Computer Physics Communications 211 (2017) 16–22

Contents lists available at ScienceDirect

Computer Physics Communications

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

The novel implicit LU-SGS parallel iterative method based on the diffusion equation of a nuclear reactor on a GPU cluster

Jilin Zhang [a,b,c,d,f], Chaoqun Sha [e], Yusen Wu [a,b,*], Jian Wan [a,b,d,f], Li Zhou [a,b], Yongjian Ren [a,b], Huayou Si [a,b], Yuyu Yin [a,b,c,d,f], Ya Jing [g]

[a] School of Computer Science and Technology, Hangzhou Dianzi University, 310018, Hangzhou, China
[b] Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou, China
[c] College of Electrical Engineering, Zhejiang University, 310058, Hangzhou, China
[d] School of Information and Electronic engineering, Zhejiang University of Science & Technology, 310023, Hangzhou, China
[e] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
[f] Zhejiang Provincial Engineering Center on Media Data Cloud Processing and Analysis, Hangzhou, Zhejiang, China
[g] China Internet Network Information Center, Beijing, 100190, China

ARTICLE INFO

Article history:
Received 20 February 2016
Received in revised form
25 May 2016
Accepted 2 July 2016
Available online 12 July 2016

Keywords:
Reactor
GPU parallel
CUDA
LU-SGS

ABSTRACT

GPU not only is used in the field of graphic technology but also has been widely used in areas needing a large number of numerical calculations. In the energy industry, because of low carbon, high energy density, high duration and other characteristics, the development of nuclear energy cannot easily be replaced by other energy sources. Management of core fuel is one of the major areas of concern in a nuclear power plant, and it is directly related to the economic benefits and cost of nuclear power. The large-scale reactor core expansion equation is large and complicated, so the calculation of the diffusion equation is crucial in the core fuel management process. In this paper, we use CUDA programming technology on a GPU cluster to run the LU-SGS parallel iterative calculation against the background of the diffusion equation of the reactor. We divide one-dimensional and two-dimensional mesh into a plurality of domains, with each domain evenly distributed on the GPU blocks. A parallel collision scheme is put forward that defines the virtual boundary of the grid exchange information and data transmission by non-stop collision. Compared with the serial program, the experiment shows that GPU greatly improves the efficiency of program execution and verifies that GPU is playing a much more important role in the field of numerical calculations.

1. Introduction

In recent years, compute unified device architecture (CUDA) has been widely used in a number of important areas, including medical imaging, computational fluid dynamics, environmental science and material science. In the energy industry because nuclear energy has a low carbon content, high energy density, high sustainability and other characteristics, nuclear energy development trends in environmental science and materials science cannot be replaced by others. The thorium-based molten salt reactor nuclear power system (TMSR) [1,2], one of the first strategic pilot projects in the Chinese Academy of Sciences, provides an efficient data storage service and a high performance computing service for nuclear reactors. Chinese Academy of Sciences Liren Shen and Feifei Wang [3] built the TMSR system on a GPU cluster, and using the Linpack test, verified that the TMSR platform has strong floating point computing power. They ran a three-dimensional neutron reactor diffusion equation in the experimental environment that has a GPU cluster that, compared with a test on the CPU, decreased the computing time by nearly 4.5 times.

The Lower-Upper Symmetric Gauss–Seidel (LU-SGS) [4,5] iterator method discussed in this paper is widely utilized due to its advantages of mesh topology and is the most popular method used in air dynamics. Complicated physical phenomena such as shock boundary layer interference and high temperature flow are major concerns of hypersonic vehicles. To solve these problems, we need to combine computational fluid dynamics,

* Correspondence to: School of Computer Science and Technology, Hangzhou Dianzi University, No. 1, 2nd Street, Jianggan District, Hangzhou, China.
E-mail address: wuyusen8988@126.com (Y. Wu).

http://dx.doi.org/10.1016/j.cpc.2016.07.014
0010-4655/

computational structural dynamics, computational heat transfer and other fields [6]. Focusing on these issues, some scholars use the Mentor'SST equation [7] turbulence model, Roe format, AUSM+-up format [8] and LU-SGS method to compile the hypersonic CFD calculation program. Based on GPU architecture and CUDA programming to solve the data parallel implicit CFD [9] greatly improved the calculation speed of the implicit solution.

The LU-SGS iterative method has been widely used in many fields. The Science and Technology department of Sheriff University proposed a new parallel Seidel Gauss method [10] for the solution of partial differential equations using the finite difference method. Experimental results of the iterator method show that the iteration number of convergence criteria for any number of sub-domains is not much different than that of the original Seidel Gauss method. Combined with the theory of reactor diffusion equation parallel research and based on the idea of collision exchange, we propose an implicit LU-SGS iterative parallelization algorithm, carrying out multi-dimensional iterative experiments. By analyzing the results of the LU-SGS iterator method on a GPU cluster, research is conducted on the overhead and benefits of using GPUs that include data transmission overhead, CUDA context initialization overhead and load core computing cost; research on the effects of complex memory hierarchy and thread level structure on the efficiency of running programs.

The techniques discussed in this paper are all based on the rapid development of GPU architecture. GPU technology from the era of fixed-function graphics pipelines to the separate rendering architecture era, and then to unified graphics and computing processors has been rapid. In particular, the era of unified rendering is a milestone in the development of GPU. It replaces the traditional vertex and pixel rendering pipeline separation structure with unified rendering hardware and provides the ability to calculate the geometry. Guangping Tang and his colleagues from the school of Computer Science in New York University also put forward parallel running of the tridiagonal matrix equation in a multi-core GPU architecture system [11]. Tang proposes an optimized parallel algorithm that combines the cyclic reduction method and partition method to improve the efficiency of the three diagonal linear equations. Experimental results show that the cyclic reduction method and the partition method are improved by 13.2% and 19.2%, respectively.

The computer graphics processor (GPU) integrated geometric transformation, lighting, cutting and rendering engine and other functions, with at least 10 million polygon processing power [12] greatly improves the speed of computer graphics processing, enhances the quality of the graphics, and promotes the development of the field of computer graphics applications. GPU is not only used in the research of graphic images, but it has been widely used in the field of numerical computation because of its strong floating point computing power. More and more scientists will process large numerical calculations on the GPU that substantially reduce execution time and improve the efficiency of the program.

The rest of the paper is organized as follows. In Section 2, two derivation methods of the LU-SGS iterative method are discussed, the LU splitting method and the LU approximate decomposition method, and the equivalence of the two methods is verified. Subsequently, the steps of the LU-SGS solution for a two-dimensional grid are introduced. In Section 3, the combination of the LU-SGS method and CUDA programming is proposed, and the implementation process of the CUDA kernel function is analyzed. In Section 4, actual experimental results are obtained, and the impact of the LU-SGS method implementation cost and other issues are discussed. Conclusions and future work are shown in Section 5.

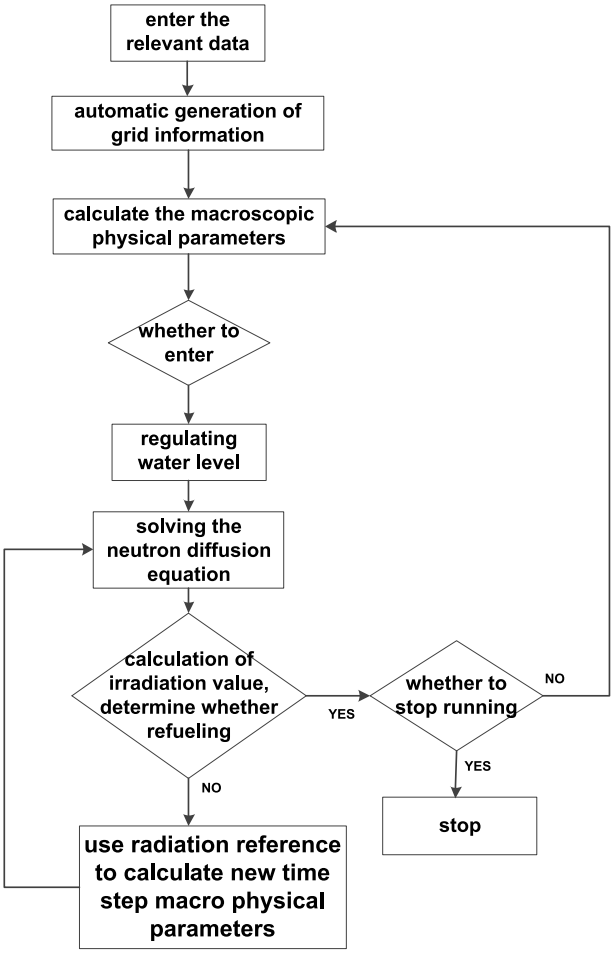

Fig. 1. Reactor fuel management program.

2. Reactor diffusion theory and derivation of LU-SGS iterative method

2.1. Reactor diffusion equation

Reactor core fuel management [13] is one of the problems that concerns nuclear power stations. Optimizing fuel management can lower nuclear energy cost, improve the economic efficiency of nuclear power and provide a security guarantee for operation of the reactor. Engineers can obtain the best management plan by simulation analysis of several processes: initial loading, refueling, reaching the equilibrium state. The management process is shown in Fig. 1. Solution of the diffusion equation of the reactor occupies an important position in the calculation of the overall design of industrial reactors. Research on finite methods for solving the diffusion equation of reactors has been conducted for years, including the Mesh method [14], Nodal method [15], Green function [16], finite difference method [17], the symmetric sip methods [18] and so on. Solving this type of problem with the finite difference method is easy, but it is difficult calculate two-dimensional or three-dimensional neutron diffusion equations.

The reactor diffusion equation [19] is as follows:

$$-\nabla \cdot D_\mu \nabla \varphi(r) + \sum_{t\cdot\mu} \varphi_\mu(r) - \sum_{\mu'=1}^{G} \sum_{\mu'\to\mu} \varphi_{\mu'}(r) = \frac{x_\mu}{k}\theta(r)$$

$$\theta(r) = \sum_{\mu'=1}^{G} \left(\nu \sum_{f}\right)_{\mu'} \varphi_{\mu'}(r) \quad \mu = 1, 2, 3 \ldots G.$$

Boundary condition:

$$-D_j \nabla \phi_j \cdot n = A_j \phi_j \qquad A_j = \begin{cases} \sqrt{\sum_{t,j} D_j} \\ 0 \end{cases}$$

$\phi_\mu(r)$ is the neutron fluence rate in group μ; $\sum_{\mu' \to \mu}$ is the scattering cross section from group μ' to group μ; ν is the number of neutrons; D_μ is the diffusion coefficient of group μ; $\sum_{t\cdot\mu}$ is the macro total cross section of group μ; $\sum_{\mu'=1}^{G}$ is the macroscopic fission cross section of group μ'.

To be convenient for computing, turn the differential equation into the different equation:

$$A\phi = S$$

A is an n-order and seven-angle matrix. For the large core, n is generally huge, so we commonly use the Seidel iterative method and the Successive Over Relaxation iteration method to speed up the process of running the software. We consider using the Collision iterative principle in the LU-SGS iterative method, utilizing residue value ω and using an extrapolation method to accelerate equation convergence. The smaller the residual value is, the longer is the execution time and the higher is the accuracy.

2.2. Derivation of LU-SGS iterative method

There are two methods for the derivation of the LU-SGS iterator method: the LU splitting method and the LU approximate factorization method. The LU splitting method [20] does not generate error in the process of splitting the matrix, but it generates error in the process of applying the solution equation. The LU approximate factorization method is the opposite. D stands for the diagonal matrix, L stands for the lower-diagonal matrix, and U stands for the upper-diagonal matrix.

(1) LU splitting method

Forward scanning formula:

$$\Delta R^* = D^{-1}(RHS - L\Delta R^*) \tag{1}$$

$$(D + L)\Delta R^* = RHS. \tag{2}$$

Backward scanning formula:

$$(D + U)\Delta R^* = RHS - L\Delta R^*. \tag{3}$$

Change the formula on either side of the equal signed (3):

$$D\Delta R^* = RHS - L\Delta R^* - U\Delta R^* \tag{4}$$

$$\begin{aligned} \Delta R^* &= D^{-1}(RHS - L\Delta R^* - U\Delta R^*) \\ &= D^{-1}(D\Delta R^* - U\Delta R^*) \\ &= \Delta R^* - D^{-1}U\Delta R^*. \end{aligned} \tag{5}$$

(2) LU approximate factorization method

Approximate decompose method for the left side of the linear equations:

$$\begin{aligned} (D + L + U) &= D(I + D^{-1}L + D^{-1}U) \\ &\approx D(I + D^{-1}L)(I + D^{-1}U) \\ &= D^{-1}(D + L)(D + U). \end{aligned} \tag{6}$$

The equations are transformed into:

$$D^{-1}(D + L)(D + U)\Delta R^* = RHS \tag{7}$$

Forward scanning, also known as downward scanning. We assume that $\Delta R^* = D^{-1}(D + U)\Delta R^*$, so Eq. (7) transforms into

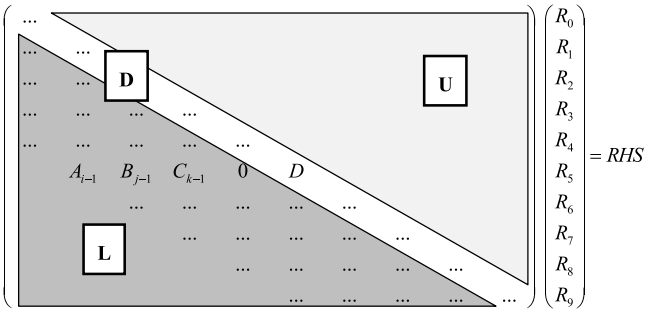

Fig. 2. Implicit two-dimensional LU-SGS matrix equations.

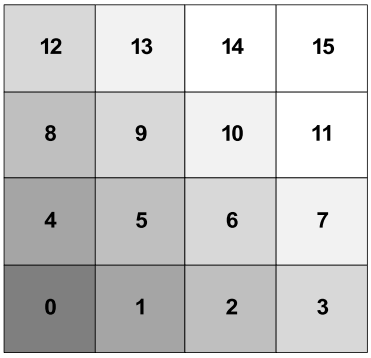

Fig. 3. 4 × 4 two-dimensional structure grid.

$(D + L)\Delta R^* = RHS$. In the calculation of the actual flow field, each cell can be expressed by the equations above. By combining all the grid and linear equations, we get linear equations of flow field in the current grid, as shown in Fig. 2.

The following formula can be obtained from Fig. 2:

$$D\Delta R^* + A_{i-1}\Delta^*_{i-1} + B_{j-1}\Delta R^*_{j-1} + C_{k-1}\Delta R^*_{k-1} = RHS \tag{8}$$

$$\Delta R^* = D^{-1}(RHS - A_{i-1}\Delta R^*_{i-1} - B_{j-1}\Delta R^*_{j-1} - C_{k-1}\Delta R^*_{k-1}) \tag{9}$$

$$= D^{-1}(RHS - L\Delta R^*). \tag{10}$$

Formula (11) given in Box I is the same as LU splitting method formula (2), so it is proved that the two splitting methods are equivalent to forward scanning. In the same way, backward scanning is proved equivalent.

2.3. LU-SGS 4 × 4 two-dimensional grid

We use a two-dimensional structure grid as an example to elaborate the iteration of the LU-SGS method in more detail; each block of mesh starts from 0, as shown in Fig. 3. Combine each grid cell with matrix linear equations, and then linear equations (11) can be obtained.

Taking D as the dividing point, we divide the linear equations into upper triangle and lower triangle and scan upward and downward, respectively. As a result of upward scanning, in Fig. 3, we find that the interface of the 0th grid is the 1st and the 4th sub-grid, corresponding to $D_0R_0 + U_1R_1 + U_4R_4 = RHS$. Downward scanning is similar. Solving in turns, we can obtain data information for all sub-grids of a 4 × 4 two-dimensional structured grid.

3. Parallel LU-SGS iterator method

CUDA is a new type of GPGPU architecture [21] introduced by NVIDIA Corporation. It is a program for writing in display

$$\begin{pmatrix} D_0 & U_1 & & & U_4 & & & & & & & & & & & \\ L_0 & D_1 & U_2 & & & U_5 & & & & & & & & & & \\ & L_1 & D_2 & U_3 & & & U_6 & & & & & & & & & \\ & & L_2 & D_3 & U_4 & & & U_7 & & & & & & & & \\ L_0 & & & L_3 & D_4 & U_5 & & & U_8 & & & & & & & \\ & L_1 & & & L_4 & D_5 & U_6 & & & U_9 & & & & & & \\ & & L_2 & & & L_5 & D_6 & U_7 & & & U_{10} & & & & & \\ & & & L_3 & & & L_6 & D_7 & U_8 & & & U_{11} & & & & \\ & & & & L_4 & & & L_7 & D_8 & U_9 & & & U_{12} & & & \\ & & & & & L_5 & & & L_8 & D_9 & U_{10} & & & U_{13} & & \\ & & & & & & L_6 & & & L_9 & D_{10} & U_{11} & & & U_{14} & \\ & & & & & & & L_7 & & & L_{10} & D_{11} & U_{12} & & & U_{15} \\ & & & 0 & & & & & L_8 & & & L_{11} & D_{12} & U_{13} & & \\ & & & & & & & & & L_9 & & & L_{12} & D_{13} & U_{14} & \\ & & & & & & & & & & L_{10} & & & L_{13} & D_{14} & U_{15} \\ & & & & & & & & & & & L_{11} & & & L_{14} & D_{15} \end{pmatrix} \begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \\ R_7 \\ R_8 \\ R_9 \\ R_{10} \\ R_{11} \\ R_{12} \\ R_{13} \\ R_{14} \\ R_{15} \end{pmatrix} \tag{11}$$

Box I.

equipment for execution based on the C language. Users do not need to learn complex vertex shader language, display chip instructions or special hardware structure.

CUDA architecture can be divided into two parts: the HOST and the DEVICE [22]. The HOST part refers to code executed on the CPU, and the DEVICE part refers to the program executed on the display chip, which called a kernel function. The smallest unit of execution of GPU in CUDA is a thread, and multiple threads can form a block; threads in a block can access the same shared memory, as shown in Fig. 7, where each SM shares the same piece of 64 × 1K shared memory. The number of threads within each block is limited; multiple blocks can form a grid, but because different blocks are unable to access the same piece of shared memory, it is difficult to communicate between different blocks. Each thread has its own register and local memory; in addition, all threads can share the same global memory, constant memory and texture memory.

Take a one-dimensional LU-SGS iterative method as an example to simulate the execution process of the parallel algorithm. We use ● to denote a given number, ◍ to denote the new value of virtual boundary data after collision, ○ to denote the one-dimensional sub-grid and K to denote the number of iterations. One LU-SGS iteration consists of a right iteration and a left iteration, and the experimental domain will be divided into n sub-domains, as shown in Fig. 4.

For two-dimensional problems, we use two-dimensional blocks to greatly reduce the difficulty of writing code. We use two dimensional texture memory to simulate heat conduction. Compared to the global and constant memory, texture memory has the following advantages: cache containing, not bound by access patterns, addressing calculation of delay hidden better, can be broadcast to more than one independent variable in a single operation, etc. We test the two-dimensional LU-SGS program in texture memory, and then, we will compare performance of multiple groups of global memory and texture memory experimental results.

Each block processes one parallel task. A K40 graphics card for each multi-stream processor (SM) [23] has 192 stream processors (SP). However, blocks cannot communicate like the interprocess communication of MPI [24,25]. Therefore, we propose a new iterative scheme, as shown in Fig. 5, and we introduce synchronization primitives. We use ◍ to denote the iterated sub-grid and ⊖ to denote the virtual boundary. The number of iterations is manually set for the experiment. This iterative process is proposed because blocks cannot communicate and because it can reduce synchronization overhead.

Fig. 6 represents the schematic diagram of upward and downward scanning, respectively. Assume that data transfer from the black shadow part to the white part represents one communication overhead, and the black shadow section is needed for communication data. When using texture memory, we can directly access the texture through x, y. In a kernel function, we need to map the threadIdx and blockIdx to the location of the pixel, and we obtain the data of the adjacent grid blocks using the tex2D function.

Table 1
Results of one-dimensional LU-SGS method.

No.	Array size	Parallel GPU time	Serial CPU time	Speedup
1	2048	26.743	175.084	6.547
2	4096	146.396	1379.64	9.424
3	8192	429.734	11122.98	22.574
4	10240	2255.90	92891.23	41.177

4. Numerical experiment and analysis

4.1. Experimental equipment architecture

NVIDIA server architecture is based on a hybrid computing model, where CPU and GPU work together. The architecture is applied in a wide range of scientific computing applications. The current Kepler architecture has once again raised the HPC industry standards. GK110 GPU Kepler [26,27] transistor is composed of 7.1 billion. Currently, the GK110 has been applied in the Tesla K20 and K40 GPU computing accelerator. The experiment uses 4 T K40 GPU servers, and its core architecture is shown in Fig. 7.

4.2. The experimental results and analysis

Table 1 shows the result of a one-dimensional LU-SGS iterative method on CPU and GPU. Table 2 shows the results of a two-dimensional LU-SGS grid iterative method. The one-dimensional residual value is set to 1e–8. Due to the large size of the two-dimensional grid, in order to shorten execution time and improve execution efficiency, the residual value is set to 1e–3.

Using the definition of speedup ratio (the single core serial time vs. the multi-core parallel time), each test is conducted in the corresponding K40 Sever and 16 core serial CPU. With continuous expansion of the number of arrays and grids, CPU serial time increases at a speed of several tens of times. The parallel time of the incremental speed slightly is slow, the incremental rate is not large, and it has been controlled at 5–8 times. Two-dimensional tests obtained the highest speedup due to the large amount of

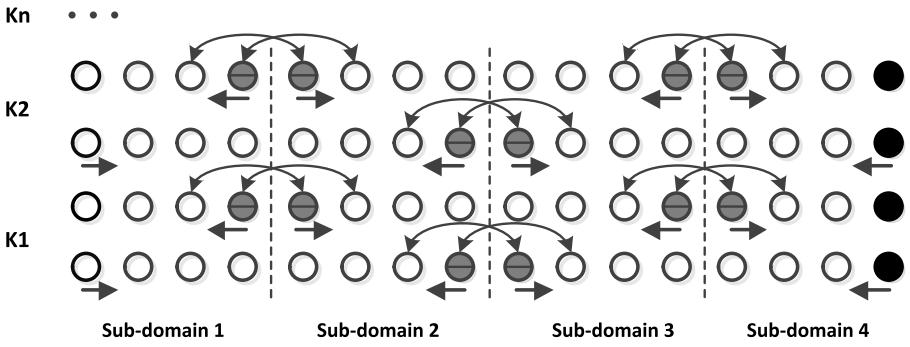

Fig. 4. Schematic diagram of one-dimensional LU-SGS collision iteration.

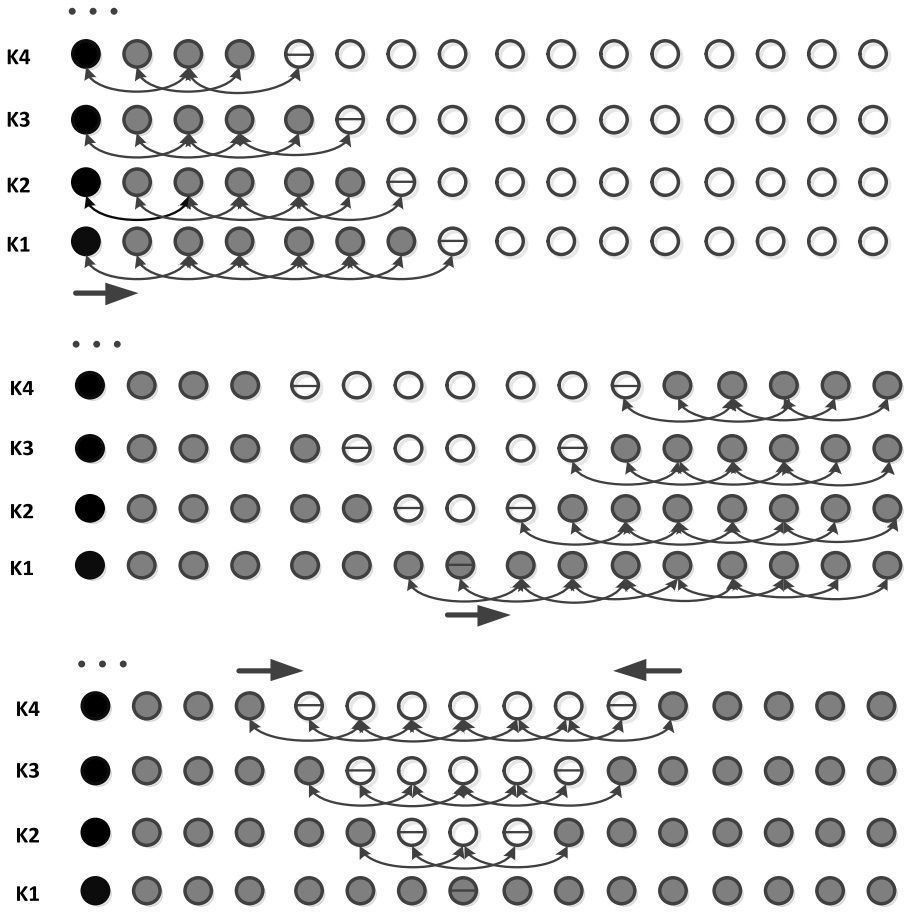

Fig. 5. A new parallel iterative scheme on GPU.

Table 2
Results of the global memory.

No.	Array size	Parallel GPU time	Serial CPU time	Speedup
1	256 × 256	24.679	146.764	5.947
2	512 × 512	99.011	1808.523	18.266
3	1024 × 1024	155.452	4621.609	29.730
4	2048 × 2048	10032.442	10032.442	27.943

computation, and they can effectively use the computing power of GPU. Fig. 8 lists the different data size of the texture memory and global memory.

After starting GPU parallel, due to the high parallelism degree between the GPUs and increasing transmission overhead among the cards, when 2-4 GPU cards are opened, the execution rate decreases by 15%∼30%. In addition, by using the method of zero copy memory to avoid unnecessary data replication, the performance can be enhanced by nearly 20%. Combined with texture memory to reduce the memory scheduling overhead, the acceleration ratio can be improved by 1–1.5 times, as shown in Table 3.

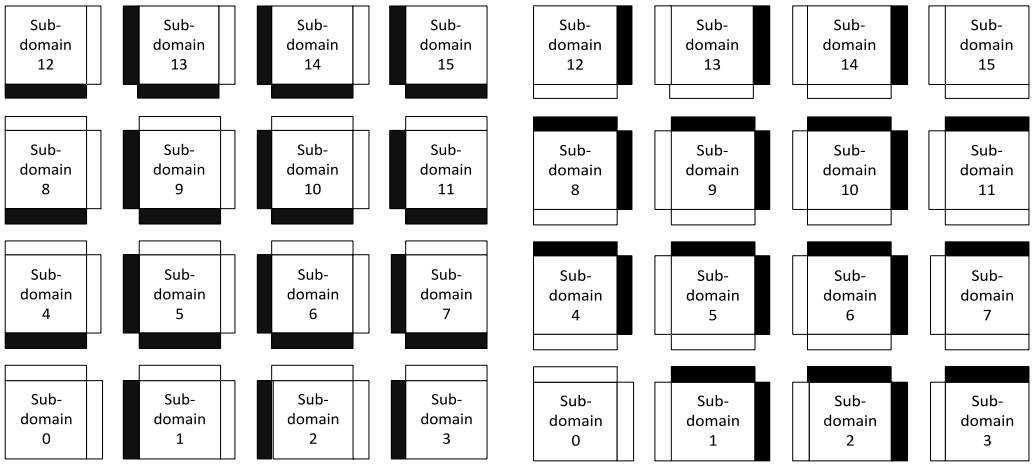

Fig. 6. Upward and downward scanning.

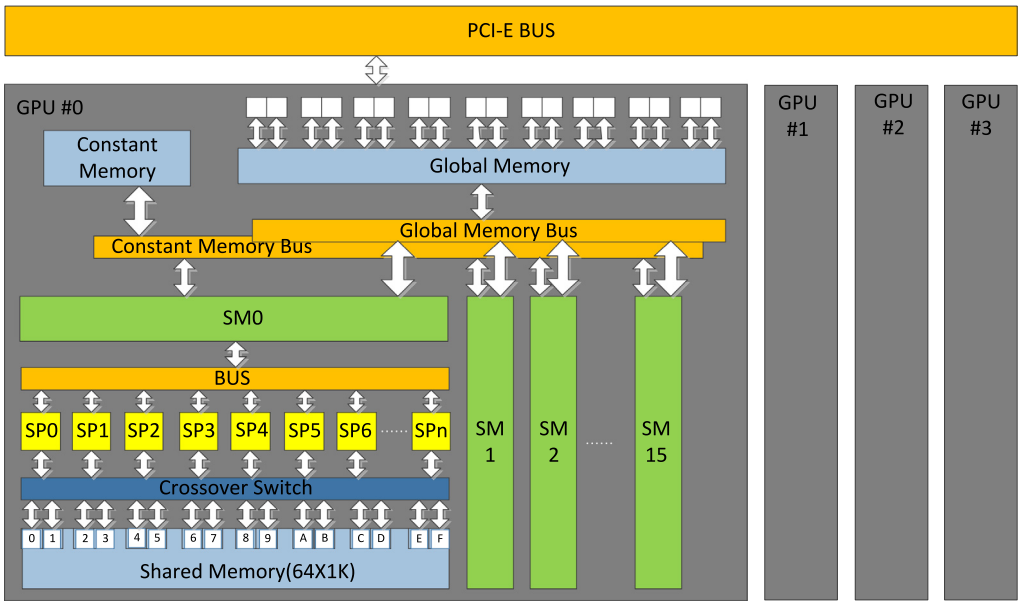

Fig. 7. The composition of the 4 GPUs K40 server module diagram.

Table 3
Results of the texture memory.

No.	Array size	Parallel GPU time	Serial CPU time	Speedup
1	256 × 256	13.406	109.346	8.156
2	512 × 512	69.501	1561.07	22.461
3	1024 × 1024	102.079	3892.101	38.128
4	2048 × 2048	177.650	8023.912	45.167

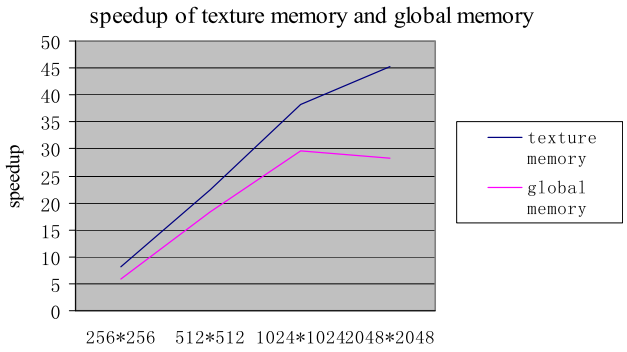

Fig. 8. The performance speedup of texture memory and global memory.

According to Fig. 8, we find that the texture memory acceleration ratio shows an upward trend; a clear turning point is observed in the third grid, and global memory is more obvious. Speedup has decreased by 1.787 in the fourth grid. Analysis of the results show the following two reasons:

(1) The GPU server temperature is too high, reducing frequency automatically.

(2) The task scheduling overhead between memory blocks is increased due to external reasons, such as the execution of parallel programs that require a large number of threads to be scheduled.

In general, factors affecting efficiency of the implementation of the program are as follows:

(1) Reasonable distribution of the number of threads can reduce the unnecessary thread scheduling.

(2) Familiarity with GPU architecture, using efficient and reasonable parallel algorithms can reduce the running time substantially.

(3) Different memory's read and write operation clock cycle variation. Shared memory requires 10–20 clock cycles, but global memory requires more than 200. Shared memory is limited, and reasonable use can play a major role.

5. Conclusion and future work

By comparison, we can sum up the advantages of GPU for floating point computing and reasonable scheduling of memory. With a high-performance price ratio, the power of GPU floating point computing is approximately 10 times that of CPU, the bandwidth is 5 times that of CPU, but the cost of GPU is only 3–4 times that of CPU. In addition, GPU has good portability, ordinary desktop or notebook computers can support general floating point computing. Due to the characteristics of graphics processing and general computing, the results can be directly displayed by visual devices.

One- or two-dimensional LU-SGS iteration in different memory indicates that the power of GPU for complex floating point computing is very strong. By analyzing the correlation between texture memory and the global memory execution rate and setting reasonable a kernel function, we reduce the scheduling overhead between blocks. GPU will play an increasingly large role in the field of mathematical computing for material and energy that require a large number of complex calculations.

Future research may focus on the optimization of large-scale applications, algorithm and system structure, and speeding up the pace of GPU in the development of application software.

Acknowledgments

This work is partly supported by the National Natural Science Foundation of China under Grants No. 61672200, No. 61572163, No. 61472112, No. 61572163, No. 61202094 and No. 61472109; National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant No. 2014BAK14B04; the Zhejiang Natural Science Funds under Grants No. LY16F020018 and No. LY13F020047; the National High Technology Research and Development Program of China under Grant No. 2015AA01A303; the Key Laboratory of Complex Systems Modeling and Simulation program of the Ministry of Education and the Chinese Postdoctoral Science Foundation No. 2013M541780 and No. 2013M540492.

References

[1] Mianheng Jiang, Hongjie Xu, Zhimin Dai, Bull. Chinese Acad. Sci. (2012).
[2] X. Zhang, X. Wang, W. Gong, Y. Fu, ASME 2015 Pressure Vessels and Piping Conference, American Society of Mechanical Engineers, 2015.
[3] F. Wang, H. Wang, H. Yu, He Jishu/Nucl. Tech. 38 (4) (2015).
[4] S. Yoon, A. Jameson, AIAA J. 26 (9) (1988) 1025–1026.
[5] ShiGang Li, ChangJun Hu, JunChao Zhang, YunQuan Zhang, Sci. China Inf. Sci. 58 (9) (2015) 1–14. (SCI, Impact factor: 0.85).
[6] W. Cao, C.F. Xu, Z.H. Wang, L. Yao, H.Y. Liu, Cluster Comput. 17 (2) (2013) 255–270.
[7] AIAA, AIAA J. (2006).
[8] D. Knight, J. Longo, D. Drikakis, D. Gaitonde, A. Lani, I. Nompelis, et al., Prog. Aerosp. Sci. 48–49 (2) (2012) 8–26.
[9] R. Tavakoli, P. Davami, Appl. Math. Comput. 188 (1) (2007) 713–719.
[10] M.S. Liou, J. Comput. Phys. 214 (1) (2006) 137–170.
[11] Y. Komura, Y. Okabe, Meeting abstracts of the Physical Society of Japan. Vol.66, The Physical Society of Japan (JPS), 2011, pp. 1155–1161.
[12] d.R.E. Gutiérrez, F.J. Jiménez-Hornero, A.B. Ariza-Villaverde, J.M. Gómez-López, Comput. Geosci. 64 (1) (2013) 1–6.
[13] P. Silvennoiem, M.M.H. Ragheb, T.J. Higgins, IEEE Trans. Nuclear Sci. 25 (2) (1978) 1034–1036.
[14] M. Sterk, B. Robi, R. Trobec, Parallel Numer. (2005).
[15] D. Ginestar, G. Verdú, R. Miró, D. Hennig, Ann. Nucl. Energy 29 (29) (2002) 1171–1194.
[16] A. Trkov, M. Najžer, L. Škerget, J. Nucl. Sci. Technol. 27 (6) (1990) 766–777.
[17] Yoshitaka Naito, Shin-ichiro Tsuruta, Masatoshi Hayashi, J. Nucl. Sci. Technol. 18 (6) (1981) 571–580.
[18] H.H. Wang, Nucl. Sci. Eng. (United States) 67 (2) (1978).
[19] E.B. Davies, Rep. Math. Phys. 11 (2) (1977) 169–188.
[20] J.S. Shuen, J. Comput. Phys. 99 (2) (1992) 233–250.
[21] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, Y. Shi, J. Supercomput. 64 (2) (2013) 942–967.
[22] Z. Weng, P.E. Strazdins, 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, IPDPSW, IEEE Computer Society, 2014, pp. 1275–1284.
[23] R.A. Hankins, G.N. Chinya, J.D. Collins, P.H. Wang, R. Rakvic, H. Wang, et al., ACM SIGARCH Comput. Archit. News 34 (2) (2006) 114–127.
[24] Shigang Li, Torsten Hoefler, Snir Marc, Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing, HPDC'13, 2013, pp. 85–96. (acceptance rate: 15%, 20/131, Best Paper Nomination, 3/20).
[25] Shigang Li, Torsten Hoefler, Chungjin Hu, Snir Marc, Cluster Comput. (2014) 1–17. (SCI, Impact factor: 0.949).
[26] R. Shao, D. Linton, Spence Ivor, N. Zheng, Seventh International Conference on Graphic and Image Processing, International Society for Optics and Photonics, 2015.
[27] Shigang Li, Jingyuan Hu, Xin Cheng, Chongchong Zhao, Procedings of the 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2013, pp. 198–202.