Parallel Lattice Boltzmann Simulation for Fluid Flow on Multicore Platform

Weibin Guo[1], Cheqing Jin[2], Jianhua Li[1], Gaoqi He[1]
[1] Department of Computer Science and Engineering
East China University of Science and Technology
[2] Software Engineering Institute
East China Normal University
Shanghai, China
e-mail: gweibin@ecust.edu.cn

***Abstract*—During the past fifteen years Lattice Boltzmann method has attracted much attention in the area of CFD, whereas it has also been recognized that it is both computationally demanding and memory intensive. In this paper we give a brief introduction to the equations for LBM and the basic features, describe the programming model in multicore computing environment, and outline the parallelization strategies. Moreover, we explore corresponding implementation on multicore platform. In order to demonstrate the efficiency, several benchmark steady fluid flow experiments have been performed. The results show that our approach achieves good performance.**

Keywords- Lattice Boltzmann method; Fluid flow; Parallel; Multicore; OpenMP

I. INTRODUCTION

The Lattice Boltzmann method (LBM) has evolved to a promising alternative to the well-established mesoscopic approaches based on finite elements/ volumes for computational fluid dynamics (CFD) simulations. The basic idea behind is to develop a simplified kinetic model that incorporates the essential physics and reproduces correct macroscopic averaged properties. As a result it possesses some unique advantages, e.g., simplicity - simplicity in the mesh used (Cartesian mesh) and in the operations performed (collision, advection and boundary update), extensibility – can be easily extended to simulate a wide range of flow problems, flexibility – deal flexible with complex boundary conditions. These properties make LBM can be viewed as a new paradigm to solve numerically, and an efficient method to model complex fluid systems [1,2,3].

One of the challenges of LBM is the requirement of huge computer resources. Employing LBM you model flows on computer, it often involve many complex mathematics and physics problems such as anomalistic structures, non-linear dynamics systems, active boundary, and strict restrict conditions, etc. These produce numerical simulations with high demands for computational power in terms of memory and speed [4,5]. There is a strong need to develop techniques for improving the LBM performance.

The appearance of multicore processors opens the doors of mainstream computing for parallel computing. Moore's law due to ever increasing clock speeds has been subsumed by increasing members of cores per microchip. Multicore processors deliver significantly greater computing power through concurrency compared to conventional single core processor chips. Future high performance computing (HPC) machines will almost certainly contain multicore chips, likely tied together into shared memory nodes as the machine building block [6,7]. It provides a natural programming paradigm, and this shift leads the integration of parallel programming standards for high-end shard-memory machine architectures into desktop programming environments.

It is therefore clear that multicore-based HPC environment is the possibility to fulfill the large computational requirements for simulating complex flows, and provides a reliable solver for LBM simulations. Williams et al. studied performance optimization of the LBM on multicore platforms [6]. Liu et al. presented a design of a unified parallel implementation of the LBM on several multicore platforms including a cluster of Cell-based PlayStation3 consoles and Compute Unified Device Architecture based implementations on GPUs [7]. Stuermer implemented the LBM on Cell [8]. Lacoursière described a hybrid block parallel method to approximately solve complementarity problems in real-time on multicore CPUs [9]. Donath comparison of different parallel lattice Boltzmann implementations on multi-core multi-socket systems [10].

In this paper, we present a parallel lattice-Boltzmann implementation for flow simulations. The code was designed to run on a multicore platform, which used MS Visual studio C++ 2005 and OpenMP as the develop tool.

In the next section, we introduce the underlying LBM scheme, and describe the parallelization of the LBM algorithm. Section 3 covers the OpenMP programming model as well as the implementation of the standard LBM algorithm. Several benchmark steady fluid flow problems are presented in Section 4. Finally, some conclusions are drawn in Section 5.

II. LATTICE BOLTZMANN METHOD AND ITS' PARALLELIZATION

A. Basics of the Lattice Boltzmann Method

For two-dimensional incompressible flow, the 9-velocity BGK model (i*d*2*q*9) is widely used. For this model, the

978-0-7695-3679-8/09 $25.00 © 2009 IEEE
DOI 10.1109/ICIE.2009.144

directions of the discrete velocity used in the model are given by $\mathbf{e}_0 = 0$, $\mathbf{e}_i = (\cos[(i-1)\pi/2], \sin[(i-1)\pi/2])$ for i=1:4, $\mathbf{e}_i = \sqrt{2}(\cos[(i-5)\pi/2+\pi/4], \sin[(i-5)\pi/2+\pi/4])$ for i=5:8. The evolution equation of the distribution function $g_i(\mathbf{x},t)$ reads [11]

$$g_i(\mathbf{x}+c\mathbf{e}_i\Delta t, t+\Delta t) = g_i(\mathbf{x},t) - \frac{1}{\tau}[g_i(\mathbf{x},t) - g_i^{(0)}(\mathbf{x},t)], \quad (1)$$

where $\mathbf{x}$ is a point in the discretized physical space and $c = \Delta x/\Delta t$ is the particle speed, Δx and Δt are the lattice spacing and the time step, respectively. τ is the dimensionless relaxation time. $g_i^{(0)}(\mathbf{x},t)$ is the equilibrium distribution function defined by

$$g_i^{(0)} = \lambda_i p + \omega_i[3\frac{(\mathbf{e}_i \cdot \mathbf{u})}{c} + 4.5\frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{c^2} - 1.5\frac{|\mathbf{u}|^2}{c^2}], \quad (2)$$

with the weighting factors ω_i are 4/9 for the rest particles (i = 0), 1/9 for i = 1, 2, 3, 4, and 1/36 for i = 5, 6, 7, 8. $\lambda_0 = -4\sigma/c^2$, $\lambda_i = \lambda/c^2$ for i=1:4, and $\lambda_i = \gamma/c^2$ for i=5:8. σ, λ, and γ are parameters satisfying $\lambda+\gamma=\sigma, \lambda+2\gamma=1/2$.

The fluid kinematic viscosity υ is determined by $\upsilon = \frac{(2\tau-1)}{6}\frac{(\Delta x)^2}{\Delta t}$. The macroscopic flow velocity and pressure are given by

$$\mathbf{u} = \sum_{i=1}^{8} c\mathbf{e}_i g_i \ , \quad p = \frac{c^2}{4\sigma}[\sum_{i=1}^{8} g_i + s_0(\mathbf{u})]. \quad (3)$$

B. Parallelization of the LBM Algorithm

Ease of implementation, parallelism, and computational efficiency are the major features of LBM. But, for realistic applications, the LBM is computationally very demanding, since it needs fine spatial resolution and small time steps. For example, three dimensional fluid flow simulations are known to be computationally extensive. In LBM, the evolution rule is the same for all cells and updating of the cells occurs simultaneously in discrete time steps. The core algorithm can be reduced to a few manageable subroutines, facilitating high performance computing. Thus, parallelization will be a suitable solver for flow LBM simulations [1,3].

The most natural approach to parallelize the LBM is by domain decomposition, where the computational domain is partitioned into smaller subdomains of desired size according to the specification of the available processors. Each processor performs computations on a certain subdomain and exchanges information with other nodes.

In a standard LBM code with fused stream/collide step, a time step can be performed in a single sweep over the computational domain [3]. The LBM code break down into two separate pieces operating on a set of distribution functions, a non-linear collision operator and a linear propagation operator. The computer-intensive collision step, $(g_i(\mathbf{x},t) - g_i^{(0)}(\mathbf{x},t))/\tau$, relaxation towards local equilibrium, and involves data local only to that spatial lattice sites they do not need any communication, the scheme is the inherent spatial locality, and allows concurrent. For the propagation step, $g_i(\mathbf{x}+c\mathbf{e}_i\Delta t, t+\Delta t) = g_i(\mathbf{x},t)$, which mainly deals with memory access, interaction between processors is necessary, i.e., at this step particle on a border node can move to a lattice point in the domain of a neighboring processor or vice versa. By using a ghost cell layer in the surrounding of the subdomain, the propagation step can be isolated from the data exchange step, and can perform in parallel. After the propagation step, the values in the ghost layer are sent to the neighboring processor.

Hence, the computation is independently carried out point-by-point in the LBM. Due to the local character of the LBM, the parallelization by simple domain decomposition is straightforward and brings good results concerning the parallel speed-up. Parallelization can be based on one- or two-dimensional domain partitioning in equal size, namely slice, and box decompositions [12].

III. LATTICE BOLTZMANN SIMULATION ON MULTICORE PLATFORM

A. Programming model in multicore computing environment

High end distributed and distributed shared memory platforms with many cores will be deployed in the coming years to solve flow problems. Their individual nodes will be heterogeneous multithreading, multicore systems, capable of executing many threads of control. The multicore system presents applications developers with many challenges. For example, the code will need to expose a sufficient amount parallelism, additional resource sharing (and contention) between threads that run on the same core [13].

Fortunately, the integration of the OpenMP parallel programming model into Microsoft Visual C++ 2005 provides possible way for multicore application that bring parallel computing to the desktop. OpenMP is essentially a comparatively recent standardization SMP development and practice. By using OpenMP, it is relatively easy to create parallel applications in C, C++, and FORTRAN [14,15,16]. As a shared-memory programming paradigm, OpenMP is suitable for parallelizing applications on simultaneous multithreaded (SMT) and multicore processors. OpenMP helps developers to create multithreaded applications more easily while retaining the look and feel of serial programming. It consists of a set of compiler directives and library routines. The compiler generates a multi-threaded code based on the specified directives. It allows multilevel loop nest parallelism, enhances support for nested parallelism and introduces tasks, which are conceptually placed into a pool of tasks for subsequent execution by an arbitrary thread.

B. Implementation of LBM

From the perspective of the application programmer, a multicore processor is an SMP on a chip and OpenMP programs just run nicely. Of course there is a lot more

sharing of resources, which may heavily impact performance. Sharing of caches can be a major advantage [17].

For LBM code, there is no temporal coherence or space coherence data. We can take advantage of the tight coupling of multi-core processors to derive work estimates for tasks with very low overhead. Shared-memory parallel systems are not as restrictive in terms of communication costs such that using a higher number of partitions can be considered. Hence, operating on shared-memory reduces the complexity of cores and on-chip communication, allowing more cores per transistors. In fact, the implementation of the LBM code is parallelized using OpenMP, using loop scheduling and chunk decomposition scheme to partite the whole lattice onto a 1-dimensional or 2-dimensional processor grid.

The program begins with a single master thread of execution. The master thread spawns teams of threads in response to OpenMP directives, which perform work in parallel. That is, each thread read the values of the current time step from the shared-memory, execute the relaxation and write the results back to a global array, as this can be done independently for all cells. So, OpenMP directives are inserted at key locations in the source code. The compiler interprets the directives and creates the necessary code to parallelize the indicated regions. Global error estimate was computed in each iteration step and used as a stopping criterion for these runs.

IV. EXPERIMENTS

As test cases we consider two benchmark fluid flow problems: (i) incompressible flow over a backward-facing step, and (ii) lib driven cavity flow. We tested these flows on Quad-Core computers Dell PowerEdge 2950 based on Intel Xeon CPU E5405 2.00 GHz with 1.99 GB main memory, 8.0 GB memory module support. On the software side we used the OpenMP, and all the benchmarks were also compiled by Microsoft Visual Studio .Net 2005.

A. Backward-facing step flow

The problem domain and boundary conditions of the incompressible backward-facing step flow considered in this study are summarized in Figure 1. The problem is geometrically defined by two infinite parallel plates between which a fluid flows with specified boundary conditions at the inlet and outlet of the channel, with a backward-facing step at the channel entrance.

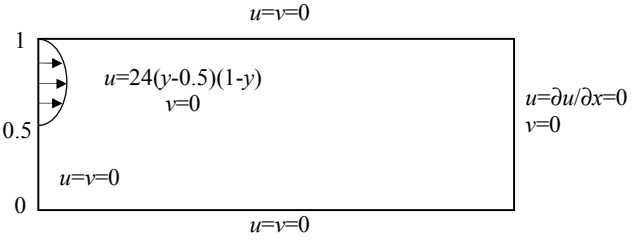

Figure 1. Computational domain configuration.

A characteristic of this flow is that there is one eddy near the floor and the ceiling, respectively [18]. The Reynolds number is defined based on the height of the channel h and the average entrance velocity. The extent of the computation domain in the x direction is $15h$. The lattice size was 600×40. Here, using the slice decomposition, the overall computational load was partitioned and assigned uniformly among 1, 2, and 4 cores, respectively. The streamlines for Re=800 are shown in Figure 2. The location of the recirculation regions near the floor and of the region near the ceiling is (3.25, 0.325) and (7.325, 0.850). The corresponding sizes of recirculation bubble are 5.775 and 4.712. The data are in agreement with experiments [19]. The experiment indicates that the running time reduces as the number of cores increase and the efficiency increases with the problem size.

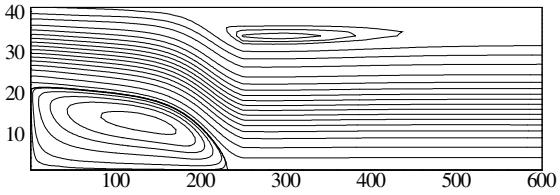

Figure 2. Contours of streamfunction for Re=800.

B. Driven Cavity Flow

As a benchmark, the configuration of lid driven cavity flow shown in Figure 3 considered here consists of a two-dimensional square cavity whose top plate moves from left to right with constant velocity, while the other three boundaries are fixed.

The simulation was carried out on a 256×256 lattice. In this example, using the box decomposition, the overall computational load was equipartitioned and assigned uniformly among 1, 2, and 4 cores, resulting in uniform balance of computational load. Figure 4 shows the contours of streamfunction at Re=3200.

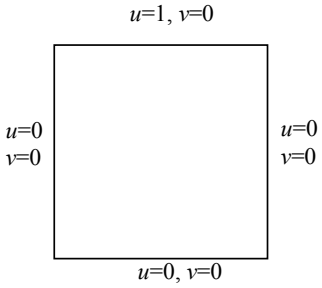

Figure 3. Configuration.

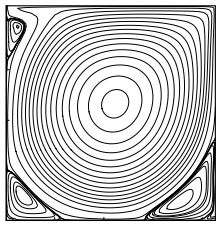

Figure 4. Contours of flow Streamfunction at Re=3200

The Reynolds number Re used in the problems is Re=LU/υ, where L is the height of the cavity, U is the velocity of the top plate. The result shows clearly the flow pattern, i.e. there are one center primary vortex and three first-class vortices, a pair of secondary ones of much smaller

strength develop in the lower corners of the cavity and a tertiary vortex appears in the lower right corner. The locations of the vortex, (0.518, 0.535) for primary vortex, (0.059, 0.910) for top left vortex, (0.081, 0.131) for lower left vortex and (0.831, 0.087) for lower right vortex, agree well with those of previous work [19].

V. CONCLUSIONS

Physically based simulation is an important component of many applications in current research areas of CFD. LBM is a remarkably effective computational tool for tackling complex fluid problems which can be extremely difficult via conventional methods. Multicore processors will dominate scientific computing in the near future. In this work we have presented techniques for LBM simulation for fluid flows on multi-core architectures. We applied domain partitioning to perform the computations in parallel, and introduced the implementation strategy on Intel quad-core machines using MS Visual studio C++ 2005 and OpenMP.

Results show that this approach can offer significant performance benefits on real scientific applications, and the speed-up also increases with the problem size. We therefore believe that multicore-based HPC systems will be an important tool in modeling complex CFD problems in the future. Next work will continue exploring parallelization for 3-dimensional flows on multicore systems.

ACKNOWLEDGMENT

This work was supported by Hi-Tech Research and Development Program of China (Grant No. 2006AA10Z315), MOE-Intel Model Curriculum Program, and by the National Natural Science Foundation of China (Grant No. 60703026, Grant No. 60803020).

REFERENCES

[1] S. Succi, The lattice Boltzmann equation for fluid dynamics and beyond, Oxford: Clarendon Press, 2001, pp.3–25.

[2] T. Zeiser, Flow simulation with lattice Boltzmann methods: Basics and recent enhancements, Karlsruhe: Bundesanstalt für Wasserbau, 2007, pp.15–23.

[3] C. Körner, T. Pohl, U. Rüde, N. Thürey, and T. Zeiser, "Parallel Lattice Boltzmann Method for CFD Applications," in Numerical Solution of Partial Differential Equations on Parallel Computers, LNCSE, vol. 51, A.M. Bruaset and A. Tveito, eds. 2006, pp. 439–465.

[4] K. Mattila, J. Hyvaluoma, and J. Timoneh, "Comparison of implementations of the lattice-Boltzmann method," Computers and Mathematics with Applications, vol. 55, 2008, pp. 1514–1524.

[5] K. Stratford and I. Pagonabarraga, "Parallel simulation of particle suspensions with the lattice Boltzmann method," Computers and Mathematics with Applications, vol. 55, 2008, pp. 1585–1593.

[6] S. Williams, J. Carter and L. Oliker, J. Shalf, and K. Yelick. "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms," Proc. IEEE Int. Symp. Parallel & Distributed Processing (IPDPS 2008), IEEE Press, Apr. 2008, pp. 1–14.

[7] L. Peng, K. Nomura, T. Oyakawa, R. K. Kalia, A. Nkano, and P. Vashishta. "Parallel Lattice Boltzmann Flow Simulation on Emerging Multi-core Platforms," Euro-Par 2008, LNCS, vol. 5168, 2008, pp. 763–777.

[8] M. Stuermer, "Fluid simulation using the Lattice Boltzmann Method on the Cell Processor", Vortrag: inladung, Zentralinstitut fur Angewandte Mathematik des Forschungszentrum Juelich, 2007.

[9] C. Lacoursière, "A Parallel Block Iterative Method for Interactive Contacting Rigid Multibody Simulations on Multicore PCs," PARA 2006, LNCS, vol. 4699, 2007, pp. 956–965.

[10] S. Donath, K. Iglberger, G. Wellefin, T. Zeiser, A. Nitsure, and U. Rüde, "Performance comparison of different parallel lattice Boltzmann implementations on multi-core multi-socket systems," Int. J. Comp. Sci. Eng., vol. 4:1, 2008, pp. 3–11.

[11] Z. L. Guo, B. C. Shi, and N. C. Wang, "Lattice BGK model for incompressible Navier-Stokes equation," J. Comput. Phys., vol. 165, 2000, pp. 288–306.

[12] D. Kandhai, A. Koponen, and A. G. Hoekstra et al., "Lattice-Boltzmann hydrodynamics on parallel systems," Comput. Phys. Commun, vol. 111, 1998, pp. 14–26.

[13] B. Cgapman, "Managing Multcore with OpenMP," EuroPVM/MPI 2008, LNCS, vol. 5205, 2008, pp. 3–4.

[14] A. Marowka. "Performance of OpenMP Benchmarks on Multicore Processors," ICA3PP 2008, LNCS, vol. 5022, 2008, pp. 208–219.

[15] M. C. Maury, X. Ding, C. D. Antonopoulos, and D. S. Nikolopoulos, "An evaluation of OpenMP on Current and Emerging Multithreaded/Multicore Processors," IWOMP 2005/2006, LNCS, vol. 4315, 2008, pp. 133–144.

[16] V. Sarkar, "Programming Challenges for Petascale and Multicore Parallel Systems," HPCC 2007, LNCS, vol. 4782, 2007, pp. 1.

[17] C. Terboven, D. Mey, and S. Sarholz, "OpenMP on Multicore Architectures," IWOMP 2007, LNCS, vol. 4935, 2008, pp. 54–64.

[18] J. L. Shon, , "Evaluation of FIDAP on some classical laminar and turbulent benchmarks," J. Numerical Methods in Fluids, vol. 8, 1988, pp. 1469–1490.

[19] U. Ghia, K. N. Ghia, and C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," J. Comput. Phys., vol. 48, 1982, pp. 387–413.