SPE or PPE can issue DMA commands, which include a get command for retrieving memory contents, and a put command for writing data into memory.) For inter-console message passing, we use the message passing interface (MPI). The hybrid thread + message passing programming thus combines: (1) Inter-console parallelization with spatial decomposition into domains based on message passing; and (2) intra-console parallelization through multithread processing of interleaved rows of the lattice within each domain.

The pLBM algorithm consists of three functions: collision, streaming, and communication. The total simulation system Ω is decomposed into several sub-domains Ω_i, where $\Omega = \cup_i \Omega_i$, and each domain is mapped onto a processor (see Fig. 1). The collision and streaming functions update DFs on a single domain, while the communication function is responsible for inter-domain DF migrations. To simplify discussion, Fig. 1 shows a schematic of a 2-dimentional system (the actual implementation is for 3 dimensions). Here, the white squares denote open nodes that have DFs, the black squares denote closed nodes that represent obstacles (and hence no flow), and the gray squares denote buffer nodes that hold buffer DFs for inter-domain communication, which are initialized with the corresponding geometry information (open or closed) in neighbor domains at the beginning of simulation. In the 2-dimensional example, a single domain consists of $N_x \times N_y$ nodes, where N_x and N_y are the numbers of lattice sites in the x and y directions, respectively. Each domain is augmented with a surrounding buffer layer of one lattice spacing, which is used for

inter-domain DF migrations. A boundary condition is imposed for DFs propagating toward the closed nodes: reflecting DFs propagation into the closed nodes toward the opposite direction.

In the following, we first present multicore parallel algorithms of collision and streaming functions, which are local within each domain. Subsequently, inter-processor parallelization based on spatial decomposition is described.

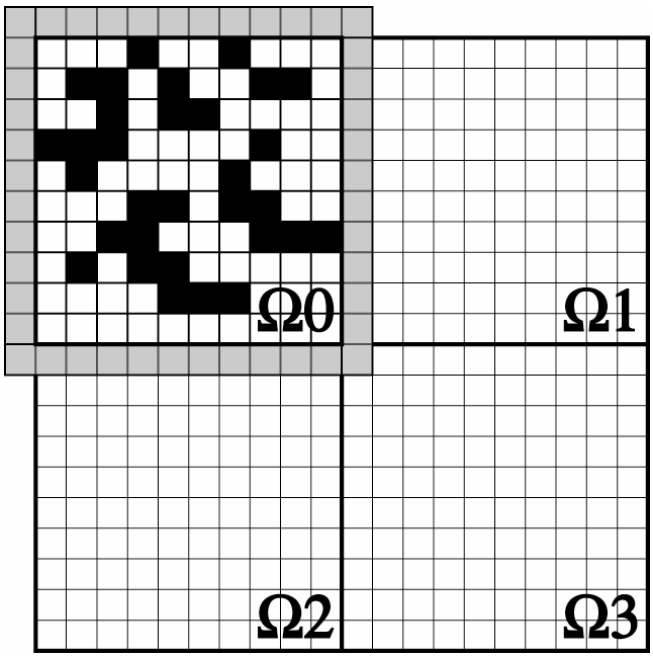


Fig. 1. Schematic of spatial decomposition in 2 dimensions with 4 domains. White squares are open lattice sites that have the DFs of flow particles. Black squares represent obstacles, where flow does not exist. Gray squares are buffer sites, where some of the DFs move in after streaming.

Collision

It is a challenging task to design a parallel algorithm due to CBE hardware restrictions. Six SPE programs can be simultaneously performed using POSIX threads on PlayStation3 (only 6 SPEs out of 8 are available for user programming). As mentioned in previous researches, the partitioning of work among the SPEs for load balancing is crucial to high performance [15]. For optimal load balancing, we parallelize by first diving the simulation problem into a large number (N_x) of chunks, where chunk ID j ($j \in [0, N_x - 1]$) processes lattice sites $\vec{x} = \left((j+1)\Delta x, k\Delta y, l\Delta z\right)$ ($k \in [1, N_y], l \in [1, N_z]$). Here, we use N_x, N_y and N_z to denote the numbers of lattice sites *per domain* in the x, y and z directions, respectively. We then interleavingly assign chunks to threads, i.e., chunk ID j is assigned to SPE with thread ID j mod N_{thread} ($j \in [0, N_{thread}-1]$). In our case, the number of threads N_{thread} is 6, so chunk 0 and chunk 6 are assigned to SPE 0, while chunk 1 and chunk 7 are assigned to SPE 1. In Fig.

2(a), the area enclosed by the dotted lines shows the computational task assigned to the first thread with thread ID 0.

One problem in the interleaved thread parallelization is that multiple threads may update a common lattice site. To avoid such a critical section, we have designed a double-layered DF consisting of two floating-point arrays *DF0* and *DF1*, shown in Fig. 2(b). (In Eqs. (3) and (4), $f_i(\vec{x},t)$ and $f_i(\vec{x},t^+)$ denote *DF0* and *DF1*, respectively.) In each LBM loop, the collision subroutine transfers DFs from the array *DF0* to local store on SPE, updates the DFs, and subsequently copies it back to the array *DF1*. The pseudo-code of collision subroutine is given in Table 1, where *fetchAddrData* is the address for a DMA get operation from *DF0* to local storage of SPE, *fetchAddrFlag* is the address for DMA get from main memory to local storage of SPE, and *putAddrData* is the address for DMA put from *DF1* to main memory. In the table, *geom*(*i*,*j*) denotes the flags (open or closed) of the *j*-th cell in chunk *i*.

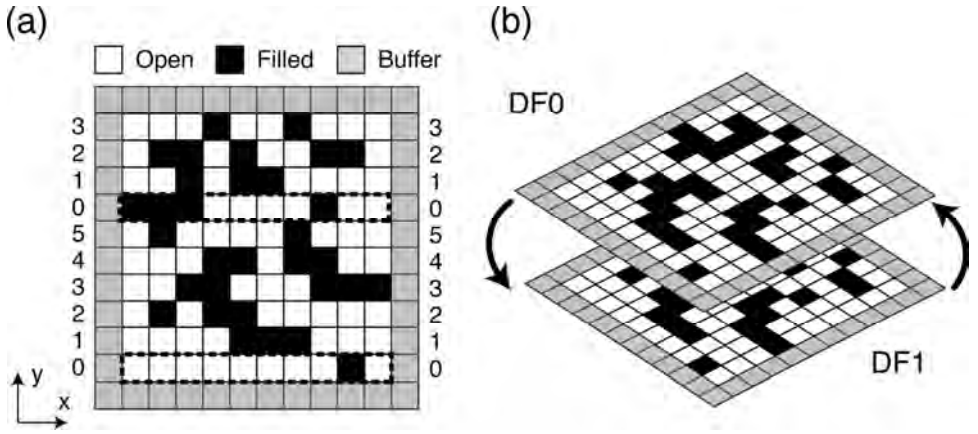


Fig. 2. (a) Schematic of a 2-dimensional system setup for each domain in spatial decomposition. White squares are open lattice sites that have the DFs of flow particles. Black squares represent obstacles in the system, where flow does not exist. Gray squares are buffer sites, where some of the DFs move in after streaming. The simulation system is divided into N_y computational chunks, each of which consists of N_yN_z lattice sites, and the chunks are interleavingly assigned to SPEs. The numerals show thread ID responsible for each chunk. (b) Schematic of a double-layered DF calculation comprising of two floating point arrays *DF0* and *DF1*. The collision function reads DFs from the array *DF0* to do updates, and then store the updated information in the array *DF1*. Subsequently, the streaming function propagates DFs from the array *DF1* to the array *DF0*.

Table 1. Collision calculation algorithm within SPE

Input:

N_x, N_y, N_z	{number of LBM lattice sites in the *x, y* and *z* directions}
N_{thread}	{number of threads}
tID	{thread ID}
array *DF0* in PPE of size *N*	{array of density functions, where $N = N_x N_y N_z$}
array *geom*	{array of geometry flags}

Output:

array *DF1* in PPE of size *N*	{array of density functions}

Steps:

1 *chunkID* ← *tID*
2 *chunksize* ← N/N_x
3 **while** *chunkID* < N/N_x **do**
4 *fetchAddrData* ← address of *DF0* + *chunkID*×*chunksize*
5 *fetchAddrFlag* ← address of *geom* + *chunkID*×*chunksize*
6 *putAddrData* ← address of *DF1* + *chunkID*×*chunksize*
7 initiate DMA transfers to get data
8 fetch data from *DF0* and *geom*
9 wait for the data
10 **for** *j* ← 0 to *chunksize*–1
11 $\rho(\vec{x},t) \leftarrow \sum_i f_i(\vec{x},t)$ {see Eq. (1)}
12 $\vec{u}(\vec{x},t) \leftarrow \rho^{-1}(\vec{x},t)\sum_i \vec{e}_i f_i(\vec{x},t)$ {see Eq. (2)}
13 $f_i(\vec{x},t^+) \leftarrow f_i(\vec{x},t) - [f_i(\vec{x},t) - f_i^{eq}(\rho(\vec{x}),\vec{u}(\vec{x}))]/\tau$ {see Eq. (3)}
14 **if** *geom*(*chunkID*, *j*) is open **then** update density functions;
15 initiate DMA put for the computed results
16 *chunckID* ← *chunkID* + N_{thread}
17 synchronize using inter-SPE communication

Streaming

The streaming function propagates the DFs according to their flow directions, see Eq. (4). Here the DFs are copied from main memory to main memory, between array *DF1* and array *DF0* in Fig. 2(b). Before propagating DFs, a boundary condition such as reflection rule must be considered according to the simulation geometry. In the case of a static geometry, where the relation between source and destination lattice sites does not change, we avoid repeated computing of boundary condition by defining another array to keep the indices of destination lattice sites for each DF,

which significantly speeds up the streaming function. Furthermore, we find that the hardware-supported threads on PPE improve the performance of the complicated memory copy. We use two POSIX threads, each of which is responsible for half of the data transfer. This improves the performance of the streaming computation by 20-30%.

Communication

After the streaming function, some of the DFs move out of their domains. In the communication function, DFs in the buffer lattice sites migrate to proper destination domains. Figure 1 shows a schematic of the domain decomposition consisting of four sub-domains Ω0-Ω3. We employ a 6-way dead-lock free communication scheme, in which data transfer is completed in 6 steps. The inter-domain communication is implemented with MPI.

Detail of the above migration step is as follows. After a streaming function call, the DF values of the layer of buffer lattice sites (shown in gray in Figs. 1 and 2) are transferred to neighbor domains that share boundaries. The interdomain communications take place in positive and negative directions along the x, y and z axes, respectively (i.e. in 6 communication steps) in a three-dimensional system. In a single communication step, a data packet is constructed from the DF values of the buffer lattice sites in the communication direction. After exchanging the data packets between the neighbor domains, the DF values of destination lattice sites are updated. This communication procedure automatically takes care of communications between domains in diagonal directions (e.g. Ω0 and Ω3 in Fig. 1), which reduces the number of interdomain communications. In addition, all interdomain communications follow a simple rule to make them deadlock free. Each domain has a directional parity, either 0 or 1, in the x, y and z directions. In communication in the positive direction, odd-parity domains first send data packets that are received by destination domains with even parity, and vice versa in the negative direction.

3 Results

3.1 Experimental Platforms

We have implemented the pLBM algorithm on a cluster of 9 PlayStation3 consoles connected via a Gigabit Ethernet switch. To compare the communication performance of the low-cost PlayStation3 cluster, we have also implemented pLBM on the 131,072-processor IBM BlueGene/L computer at the Lawrence Livermore National Laboratory (LLNL). On the BlueGene/L, multi-threading is not supported and hence thread parallelism of pLBM is disabled. The two computing platforms are described below.

PlayStation3 Cluster (Cell Broadband Engine)

9 PlayStation3 consoles are connected via a Gigabit Ethernet switch, where each PlayStation3 contains: (1) a 3.2 GHz 64-bit RISC PowerPC processor (PPE) with 32KB L1 and 512KB L2 caches and 256MB main memory; and (2) eight 3.2GHz 32-bit SPEs with 256KB of local store (LS) and Memory Flow Controller (MFC). The PPE, SPEs, and main memory are interconnected by a fast internal bus called the Elemental Interface Bus (EIB), with the peak bandwidth of 204.8GB/s, while the memory and I/O interface controller (MIC) supports a peak bandwidth of 25 GB/s inbound and 35GB/s outbound. Each PlayStation3 has a Gigabit Ethernet port.

We have installed a Fedora Core 6 Linux OS distribution with libraries and infrastructure to support the IBM Cell Software Development Kit (SDK) version 2.1. The SDK offers an IBM compiler and the GNU compiler collection for the Cell processor. Message Passing Interface (MPI) is installed as in a standard Linux cluster. We use the Cell SDK for instruction-level profiling and performance analysis of the code. The code is compiled using GNU C compiler (gcc) with optimization option '-O3' and MPI library version 1.2.6.

BlueGene/L

The BlueGene/L at the LLNL consists of 65,536 computational nodes (CNs), each of which has two PowerPC 400 processors (131,072 processors in total) with 700MHz clock speed. On a single CN, the two processors share 512MB memory. Each processor has 32KB instruction/data cache, 2MB L2 cache, and 4MB L3 cache. The theoretical peak performance is 2.8 gigaflops per processor. Two types of interconnection (3D torus and tree topologies) are designed for distinct purposes. The 3D torus network is used mostly for common (e.g., point-to-point) communications, while the tree network is optimized for collective communications. The interconnection bandwidths are 175MB/s and 350MB/s per link, respectively.

3.2 Scalability Test Results

We first test the intra-processor scalability of pLBM based on multithreading on a single PlayStation3 console. Figure 3(a) shows the running time for the collision function as a function of the number of SPEs S from 1 to 6 for a simulation system with 64^3 lattice sites. Figure 3(b) shows the corresponding strong-scaling speed-up, i.e., the running time on a single SPE divided by that on S SPEs. The algorithm scales nearly linearly with the number of SPEs. On 6 SPEs, the speed-up is 5.29, and the parallel efficiency (defined as the speed-up divided by S) is 0.882.

Figure 4 shows the inter-processor parallel efficiency of pLBM on the PlayStation3 cluster and BlueGene/L. Here, we scale the number of lattice sites linearly with the number of processors: 64^3P lattice sites on P processors. The weak-scaling speed-up is the running time on 1 processor divided by that on P processors, and the parallel efficiency is the speed-up divided by P. Figure

4(a) shows nearly perfect parallel efficiency, 0.977, on 65,536 processors of the BlueGene/L. This is due to the high-end 3D toroidal network. (We have used a co-processor mode, in which one of the two processors in each CN performs computation, while the other processor manages communication.) On the other hand, the parallel efficiency of the PlayStation3 cluster is rather low (0.704 on 8 consoles), very likely due to the small bandwidth and large latency of the low-price Gigabyte Ethernet switch. Moreover, currently only 6 SPEs are available for general use on PlayStation3. One SPE is disabled considering wafer yield rate and cost increase, while another is used for the virtualization layer (so called hypervisor or GameOS). Hardware such as network adapter can be accessed through the virtualization layer, which generates additional network latency. Therefore, besides the low performance of Ethernet itself, the increase in system call time through the virtualization layer may also be responsible for the performance degradation.

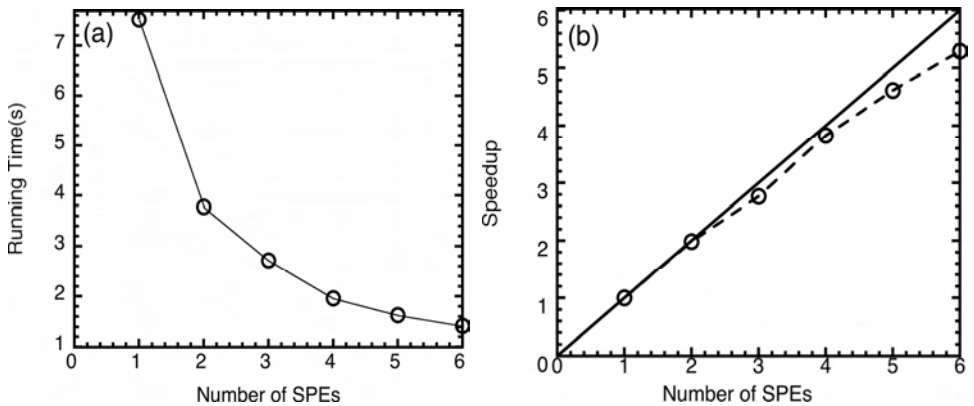


Fig. 3. (a) Running time for the pLBM flow simulation involving 64^3 lattice sites on a single PlayStation3 console as a function of the number of SPEs. (b) Strong-scaling speed-up of the pLBM algorithm (circles) on a single PlayStation3 console as a function of the number of SPEs. The solid line shows the ideal speed-up.

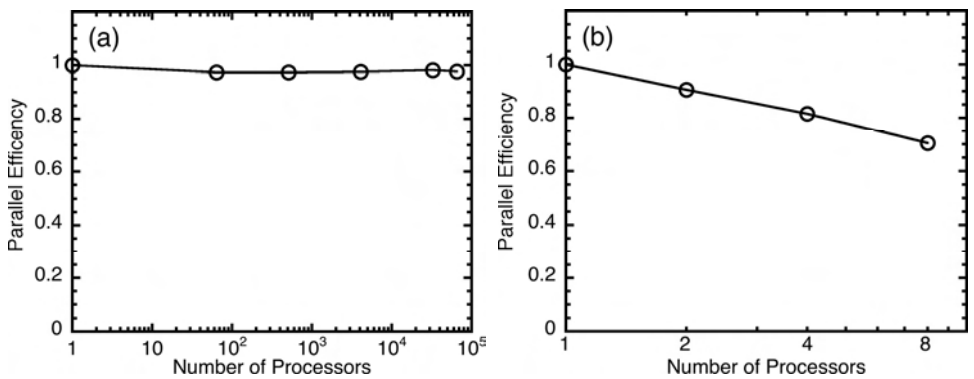


Fig. 4. Weak-scaling parallel efficiency of the pLBM flow simulation as a function of the number of processors, where each processor is in charge of 64^3P lattice sites on (a) BlueGene/L and (b) PlayStation3 cluster.

To assess the comparative performance of the PlayStation3 cluster over a conventional Linux cluster, Fig. 5(a) compares the running time for the pLBM code as a function of the problem size ranging from 8^3 to 32^3 lattice sites on an 8-console PlayStation3 cluster with that of an 8-node PowerPC cluster. The latter is measured by running the MPI-only pLBM program (which has been used on the BlueGene/L) on the PlayStation3 cluster by using only PPE. The corresponding performance improvement, i.e. the running time on the PlayStation3 cluster divided by that on the PowerPC cluster, is plotted in Fig. 5(b). Evidently, the PlayStation3 cluster outperforms the PowerPC cluster for all the tested problem sizes, and the performance enhancement is an increasing function of the problem size. This indicates that the DMA efficiency increases with the data size. For the largest problem size, the performance enhancement is 13.2.

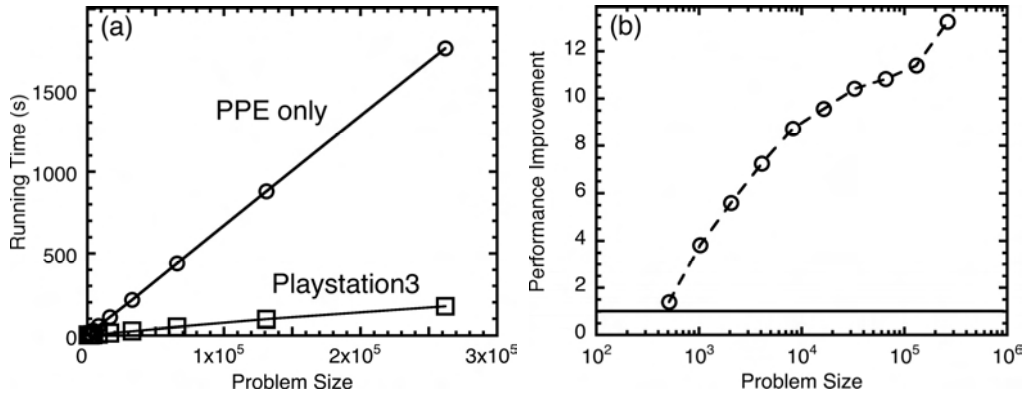


Fig. 5. (a) Running times for pLBM on the PlayStation3 (squares) and PowerPC clusters (circles) as a function of the problem size. (b) Performance enhancement of the PlayStation3 cluster over the PowerPC cluster for different problem sizes. The horizontal solid line signifies the equal speed of the PlayStation3 and PowerPC clusters.

4 Conclusion

The performance test results discussed above prove the applicability of low-cost Playstation3 clusters to practical scientific computing applications. Our parallel lattice Boltzmann method code for flow simulation has achieved high multithreading parallel efficiency (0.882) on 6 SPEs within each PlayStation3 console. Despite the limited bandwidth of the low-price Ethernet switch, the pLBM code achieves reasonable (0.704) inter-console parallel efficiency. In addition, the PlayStation3 cluster outperforms a conventional PowerPC-based cluster by a factor of 13.2. This is largely due to the multicore-scalable pLBM algorithm, which maximally exposes concurrency and data locality. Designing such metascalable algorithms for broad applications is crucial in the coming many-core era.

We have applied the pLBM code on the PlayStation3 cluster to simulate fluid flow in fractured glass. Figure 6 visualizes our pLBM simulation of fluid flow through fractured silica glass, where

the fractured surface has been prepared through voxelation of atomistic simulation data [16]. Such a flow simulation in a complex geometry is important in many areas, e.g., for maximizing oil recovery in petroleum industry.

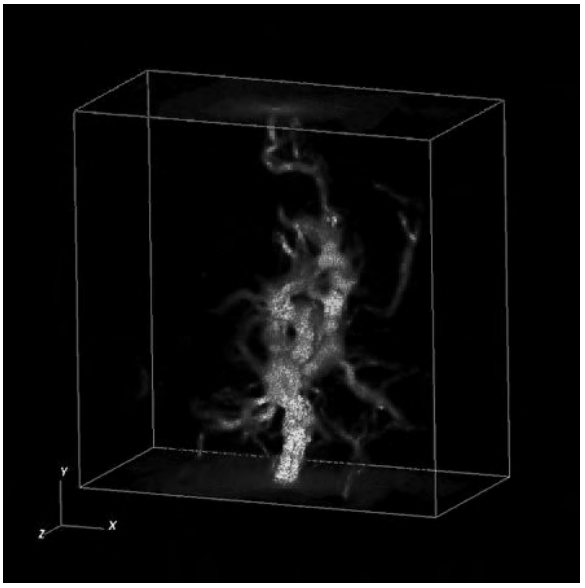


Fig. 6. Visualization of a pLBM simulation of fluid flow in fractured silica on the PlayStation3 cluster. Here the magnitude of the fluid velocity is color-coded.

Acknowledgements

This work was partially supported by ARO, Chevron—CiSoft, DOE, DTRA, and NSF. Numerical tests were performed using the Playstation3 cluster at the Collaboratory for Advanced Computing and Simulations of the University of Southern California and the 131,072-processor IBM BlueGene/L computer at the Lawrence Livermore National Laboratory.

References

1. Buttari A, Luszczek P, Kurzak J, Dongarra J, Bosilca G.: SCOP3: A Rough Guide to Scientific Computing on the PlayStation 3. Knoxville, University of Tennessee. (2007)
2. Johns CR, Brokenshire DA: Introduction to the Cell Broadband Engine Architecture. *IBM Journal of Research and Development* (2007) 51:503.
3. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Pishker WL, Shalf J, Williams SW, Yelick KA.: The Landscape of Parallel Computing Research: A View from Berkeley.

Berkeley, University of California. (2006)

4. Shalf J.: The New Landscape of Parallel Computer Architecture. *Journal of Physics: Conference Series* (2007) 78:012066.
5. Dongarra J, Gannon D, Fox G, Kennedy K.: The Impact of Multicore on Computational Science Software. *CTWatch Quarterly* (2007) 3:11.
6. Nakano A, Kalia RK, Nomura K, Sharma A, Vashishta P, Shimojo F, Duin ACTv, W. A. Goddard I, Biswas R, Srivastava D, Yang LH.: De Novo Ultrascale Atomistic Simulations on High-end Parallel Supercomputers. *International Journal of High Performance Computing Applications* (2008) 22:113.
7. Ladd AJC, Verberg R.: Lattice-Boltzmann Simulations of Particle-fluid Suspensions. *Journal of Statistical Physics* (2001) 104:1191.
8. Amati G, Succi S, Piva R.: Massively Parallel Lattice-Boltzmann Simulation of Turbulent Channel Flow. *International Journal of Modern Physics C* (1997) 8:869.
9. Chen S, Doolen GD.: Lattice Boltzmann Method for Fluid Flows. *Annual Review of Fluid Mechanics* (1998) 30:329.
10. Derksen JJ, Kooman JL, van den Akker HEA: Parallel Fluid Flow Simulations by Means of a Lattice-Boltzmann Scheme. *High-Performance Computing and Networking* (1997) 1225:524.
11. Desplat JC, Pagonabarraga I, Bladon P.: LUDWIG: A Parallel Lattice-Boltzmann Code for Complex Fluids. *Computer Physics Communications* (2001) 134:273.
12. Pan CX, Prins JF, Miller CT.: A High-performance Lattice Boltzmann Implementation to Model Flow in Porous Media. *Computer Physics Communications* (2004) 158:89.
13. Harting J, Venturoli M, Coveney PV.: Large-scale Grid-enabled Lattice Boltzmann Simulations of Complex Fluid Flow in Porous Media and under Shear. *Philosophical Transactions of the Royal Society of London Series a-Mathematical Physical and Engineering Sciences* (2004) 362:1703.
14. Li W, Wei XM, Kaufman A.: Implementing Lattice Boltzmann Computation on Graphics Hardware. *Visual Computer* (2003) 19:444.
15. Bader DA, Agarwal V.: FFTC: Fastest Fourier Transform for the IBM Cell Broadband Engine. *Proceedings of the International Conference on High Performance Computing (HiPC) IEEE*. (2007)
16. Chen Y, Lu Z, Nomura K, Wang W, Kalia RK, Nakano A, Vashishta P.: Interaction of Voids and Nanoductility in Silica Glass. *Physical Review Letters* (2007) 99:155506.