Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Physics 221 (2007) 799–804

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

GPU accelerated molecular dynamics simulation of thermal conductivities

Juekuan Yang [a,*], Yujuan Wang [a,1], Yunfei Chen [a,b,2]

[a] *Department of Mechanical Engineering, Southeast University, Nanjing 210096, PR China*
[b] *China Education Council Key Laboratory of MEMS, Southeast University, Nanjing 210096, PR China*

Received 15 February 2006; received in revised form 26 June 2006; accepted 28 June 2006
Available online 14 August 2006

Abstract

Molecular dynamics (MD) simulations have become a powerful tool for elucidating complex physical phenomena. However, MD method is very time-consuming. This paper presents a method to accelerate computation of MD simulation. The acceleration is achieved by take advantage of modern graphics processing units (GPU). As an example, the thermal conductivities of solid argon were calculated with the GPU-based MD algorithm. The test results indicated that the GPU-based implementation is faster than that of CPU-based one. The speedup of a factor between 10 and 11 is realized.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Graphics processing unit; Molecular dynamics; Thermal conductivity

1. Introduction

Molecular dynamics (MD) is a widely-used computational tool for simulating the properties of liquids and solids at atomistic level in research areas such as chemistry, thermodynamics and several other areas [1]. However, MD simulation is based on an intensive computation process. So far, the time scale and the system size that can be probed using MD are still limited due to the computational power of current computers. For the study of problems such as nanofluids and stiction in MEMS devices, millions even billions of atoms are required. As a result, the computation time will be too long to bear.

Over the past few years, modern graphics processing units (GPUs) for commodity PC hardware has evolved into a powerful programmable parallel streaming processor. For example, the G70, Nvidia's most recent GPU, performs up to 165 billion floating-point multiplies per second (165 Gflops) [2]. However, a 3 GHz Pentium 4 CPU can only theoretically issue around 6 Gflops [3]. The enormous computational potential of GPUs

[*] Corresponding author. Tel.: +86 2583800020.
E-mail addresses: yangjk@seu.edu.cn (J. Yang), yujuanwang@seu.edu.cn (Y. Wang), yunfeichen@seu.edu.cn (Y. Chen).
[1] Tel.: +86 2583792603.
[2] Tel.: +86 2583792178.

0021-9991/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2006.06.039

has led to an explosion of research in ways to leverage GPUs for general-purpose computation on GPUs (GPGPU) [4,5]. As pointed out by Macedonia [6] the GPU has entered computing's mainstream.

At present, Monte Carlo and lattice-Boltzmann model (LBM) have been implemented on GPU. Tomov et al. [7] reported their implementation of Monte Carlo type simulation on GPU. Their experimental results have shown that the GPU-based simulation is about three times faster than the CPU version. Li et al. [8,9] used GPU to accelerate the computation of the LBM, and applied it to a variety of fluid flow problems. They got a speedup of a factor 8–15 [9]. Buck et al. [10] are planning to use the massive compute power available in today's GPU to accelerate all the key components of Gromacs which is one of the leading software packages used to simulate protein folding.

In this paper, we present an implementation of MD simulation on GPU. As an example, the MD algorithm will be used to calculate the thermal conductivities of solid argon. Our goal is to reduce the total computational time of MD simulation at a very high performance/cost ratio with the introduction of the GPU algorithm.

The paper is organized as follows. In Section 2 the MD algorithm for thermal conductivity calculation is described. The details of the implementation of MD simulation on GPU are presented in Section 3. In Section 4 the performance results for our implementation are given and compared with the CPU version. Finally, our conclusions are summarized in Section 5.

2. MD simulation

In MD simulation, each of the atoms or molecules is treated as a point mass. Given the interaction potential between atoms, the force acting on each atom can be calculated. Based on Newton's second law, the motion of a large number of atoms can be described. From the motion of the ensemble of atoms, a variety of useful microscopic and macroscopic information can be extracted such as transport coefficients, structural properties, etc.

In this paper, the Lennard-Jones (LJ) potential is used to describe the energy of interaction between two argon atoms [11]

$$\phi(r_{ij}) = 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \quad r_{ij} \leqslant r_{\mathrm{c}}, \tag{1}$$

where r_{ij} denotes the distance between the two atoms, ε is depth of potential well, σ is effectively the diameter of one of the atoms, for argon $\varepsilon = 1.67 \times 10^{-21}$ J, $\sigma = 0.34$ nm, and r_{c} is the cutoff radius. The interatomic force can be calculated by differentiating Eq. (1), and is given by

$$\boldsymbol{F}(r_{ij}) = \left(\frac{48\varepsilon}{\sigma^2}\right)\left[\left(\frac{\sigma}{r_{ij}}\right)^{14} - \frac{1}{2}\left(\frac{\sigma}{r_{ij}}\right)^{8}\right]\boldsymbol{r}_{ij}, \quad r_{ij} \leqslant r_{\mathrm{c}}. \tag{2}$$

The velocity Verlet algorithm is chosen to integrate the equations of atom motions. That is [1]

$$\begin{aligned} \boldsymbol{r}(t+\Delta t) &= \boldsymbol{r}(t) + \boldsymbol{v}(t)\Delta t + \frac{\boldsymbol{F}(t)}{2m}\Delta t^2, \\ \boldsymbol{v}(t+\Delta t) &= \boldsymbol{v}(t) + \frac{1}{2m}[\boldsymbol{F}(t) + \boldsymbol{F}(t+\Delta t)]\Delta t, \end{aligned} \tag{3}$$

where Δt is the time step of calculation, m is the mass of the atom, for argon $m = 6.63 \times 10^{-26}$ kg. In classical MD simulation, the temperature of the system is defined as the kinetic energy of atoms [11]

$$T = \frac{1}{3k_{\mathrm{B}}N}\sum_i m v_i^2 \tag{4}$$

in which, v_i is the velocity, N the number of atoms simulated and k_{B} Boltzmann's constant.

Three main techniques have been developed to predict the thermal conductivity of a dielectric material using MD simulations. These are the Green–Kubo (GK) approach (an equilibrium method), a direct application of the Fourier law of conduction (a steady state, non-equilibrium method), and unsteady methods [12].

Since the GK method can be implemented on GPU more easily than the others, it is used in the current investigation. In GK method, the thermal conductivity is given by [11]

$$k = \frac{1}{3k_B VT^2} \int_0^{+\infty} \langle \boldsymbol{J}(t) \cdot \boldsymbol{J}(0) \rangle \, \mathrm{d}t, \tag{5}$$

where V is the volume of the simulation cell and $\langle \boldsymbol{J}(t) \cdot \boldsymbol{J}(0) \rangle$ is the heat current autocorrelation function (HCACF). For a pair potential, such as the LJ potential, the heat current vector is given by

$$\boldsymbol{J} = \sum_i \boldsymbol{J}_i = \sum_i \left(E_i \boldsymbol{v}_i + \frac{1}{2} \sum_{j, j \neq i} \boldsymbol{r}_{ij} (\boldsymbol{F}_{ij} \cdot \boldsymbol{v}_i) \right), \tag{6}$$

where E_i is the total energy of atom i, and is given by

$$E_i = \frac{1}{2} m v_i^2 + \frac{1}{2} \sum_j \phi(r_{ij}). \tag{7}$$

In order to get the thermal conductivities of bulk material from a small number of atoms, periodic boundary conditions are applied to the simulation cell in all three axis directions.

3. Details of GPU implementation

Currently, GPUs contain two types of programmable processors: the vertex processors and the fragment processors. In this study, we implement the MD simulation on fragment processors. There are two main reasons for this [13]. First, there are more fragment processors than vertex processors on a typical programmable GPU. For example, the GeForce 7800 GTX graphics card used in this paper has 24 fragment processors, and only 8 vertex processors. Second, the output of the fragment processors goes directly into video memory, which can be fed straight back in as a new stream of texture data.

The two-dimensional floating-point textures are used to store the atom positions, velocities, forces, neighbor lists and heat flux. For example, in position texture, each atom position $\boldsymbol{r}_i = (x, y, z)$ is represented as a 32-bit floating point value in the red, green and blue color components of a pixel. These textures are attached to three framebuffer objects (FBO) respectively. FBO is a new OpenGL extension, which is designed to replace the pbuffer extension. It is both simpler to use and more efficient than pbuffer. Attaching to FBO allows a texture to be used for both rendering (write) and texturing (read), so avoid getting data back and forth between video memory and system memory frequently, which is a GPGPU performance bottleneck. As a texture cannot be written and read at the same time, we use a pair of position, velocities and forces textures to compute the new data from the previous values.

Neighbor list is a technique widely used in MD to improving the speed of calculation. Neighbor list is constructed with a radius $r_{\text{list}} = r_c + \Delta r$, and it will be refreshed when it is invalid. The reconstruction procedure of neighbor list is very time-consuming, and its time complexity is $O(N^2)$. For solid simulation, atoms oscillate at their equilibrium position. Thus a large enough r_{list}(=3.2 in dimensionless unit) is used in this paper in both CPU- and GPU-based simulation to avoid neighbor list reconstruction during the whole simulation process.

In neighbor list texture, the red and green color components are used to indicate the position coordinate of the neighbor atoms in the position texture. In order to run efficiently on GPU, the neighbor list of each atom should be of the same length. Thus the atoms with fewer neighbors are given dummy neighbors. And the blue component of the neighbor list texture is used to indicate whether the neighbor atom is a dummy one. The layout of neighbor list texture is similar to the connectivity texture used to implement GPU-based surgical simulators in [14].

In CPU-based MD algorithm, if atom j is a neighbor of atom i, the interatomic force $\boldsymbol{F}(r_{ij})$ will be calculated only when $r_{ij} \leqslant r_c$. In programming, this is implemented by an inner-loop branching. But in GPU fragment programs, conditional code will deteriorate the GPGPU performance [15]. In our GPU-based MD algorithm, to avoid branching inside inner loops, r_c is set equal to r_{list}, that is to say, all forces with atoms in neighbor list are computed.

According to Newton's third law, the force atom j acts on atom $i(\boldsymbol{F}_{ij})$ is equal to the force atom i acts on atom $j(\boldsymbol{F}_{ji})$, but opposite in direction, that is $\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$. The fact is commonly used in CPU implementation to minimize the computation by calculating the force between a pair of atom once. But on GPU, we choose not to take advantage of this fact, as it would require a second rendering pass and additional information to indicate the sign of the force [14].

As given by Eq. (5), the heat current vector $\boldsymbol{J}$ at different time and the temperature of the system T are needed to calculate the thermal conductivity. The render target of the flux program is the heat flux texture, where each atom's contribution to heat flux $\boldsymbol{J}_i$ is represented by the color component of a pixel. The image will be transferred to CPU every 10 time steps and the total heat flux $\boldsymbol{J}$ is evaluated on CPU by summing $\boldsymbol{J}_i$. The system temperature can be calculated according to Eq. (4). In our GPU implementation, the velocity texture will be transferred to CPU every 100 time steps to calculate the instantaneous temperature. The system temperature used in Eq. (5) is the average of these instant temperatures.

4. Experimental results

The following simulations were carried out on a PC with Intel Pentium 3.0 GHz, 1 G main memory. The graphics chip is GeForce 7800 GTX with 256M video memory. The fragment program was written in Cg [16].

In all the following simulations, a 10 fs time step was adopted. Each simulation involves 600,000 iterations (6 ns), the first 4 ns was used to attain the thermal equilibrium. And the heat flux and instant temperature were calculated in the later 2 ns.

After simulation, the HCACF in Eq. (5) was calculated using Fourier transform. But in GPU-based MD simulations, the HCACF cannot decay to zero as expected, shown in Fig. 1 (original). This is attributed to the floating-point error of GPU. Today's GPUs only support the 32-bit floating-point operations with seven significant decimal digits. But 64-bit double precision arithmetic is better for very large-scale computational science problems, e.g. MD simulation. Additionally, GPUs floating-point arithmetic is still not IEEE-compliant with deficiencies in rounding precision [17]. For example, the error of addition and division operation is $[-1.000, 0.000]$ and $[-1.199, 1.375]$, respectively for Nvidia graphics card. And these error bounds are unsymmetrical. As a result, the average momentum of the system during the whole simulation process is not zero as initialized, which cause the failure of the HCACF decaying to zero. Fortunately, as shown in Fig. 1, when t is bigger than the correlation length, the HCACF fluctuates about a constant value. We assume that this constant value is the effect of computation error on HCACF. To get the proper thermal conductivities, the mean of HCACF from $t = 10$ ps to $t = 50$ ps was calculated, and subtracted from HCACF, as shown in Fig. 1 (revised). Then the revised HCACF was substituted into Eq. (5) to calculate the thermal conductivity.

Fig. 2 shows the calculated thermal conductivities of solid argon at different temperatures. All the simulations were carried out on GPU using a 2048-atom system. Errors incurred in determining the absolute values

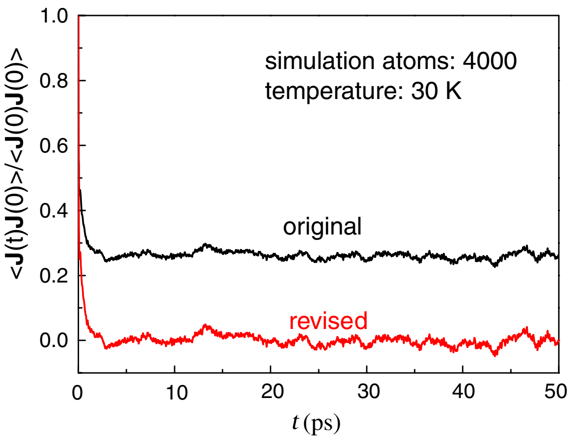

Fig. 1. Heat current autocorrelation function before and after adjustment.

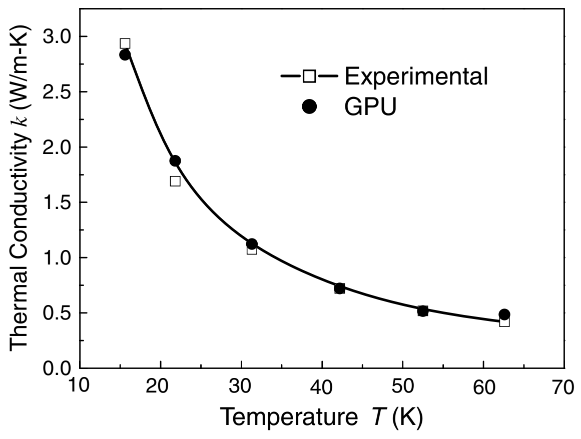

Fig. 2. Temperature dependence of thermal conductivity of solid argon calculated by GPU-based MD.

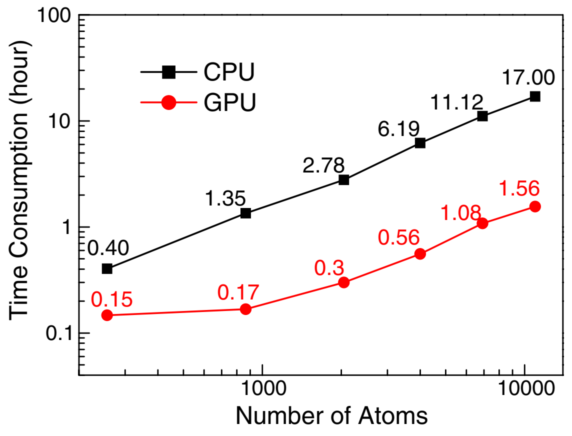

Fig. 3. Dependence of time consumption on system size.

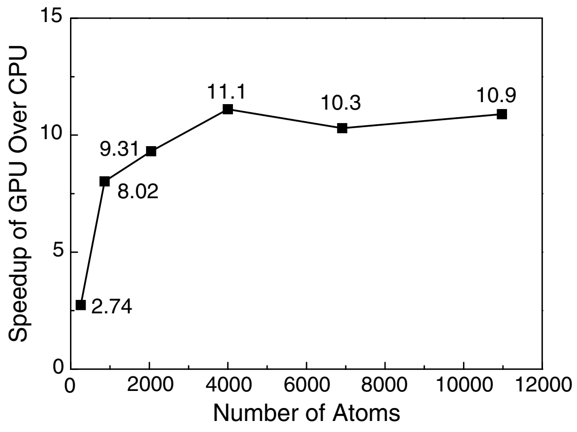

Fig. 4. Speedup of the MD simulation on GPU.

of the thermal conductivity are estimated to be about 10%. The results are consistent with the experimental data [18]. This justified the method that we used to revise HCACF and the feasibility of GPU-based MD simulation.

Fig. 3 presents the time in hours of the MD simulation as a function of the system size, running on both the CPU and the GPU. The CPU algorithm was implemented in standard C and compiled with optimization for speed. The simulation temperature is 30 K. As shown in Fig. 3, the time consumption increases as the number of atoms increases in both CPU and GPU implementation. Fig. 4 compares the two from a different perspective by showing the speedup of GPU over CPU. The speedup factor first increases as the system size increases, and then it varies between 10 and 11.

5. Conclusion

An equilibrium MD simulation of thermal conductivity based on programmable graphics hardware is presented in this paper. The calculated thermal conductivities of solid argon are consistent with the experimental data. The GPU-based implementation is faster than that of CPU-based one. For a 4000-atom system or a larger one, we get a speedup factor between 10 and 11. To attain these speeds, we have incorporated several optimization techniques into the GPU implementation.

Acknowledgements

We acknowledge Jesper Mosegaard and Youquan Liu for productive discussions. This work is supported by the National Natural Science Foundation of China (50506008 and 50505007).

References

[1] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Oxford Science Publication, Oxford, 1987.
[2] D. Geer, Taking the graphics processor beyond graphics, IEEE Computer 38 (2005) 14–16.
[3] I. Buck, T. Purcell, A toolkit for computation on GPUs, in: R. Fernando (Ed.), GPU Gems, Addison-Wesley, New York, 2004, pp. 621–636 (Chapter 37).
[4] N. Goodnight, R. Wang, G. Humphreys, Computation on programmable graphics hardware, IEEE Comput. Graph. Appl. 25 (2005) 12–15.
[5] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, A survey of general-purpose computation on graphics hardware, in: Eurographics 2005, State of Art Reports, Dublin, Ireland, 2005, pp. 21–51.
[6] M. Macedonia, The GPU enters computing's mainstream, IEEE Computer 36 (2003) 106–108.
[7] S. Tomov, M. McGuigan, R. Bennett, G. Smith, J. Spiletic, Benchmarking and implementation of probability-based simulations on programmable graphics card, Comput. Graph. 29 (2005) 71–80.
[8] W. Li, X. Wei, A. Kaufman, Implementing lattice Boltzmann computation on graphics hardware, Visual Comput. 19 (2003) 444–456.
[9] W. Li, Z. Fan, X. Wei, A. Kaufman, Flow simulation with complex boundaries, in: M. Pharr (Ed.), GPU Gems 2, Addison-Wesley, New York, 2005, pp. 747–764 (Chapter 47).
[10] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan, Brook for GPUs: streaming computing on graphics hardware, ACM Trans. Graph. 23 (2004) 777–786.
[11] D.C. Rapaport, The Art of Molecular Dynamics Simulation, second ed., University Press, Cambridge, 2004.
[12] A.J.H. McGaughey, M. Kaviany, Thermal conductivity decomposition and analysis using molecular dynamics simulations. Part I. Lennard-Jones argon, Int. J. Heat Mass Transfer 47 (2004) 1783–1798.
[13] M. Harris, Mapping computational concepts to GPUs, in: M. Pharr (Ed.), GPU Gems 2, Addison-Wesley, New York, 2005, pp. 493–508 (Chapter 31).
[14] J. Mosegaard, T.S. Sørensen, GPU accelerated surgical simulators for complex morphology, in: Proceedings of the IEEE Virtual Reality, Bonn, Germany, 2005, pp. 147–153.
[15] C. Woolley, GPU program optimization, in: M. Pharr (Ed.), GPU Gems 2, Addison-Wesley, New York, 2005, pp. 557–571 (Chapter 35).
[16] W.R. Mark, R.S. Glanville, K. Akeley, M.J. Kilgard, Cg: A system for programming graphics hardware in a C-like language, ACM Trans. Graph. 22 (2003) 896–907.
[17] K. Hillesland, A. Lastra, GPU floating-point paranoia, in: Proceedings of the ACM Workshop on General Purpose Computing on Graphics Processors, Los Angeles, USA, 2004, p. C-8.
[18] D.K. Christen, G.L. Pollack, Thermal conductivity of solid argon, Phys. Rev. B 12 (1975) 3380–3391.