RESEARCH ARTICLE OPEN ACCESS

Molecular dynamics Parallel simulation of Carbon Nanotubes based on GPU

Sheng Lai[1], Xiaohua Meng[2], Dongqin Zheng[3]
[1, 2](Department of Compute Science, Jinan University, Guangzhou 510632,China)
[3](Department of Physics, Jinan University, Guangzhou 510632,China)

----------**********----------

Abstract:

Molecular dynamics simulation is in-comparably superior to both experiments method and theoretical analysis. However, because computational effort of molecular dynamics simulation is very large, especially, the simulation of a large number of Carbon Nano Tube (CNT) particles, general CPU serial algorithm implementation is inefficient and slow. A Compute Unified Device Architecture (CUDA) based parallel algorithm of CNT molecular dynamics is proposed in this paper to take advantage of the data parallelism of Graphic Processing Unit (GPU). A CNT is divided to several blocks and processed parallel in the GPU. Experimental results show that the algorithm can obtain a speed-up more than 10 times to the CPU serial algorithm in a low-configured graphics card that has only 16 GPU stream processors.

Keywords **—Carbon Nano Tube (CNT),Molecular dynamics, Compute Unified Device Architecture (CUDA),Parallel computation,Time efficiency,Speed-up ratio**

----------**********----------

I. INTRODUCTION

Since 1957, ALDER et al. molecular dynamics have firstly been used to study the hard-ball systems, molecular dynamics has been widely applied in many fields such as crystal growth, indentation test, tribology and diamond synthesis with low- pressure, etc. In newly emerging field of the nanometer engineering, a macroscopic mechanism on the basis of continuous medium can hardly explain some particular phenomenon of nanometer engineering. Hence, Molecular dynamics became an important way for researches. In 1990, LANDMAN et al. used molecular dynamics to simulate the process of nano-indentation on Au (001) substrate surface by Ni-tip. The simulation results had been published on the journal Science, and considered to be a landmark achievement in this field. SCHIOTZd et al. took molecular dynamics simulation algorithm to analyze the plastic deformation of polycrystalline copper with different crystallite dimensions, and found that crystallite dimension and strength do not strictly follow the Hall-Petch relation.

Molecular dynamics simulation has advantages dramatically superior to both theoretical analysis and experiments. However, because of the extremely high requirements of computation and limitation of computer' capacity, molecular dynamics has always been highly concerned. To enlarge the scale of this simulation algorithm, many experts and scholars both in the domestic and overseas have done a great deal of research works. The simulation algorithm has been improved from the serial algorithm that was in order to enhancing the scale of single machine to the parallel algorithm by assigning the computation task to multiple CPUs to enlarge the simulation scale. And the number of atoms that involved in computation has also been increased from thousands to millions, or even hundred millions. Because CNT contains a large number of particles and has complex structure, its molecular dynamics simulation requires strong computation capability. When CPU simulates the serial algorithm, there is the disadvantage of great

amount of calculation, long computing time and great limitation of the particle number of simulation system.

Based on the CUDA platform, this paper studied and implemented the parallel simulation algorithm of molecular dynamics by controlling multiple GPU stream processing units and utilizing GUP's powerful parallel computation capability strong and efficient data transmission capability. The CNT system was divided into a parallelizable computational unit with multiple layers to calculate. Experimental results show that the GPU-based parallelizable simulation algorithm of CNT molecular dynamics implemented in CUDA platform can speed up computation, improve the execution efficiency and further increase the simulation calculation scale, which has great significance and effect to application of CNT.

II. BRIEF INTRODUCTION TO GPU AND CUDA

GPU, with the full name of Graphic Processor Unit, has now been widely used and becomes a powerful parallel processing unit. What's more, GPU is the processors with multi-threads and multi-cores that are very strong and efficient in parallel computation and data transmission. GUP is powerful in parallel computation and floating-point arithmetic, and efficient in data transmission. However, GPU also has its flaws. For example, the programming of GPU can be made only through graphics API (Application Program Interface), which is not convenient for programmers; GPU programming has large limitation and is not so flexible; What's more, because of the bottleneck of bandwidth (a transfer of data volume between GPU and video memory), GUP's computation capability cannot be fully played. Recent years, the performance of GPU has developed rapidly, and its memory bandwidth and floating- point arithmetic capability also have gotten great development.

CUDA (Compute Unified Device Architecture) is a computing platform developed by NVIDIA. CUDA is an infrastructure, which takes C language as the programming language and provides a great deal of high-performance computing instructions for development. In this way, we can establish a more efficient intensive data computation solution based on the powerful computing capability of the GPU. Generally, CUDA consists of two parts: host (operated in CPU) and device (operated in GPU). The birth of CUDA make GPU programming more convenient, and let GUP plays its powerful computing capability in many other fields rather than just limited in field of graphic processing.

Computation process of CUDA model can be split into 4 steps, as shown in Fig. 1.

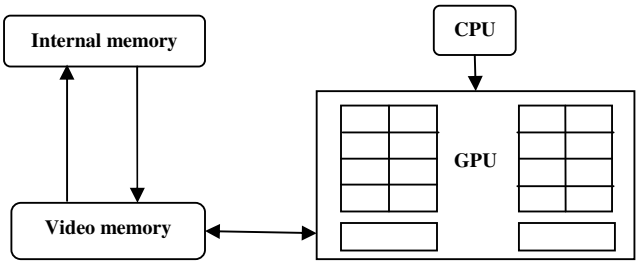

Fig. 1: Computation process of a CUDA

i) Copy the data that need to be processed from internal memory to video memory;
ii) CPU sends program instructions to GPU.
iii) Multiprocessor in GPU executes relevant instructions to the data in video memory. During computation, there may need frequent data exchange between GPU and video memory, and the final computation results are stored in video memory;
iv) Copy the computation results from video memory to internal memory;

III. VERLET SIMULATION ALGORITHM OF MOLECULAR DYNAMICS

Molecular dynamics is a set of molecule simulation methods, and is an important and effective method of computer-based atomic scale simulation. This method uses Newton mechanics to simulate the molecular motion. It samples from the moleculesystem in different status to calculate the system's configurations integrals. Based on the configuration integrals, we can calculate the thermodynamic quantities and other macroscopic properties. Thus, it is an important approach to simulate CNT with molecular dynamics.

Computations involved in Verlet algorithm mainly include following:

i) Do Taylor Expansion to $\vec{x}(t+\Delta t)$ and $\vec{x}(t-\Delta t)$,

as:

$$\vec{x}(t+\Delta t)=\vec{x}(t)+\vec{v}(t)\Delta t+\frac{\vec{a}(t)\Delta t^{2}}{2}+\frac{\vec{b}(t)\Delta t^{3}}{2}+o(\Delta t^{4}) \quad (1)$$

$$\vec{x}(t-\Delta t)=\vec{x}(t)-\vec{v}(t)\Delta t+\frac{\vec{a}(t)\Delta t^{2}}{2}-\frac{\vec{b}(t)\Delta t^{3}}{2}+o(\Delta t^{4}) \quad (2)$$

ii) Add Eq. (1) and Eq. (2) to obtain following position expression:

$$\vec{x}(t+\Delta t)=2\vec{x}(t)-\vec{x}(t-\Delta t)+\vec{a}(t)\Delta t^{2}+o(\Delta t^{4}) \quad (3)$$

iii) Make subtraction between Eq. (1) and Eq. (2), and divide both sides by 2Δt to obtain following velocity expression:

$$\vec{v}(t)=\frac{\vec{x}(t+\Delta t)-\vec{x}(t-\Delta t)}{2\Delta t}+o(\Delta t^{2}) \quad (4)$$

iv) According to Eq. (1)~(4), and with the conditions that the positions at time $t-2\Delta t$ and $t-\Delta t$ and the acceleration at time $t-\Delta t$, useVerlet algorithm to calculate integral: according to the positions at time $t-2\Delta t$ and $t-\Delta t$ and the acceleration at time $t-\Delta t$, the position at t (current time) can be obtained by putting $t=t-\Delta t$ into Eq. (3); according to the position at t (current time), the acceleration at t can be renewed by certain potential function; meanwhile, according to the positions at time t and $t-2\Delta t$, the acceleration at $t-\Delta t$ can be renewed by putting $t=t-\Delta t$ into Eq. (4). By now the positions at t and $t-2\Delta t$ and the accelerations at $t-\Delta t$ and t are all obtained. Repeat above steps.

In accordance with above description, useVerlet tabulation method to designthe serial simulation algorithm:

i) Construct a CNT structure model as shown in Fig. 2-4, and set the parameter values;
ii) Design the front-end and rear-end of caldarium;
iii) Construct and call the structure function that uses to calculate the internal forces of CNT, and the function that uses to figure out Van der Waals force between CNS (Carbon Nano Sphere) C60 and CNT;
iv) Figure out all forces, solve Newton's equation of motion, calculate in layers and do integration (the core of the algorithm);
v) Simulate the trajectory of CNS C60 that changesby the time of caldarium, and use the calculation results to draw up the curve of temperature that changes by the energy transformation inside the CNT.

IV. VERLET SIMULATION ALGORITHM OF MOLECULAR DYNAMICS

A. Establish Model Of CNT System

Read the database files for CNT system's parameters, such as initial temperature, number of particles, density and time, etc. Construct the CNT system model MOD1 and parameter model MOD2, of which the parameters can be used by MOD1, as shown in Fig.2~Fig.4. MOD1 consists of CNT and a football-shaped C60 (a molecule contains 60 carbon atoms, looks like a football, thus is also called fullerene) that set inside the CNT. This CNT, which has lamellar hollow structure and a quasi-circular structure body, consists of many hexagonal carbonic rings (composed by carbon atoms). Generally the diameter of the body ranges from one to dozens nanometers, and its length is far larger than the diameter. Initialize this model and read the speed and position coordinates of all particles of the model.

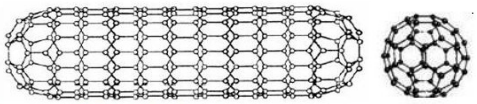

Fig.2 CNT (Carbon Nano Tube)Fig.3Molecule C60

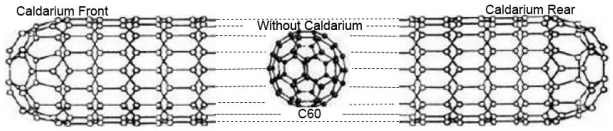

Fig.4 CNT system model

In CNT, the rest particles form hexagonal except the nozzle layer particles. In Fig.5, Non nozzle particles have 3 nearest neighbor (ip, jp and kp are the nearest neighbor of idx).

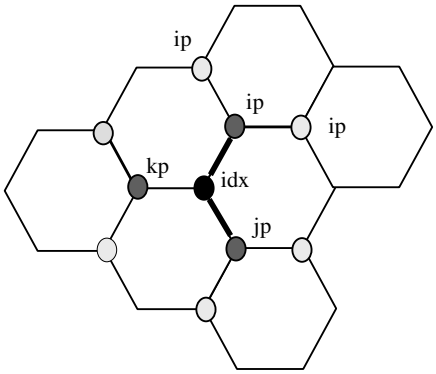

Fig. 5Carbon nanostructures

In CNT, the rest particles form hexagonal except the nozzle layer particles. In Fig.5, Non nozzle particles have 3 nearest neighbor (ip, jp and kp are the nearest neighbor of idx).

In CNT, the rest particles form hexagonal except the nozzle layer particles. In Fig.5, Non nozzle particles have 3 nearest neighbor (ip, jp and kp are the nearest neighbor of idx).

$$\xi = \frac{\Sigma_i f_i v_i}{m\Sigma_i v_i^{\,2}}$$

B. Study The Parallel Simulation Algorithm In CUDA Platform

Divide the CNT on CUDA platform into independent computational units with appropriate sizes. Such division is to improve the degree of parallel under the precondition of avoiding too much repetitive computation. The divided computational units must be of reasonable dimensions and thicknesses, for, too large computational units will result in unapparent parallel, while too small will result in unnecessary repetitive computation. (Due to the large influence of the distances between particles' forces, a distance threshold must be set to judge the positional relationship. When distances smaller than the threshold, particles can be considered as nearest particles; when distances larger than the threshold, they can be considered as next-nearest particles, and the molecular forces between them can be ignored.)

It can be expected that each of the central particle will apply a force to its up to 3 nearest neighbors and 2 neighbors of this 3 neighbors, that is, 6 next-nearest neighbors. Think of it this way, dividing each of central particle and its 3 nearest neighbors and 6 next-nearest neighbors into a group to calculate. And parallel computing in batches. In a computing batch, a particle, which was not considered as the central particle, can be took into the parallelizable computing queue, however, because of competitive relation, its 3 nearest and 6 next-nearest particles can not be took into computing queue in this batch. Similarly, the other particles without calculation, which applied a force to this 9 particles, can not be took into the queue. In this way, the particles in the same batch will be able to parallel computing without producing a competitive relationship.

Design split algorithm, using CUP traversal computation units to get *n* parallelizable computing queues:

i) If all particles are computed as central particles, then jump to step h);
ii) Find the first non-competitive particle, which was not be computed as central particle, add it to the parallelizable computing queues;
iii) Mark all nearest neighbors of this particle as the first-degree unparallelizable particles; if a neighbor has already been regarded as second-degree unparallelizable particle, change it to the first-degree uncomputable particle;
iv) Mark all next-nearest neighbors of the same particle as the second-degree unparallelizable particles; if a neighbor has already been regarded as a first-degree unparallelizable particle, then its degree will not be modified;
v) Regard this particle as the central particle that has been computed;
vi) Whether has traversed to the end of particle queue? If yes, continue; otherwise jump to step a);
vii) Start the next parallel queue, come back to step b);
viii) End.

The split algorithm completes in the stage of CPU preprocessing. The parallelizable queues, which were computed once, can be used in later

circular computing. When the number of circular time is large, the proportion of time that the split algorithm cost will be greatly reduced and its affect to the program performance can also be reduced to a low degree.

As shown in Fig.6 and Fig.7, use GPU stream processing units and Verlet algorithm to make computation and treatment, as follows:

i) According to positions of all particles of the CNT system model, calculate the bond and angel relationship between nearest neighbor particles and between next-nearest neighbors;

ii) Control GPU stream processing units to calculate particles in the different divided computational units, and compute the interaction forces between each particle of C60 and each particle of CNT wall;

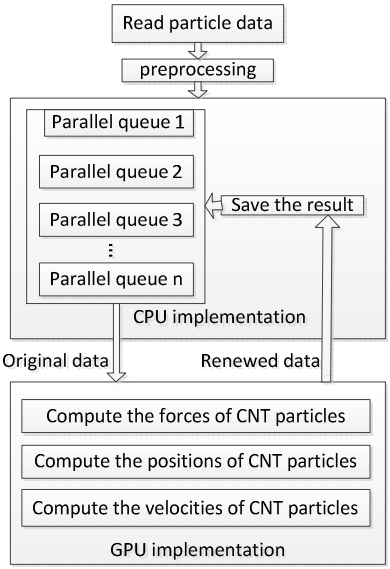

Fig.6 Dispatching mode of CPU and GPU in CNT molecular dynamics parallel simulation

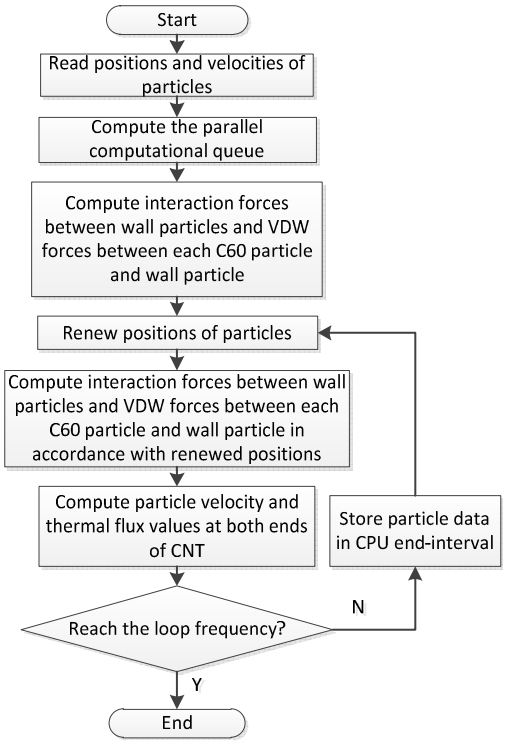

Fig. 7 Flowchart of parallel simulation algorithm of GPU-based CNT molecular dynamics

iii) According to the area that C60 is located, calculate interaction forces (Van der Waals force) between each C60 particle and each CNT particle;

iv) Renew particle position in accordance with the force and velocity of the particle, then implement steps b) and c);

v) Calculate the velocity of the particle in accordance with the force of particle, and compute the thermal flux values at front-end and rear-end of the CNT; and,

vi) If reach the loop frequency, then end the computation; otherwise, save particle data in end-interval of CPU, and return to step d).

V. OPTIMIZATION OF GPU PARALLEL ALGORITHM

Verlet simulation program is a cyclical process of "calculate force, calculate position and calculate speed". Each particle runs in a separate thread and then circulates thousands of times or more. Thus, optimizing a fraction of program will make whole program have great improvement of performance.

For the CUDA program, optimization mainly focuses on instruction, memory access, grid optimization and resource balance.

A. Instruction Optimization

At present, GPU single precision computation performance is much better than double precision computation performance, therefore, the part of program which less demands on the accuracy can be used in single precision arithmetic instead of double precision arithmetic.

As can be seen, the instruction throughput is limited in integer multiplication, division and modulo operation of GPU, and it is expensive to interrupt instruction stream because control flow instructions effect the parallel transmission of instruction transmitter. Here, we use 24-bit integer arithmetic __mul24 and __umul24 in the optimized library to replace 32-bit integer arithmetic and use __sinf (x), __fdivide (x, y) etc. to replace the corresponding operation. As far as possible to avoid integer division and modular arithmetic, or use i&(n − 1) to replace i%n and use shift operations to replace division when the divisor is power of 2.

B. Memory Access Optimization

CUDA memory is the storage modelcomposed of a variety of storage hierarchy and is significantly different from CPU memory. Corresponding memory hierarchy of the various threads structure is shown in Fig.8.

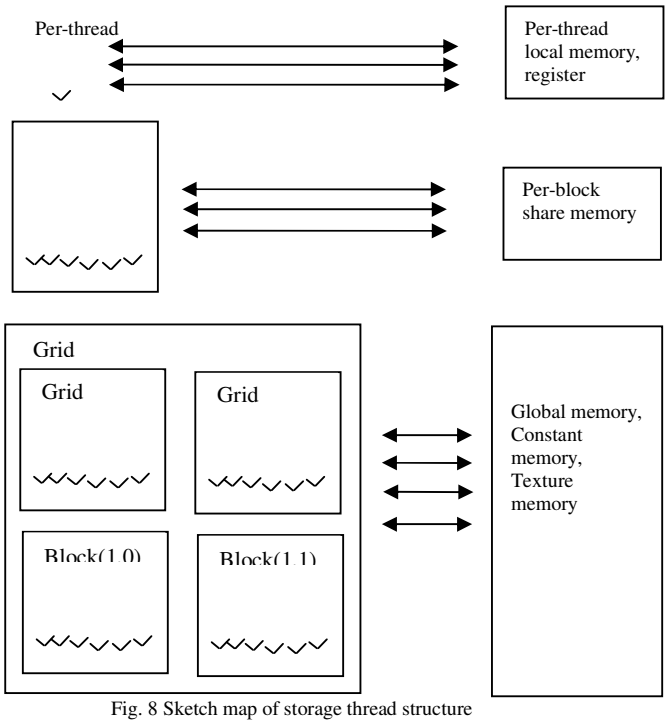

Fig. 8 Sketch map of storage thread structure

Register, share memory and constant memoryare chip high speed buffer on the GPU, and also scarce resource.So we must be reasonable to use share memory resources in the program, to reduceunnecessary temporary variables, and use the reduc-ed function to reduce the use of register. At the same time, the constant in the program andcombin-ed constant in the calculationstored in constan-t memory.

Global memory is the storage with maximum capacity in video memory, which can be to read or written to storage at any position by arbitrarily GPU thread. It can provide hide, bandwidth but also has high memorylatency because of no global memory cache. A single memory access delay reaches a 400~600 clock cycle.Therefore, the program reducesunnecessary accesses to global memory, and dramatically increase the processor utilization rate, namely when one block is accessing the memory, another can be scheduled for computations to hide the access latency.In addition, we can change the storage structure of video memory and try to meet combined access memory.

C. CPU + GPU Processing

To develop its powerful parallel computing ability, GPU parallel processing need tohave enough threads in parallel operation. Therefore CPU implements the part of smaller calculation amount. For example, because C60 has only 60 particles and small parallelism degree, put the calculation of Van Der Waals Force between C60 and CNT into CPU. Because the kernel boot is asynchronous, the CPU function is below the kernelfunction. It implements parallel processing for CPU and GPU that CPU starts the kernel and performs a GPU operation and then returns immediately to execute a CPU program.

VI. ANALYSISOFEXPERIMENTALRESULTS

This test is performed on the platform of Widows 7. GPU1 and GPU2 are different types of graphics card. Only one card is selectedeach experiment. For recording the running time and providing the convenience of the CPU and GPU parallel operation, test time adopts the CPU time. The following is hardware parameters:

TABLE1. PARTIAL HARDWARE PARAMETERS OF TEST PLATFORM

	CPU	GPU 1	GPU 2
Processor	Intel Xeon W3550	NVIDIA GeForce GT 430	NVIDIA Quadro 2000
Clock Speed	3066MHz	1400MHz	1251MHz
Cores	4×8	2×48	4×48
Internal /Video memory	8GB	2GB	1GB

A. *Parallel Algorithm Performance Comparison Between Different GPU*

Using GeForce GT 430 graphics card and Quadro 2000 graphics card, we can calculate the two time with different particle number.

TABLE 2. THE DATA OF PARALLEL PROGRAM THAT RUN ON DIFFERENT HARDWARE PLATFORMS.

Number of particles layer	Number of particles	Time of GPU1 (ms)	Time of GPU2 (ms)
50	1000	4462	4134
100	2000	6412	5632
200	4000	9625	7379
300	6000	12168	9282
600	12000	21656	14352
900	18000	30919	19828
1200	24000	40763	25178
1500	30000	50552	30420
3000	60000	98155	56855
6000	120000	194038	109996

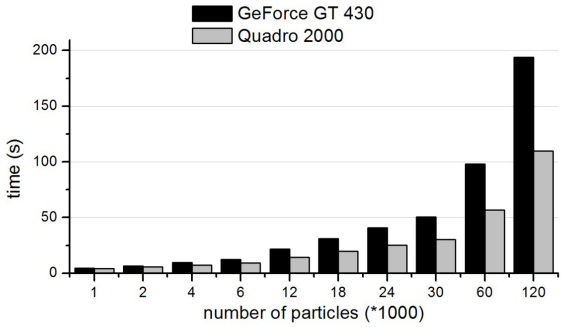

Fig.9 Two types of graphics program time-consuming comparison

From Fig.9, when the number of particles is small, two types of graphicstime little difference. But with the increasing of particle number, GeForce GT 430 graphics card timeincreases faster than Quadro 2000 graphics card. When the particle number increased to 120000, the time of former is almost 2 times of the latter.

Through horizontal comparison in Table 1 and Fig.9, the number of CUDA Cores (i.e. stream processor) in Quadro 2000 graphics card is the double of in GeForce GT 430graphics card. Combined with running way of GPU threads, running thread will increase accordingly accompany processing unit of GPU increasing. Therefore, when no other hardware limitations, GPU parallel computing performance will increase with increaseing of GPU processor. It can bring more performance improvement for Verletalgorit-hm to use the GPU with more processors.

B. *Serial And Parallel Algorithms Performance Comparison*

In order to compare the performance of GPU parallel program with CPU serial program, we start the corresponding program, and calculate the time of different particle numbers. We choose Quadro 2000 as GPU, Get Table 3 data:

TABLE 3. TIME OF GPU PARALLEL PROGRAM COMPARED WITH CPU SERIAL PROGRAM.

The layers of particle	The number of particles	Serial time (ms)	Parallel time (ms)
50	1000	15912	4134

100	2000	29172	5632
200	4000	55210	7379
300	6000	85172	9282
600	12000	163544	14352
900	18000	238150	19828
1200	24000	322000	25178
1500	30000	400878	30420
3000	60000	835889	56855
6000	120000	1675152	109996

Tabular data are plotted into the serial and parallel program execution time comparison chart:

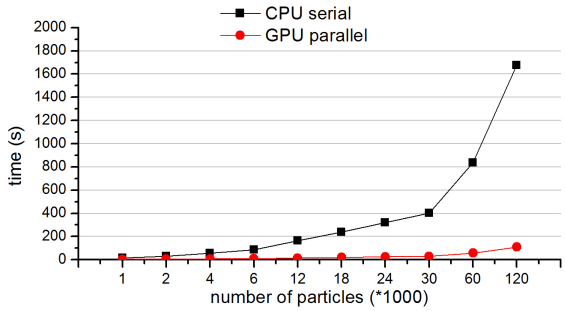

Fig.10. Time of GPU parallel program compared with CPU serial program.

We can clearly see that, when the number of calculated particles increasing, the speed-up ratio of parallel program increases compared to the serial program. When the number of particles in the 1000~4000, GPU load time of the program, data transmission time and access time are not very well hidden because the thread is too little. The speed-up effect is better and better with increasing the number of particles.

C. The Parallel Algorithm Performance Analysis

To analyze the speed-up ratio of parallel algorithm, we calculate serial and parallel program computing time in table 3 according to the following formula:

$$speed-up \quad ratio = \frac{Serial \quad time - Parallel \quad time}{Parallel \quad time} \times 100\%$$

And then draw into GPU parallel computing speed-up ratio (Fig. 11):

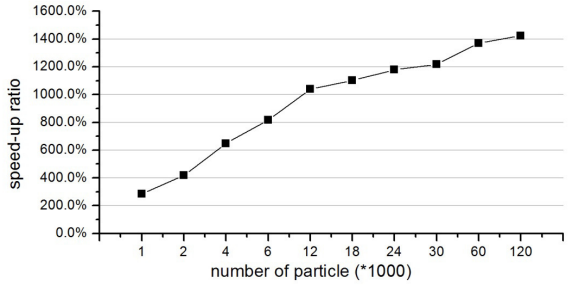

Fig.11 speed-up ratio of GPU parallel calculation

As can be seen, the speed-up ratio of parallel algorithm is increasing compared to the serial algorithm with increasing the number of particle. When the particle number reaches 12000, speed-up ratio reaches 1400%, i.e. speed-up effect is 14 times.

For the analysis the factors affecting GPU parallel program performance, we analyze program with the performance evaluation tool NVIDIA Visual Profiler. There are 12000 particles of 600 layers calculated in the target program and 128 threads in each block. We calculate each function running time accounted for the total operation time ratio through several arithmetic average calculations, and draw the map 12.

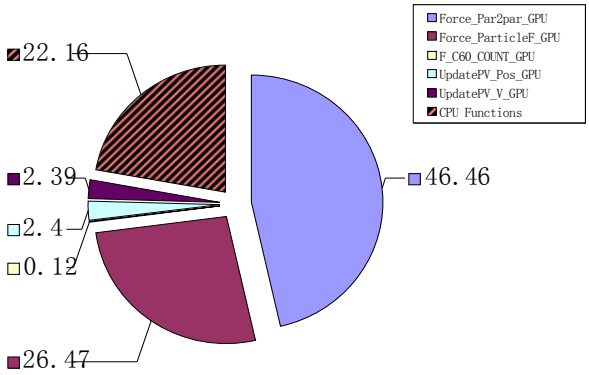

Fig.12 proportion chart of function

VII. CONCLUSION

In the simulation of CNT molecular dynamics, originalmolecular dynamics simulation has good effect. But because of the scale of number of CNT system's particles is too large, it lacks practicality due to the huge computation and long operation time.Molecular dynamics simulation algorithm with using Verlet algorithm has the

advantages of easy parallelism, and can be parallel processed in the GPU, which has powerful image processing, and floating-point computation ability, which greatly improves the efficiency of CNT molecular dynamics simulation algorithm.

Experimental results show that for small-scale CNT systems with only 12000 particles, when the algorithm is implemented through parallel GPU-based simulation algorithm running on NVIDIA Quadro 2000 with 192 stream processors, the speed-up ratio can reach much more than 10 times compared to the serial CPU-based algorithm implemented on the 4-core-8-thread Intel Xeon W3550. Thus it can efficiently overcome the defect of the algorithm on processing speed.

However, the CNT system scale, which can be computed, is limited due to limited test hardware, so it cannot deeply dig the ability of CUDA. Experimental results shown in (Fig. 11), may wish to speculate, particle number scale of CNT system is larger, the speed-up effect of GPU parallel algorithm is more obvious compare serial algorithm. In practice, the particle number of CNT system is in a big scale, more than 10 million and one billion. It is estimated that parallel simulation algorithm of GPU-based CNT molecular dynamics will play a role of imagination in research on CNT.

REFERENCES

[1] Alder B J, Wainwright T E. “Phase transition for a hard sphere system”. *The Journal of Chemical Physics*, 1957, 27(5): 1208.

[2] Landman U,Luedtke WD, Burnham NA, et al. “Atomistic mechanisms and dynamics of adhesion, nanoindentation and fracture”. *Science*, 248, pp.454-461, 1990.

[3] Schiøtz J, Jacobsen K W. “A maximum in the strength of nanocrystalline copper”. *Science*, 2003, 301(5638): 1357-1359.

[4] Verlet L. “Computer‘experiments’ on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules”. *Physical review*, 1967, 159(1): 98-103.

[5] Tamayo P, Mesirov J P, Boghosian B M. “Parallel approaches to short range molecular dynamics simulations”. *Proceedings of the 1991 ACM/IEEE conference on Supercomputing. ACM*, 1991: 462-470.

[6] Meng X H, Liu J Q, Ou Y X, et al. “Laplacian Edge Detection Algorithm Based on CUDA”. *JisuanjiGongcheng/ Computer Engineering*, 2012, 38(18).

[7] Proctor A J, Lipscomb T J, Zou A, et al. “Performance Analyses of a Parallel Verlet Neighbor List Algorithm for GPU-Optimized MD Simulations”. *Biomedical Computing (BioMedCom), 2012 ASE/IEEE International Conference on. IEEE*, 2012: 14-19.

[8] Zhang Y, Zhao J, Yuan Z, et al. “CUDA Based GPU Programming to Simulate 3D Tissue Deformation”. *Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on. IEEE,* 2010: 1-5.

[9] Alder B J, Wainwright T E. “Studies in Molecular Dynamics. I. General Method”. *Journal of Chemical Physics*, 1959, 31: 459-466.

[10] Fei H, Zhang Y Q, Wang K, et al. “Parallel Algorithm and Implementation for Molecular Dynamics Simulation Based on GPU”. *Computer Science*, 2011, 38(9): 275-278.

[11] Anderson J A, Lorenz C D, Travesset A. “General purpose molecular dynamics simulations fully implemented on graphics processing unit”. *Journal of Computational Physics*, 2008, 227(10): 5342-5359.

[12] NVIDIA_CUDA_ProgrammingGuide_3.0. http://developer.nvidia.com/object/cuda_3_0_downloads.html.