A novel parallel algorithm for near-field computation in N-body problem on GPU

Jianchen Shan
School of Computer Engineering and Science
Shanghai University
Shanghai, China

Yongmei Lei corresponding author
School of Computer Engineering and Science
Shanghai University
Shanghai, China

***Abstract*—A novel efficient parallel algorithm for the near-field computation in N-body problem on the Graphics Processing Unit (GPU) architecture is proposed in this paper. This algorithm evolved from the BPB algorithm [1] which is proposed in the author's previous work. This novel algorithm is based on the Newton's third law and Z-order Space Filling Curve (Z-SFC). Half of the computations are reduced and the highest speedup of the Compute Unified Device Architecture (CUDA) implementation compared to the serial CPU implementation reaches 326. Through the quantitative analysis of this algorithm, we propose the GPU optimization model of transformation between data access and computation.**

Keywords-parallel algorithm; GPU; CUDA; N-body; near-field computation; Newton's third law; Z-SFC

I. INTRODUCTION

Nowadays with the development of the CUDA, the general-purpose computation on graphics processing units (GPGPU) has become the hot topic in the field of high performance computing. More and more typical scientific applications including N-body simulation have been mapped on to computational platform of the GPU architecture for its features of high performance, relatively low-cost and easy to programming.

In N-body problem, the system is described by a set of N particles, and the dynamics of the system is the result of the interactions that occur for every pair of particles. In order to reduce its O (N^2) computational complexity, the researchers proposed the tree algorithms of which the Fast Multipole Method [2] is the most famous one that reduce the complexity to O (N). The key idea of tree algorithm is to classify the force on particles into two types: near-field force and far-field force. The approximation is applied to far-field interactions, while the near-field interactions are summed directly. Although many efforts have been made to map various algorithm of N-body problem to GPU, in most reported studies the simple O (N^2) algorithm was used for GPGPUs. Therefore, this paper focuses on mapping the near-field computation which is the important and common component of various three algorithms on to the GPU.

In our previous work, three mapping strategies (BPT, PPT and BPB) [1] are proposed of which the BPB algorithm is the most efficient one, because it optimizes the task partition to maximize the CUDA threads [3] and take advantage of the shared memory of GPU for data reuse to break the performance bottleneck caused by frequent data access to the global memory. The highest speedup obtained by BPB compared to the serial CPU implementation is 243. However, the amount of computation is still huge. According to the Newton's third law, the force between two interacting particles comes in pairs-equal and opposite-which doesn't need to be calculated repeatedly. Based on this theory, half of the amount of computation in near-field force could be reduced. And this paper tactfully makes use of the Z-SFC to implement the improved BPB algorithm based on the Newton's third law. The highest speedup obtained is 326. Furthermore, by analyzing the change of the amount of computation and data access from BPB algorithm to the improved BPB algorithm, this work introduce a novel GPU optimization model that enhancing the performance by transformation between data access and computation.

The paper will be organized as follow: Chapter Ⅱ describes the algorithm of the near-field computation. And in chapter Ⅲ, the improved BPB algorithm would be presented after the description of the data structure used on GPU and the previous BPB algorithm. In chapter Ⅳ, the theoretical amount of the computation and data access of the previous BPB algorithm and the improved BPB algorithm would be quantitatively analyzed and compared with each other. And the experiments are executed to demonstrate this quantitative analysis. Based on the analysis in Chapter Ⅳ, Chapter Ⅴ proposed a novel GPU optimization model of transformation between data access and computation. At last the conclusion of the article and the introduction of the future work are stated in chapter Ⅵ.

II. THE DESCRIPTION OF NEAR-FIELD COMPUTATION

A. *Space Decomposition and Encoding*

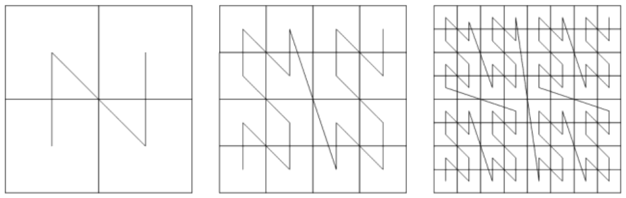
Figure 1. Space encoding by Z-SFC (K=1,2,3)

The algorithm is discussed in two-dimensional space which is divided by a quadtree into many small areas of the same size, and these small areas are called boxes. A box is defined as the father box of the particles in it, and the boxes around it are called its neighbor boxes. In order to organize and sort the initially disordered particles, each box needs to be set an index, and this process is called space encoding. In this paper, the Z-SFC [4] is employed to encode the space into one dimension as shown in Fig. 1. K is the height of the quadtree.

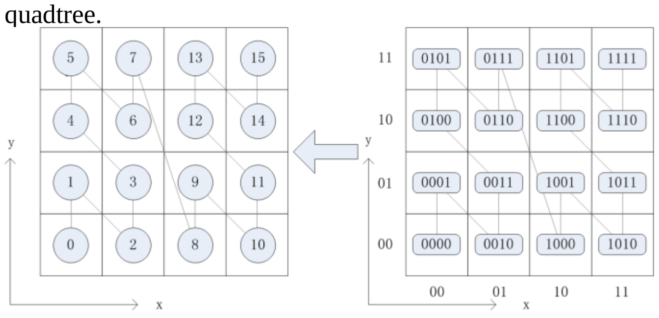

Figure 2. The principle of Z-SFC (K=2)

If K is 2 and the coordinates origin is in the lower left corner, the space is encoded as Fig. 2 (left side). Each index of box contains the location information of the box in the space as shown in Fig.2 (right side). If the coordinates of the box is presented by K-digit binaries, then, for example, the box 3 ($3_{10}=0011_2$) is the interlaced combination of the coordinates of it, which is (01, 01).

B. *The Outline of Algorithm*

The near-field force on each particle consists of two parts: (1) the force that comes from the other particles in its father box; (2) the force that comes from the particles in its father box's neighbor boxes. Thus the serial algorithm of near-field computation is to calculate the near-field force that comes from those two parts on each particle of each box in order of the boxes' indexes.

III. The Parallel Algorithm for the Near-field Computation on GPU

A. *Data Structure*

There are four liner arrays created in GPU for calculation. First array contains the data of the particles of all boxes ordered by their indexes-all data from box 0 is at the start of the memory block, all data from box 1 follows and so on. The second liner array called boxIdx is created to store the indexes of the start position of the particles in every box in the first liner array. The third liner array is the neighborlist array that contains the neighborlist of each box. The forth liner array called acc which stores the near-field force on each particle.

B. *Previous BPB Algorithm*

The BPB (box per block) algorithm has the most efficient CUDA implementation in our previous work. In this strategy, one CUDA block [3] handles the computation of the particles in one box. In case the number of particles in a box is n, the first n threads of the corresponding block handle the computations of these n particles and other threads of this block are idle. When each particle's near-field force that comes from other particles in its father box is calculated, all data of this box would be loaded to the shared memory for data reuse. When each particle's near-field force that comes from the particles in its father box's neighbors is calculated, the data of each neighbor would be loaded to the shared memory one after another. Fig. 3 is an overview of the order of loading data to the shared memory, when the near-field force on every particle in box A is calculated.

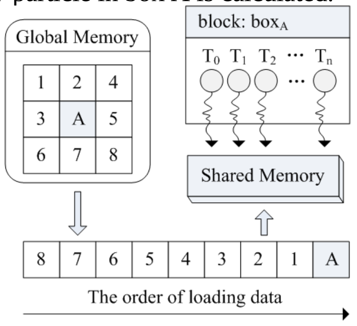

Figure 3. Loading data from global memory to shared memory [1]

The BPB algorithm breaks the performance bottleneck caused by frequent data access to the global memory. The highest speedup obtained by BPB compared to the serial CPU implementation is 243.

C. *Improved BPB Algorithm*

1) *The Principle of the Improvement*

Although the BPB algorithm has made great efforts to optimize the task partition and data access on GPU, the amount of computation is still huge. Thus a novel improved BPB algorithm based on the Newton's third law is proposed. According to the Newton's third law, the force between two interacting particles comes in pairs-equal and opposite-which doesn't need to be calculated repeatedly. Based on this theory, half of the amount of computation in near-field computation could be reduced. There are two steps of calculating the near-field force on the particles in a box: (1) calculating the force that comes from the other particles in this box; (2) calculating the force that comes from the particles in this box's neighbor boxes. Therefore, we would respectively describe the specific principle of reducing the amount of computation of each step.

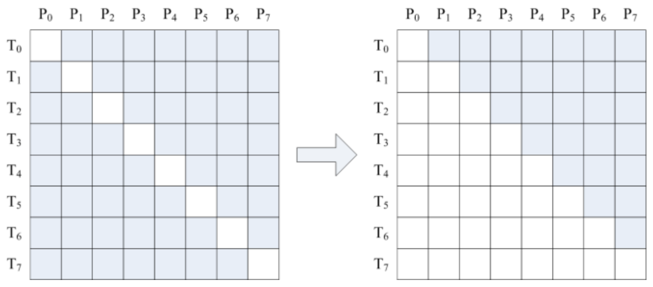

Figure 4. The principle of improvement in the first step (Ti represents the cuda thread that calculates the near-field force on particle$_i$, P_j represents the particle$_j$, and the grey suqare represents the calculation of the force between particle$_x$ and particle$_y$. (x,y) is the coordinates of this square)

In the first step, Fig.4 (left side) describes the calculation of one box in which there are eight particles before improvement. Each particle should calculate the force comes from all other seven particles, and when calculate the force of particle i on particle j, T_i only update the near-field force on particle i. Actually T_i can simultaneously update the near-field force on particle j with the equal and opposite result. Under this circumstance, there is no need for T_j to repeatedly calculate the force of particle i on particle j. Based on this principle, the calculation of this box in the first step is improved as shown in Fig.4 (right side).

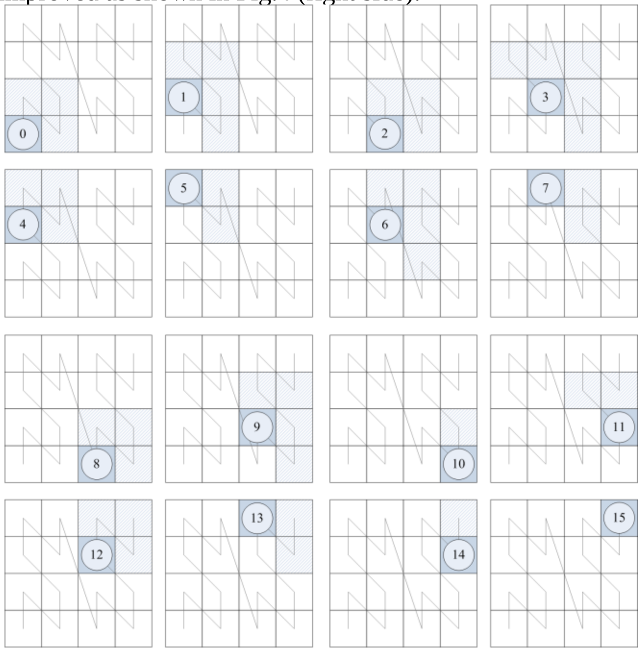

Figure 5. The principle of improvement in the second step

In the second step, when we calculate the near-field force on box A's particles from the particles in one of its neighbors called box B. The near-field force on the particles in box B from box A could be updated simultaneously. When we handle the each box's calculation of the second step in order of the index of Z-SFC, as shown in Fig.5, we just need to calculate the near-field force of the particles in the light grey boxes on the particles in the dark grey box. Thus the following rule could be concluded: in the second step, only the interactions between one box and its neighbors whose indexes are bigger than that of this box should be handled.

2) The Sequence of the Algorithm

Based on the above principle, the neighborlist array on GPU should be simplified at first. Each box's neighborlist only need to contain the neighbors whose indexes are bigger than its own index. The improved BPB algorithm also employs one CUDA block to handle the computation of the particles in one box. In case the number of particles in a box is n, the first n threads of the corresponding block handle the computations of these n particles and other threads of this block are idle. The sequence of the improved algorithm is as follow:

Step 1: Each block applies for two liner memory blocks for creating two arrays, array A and array B, in the shared memory on GPU. Array A is used for storing the data of the corresponding box's particles. Array B is used for storing the counterforce result of each interaction between a pair of particles.

Step 2: Each of the first n threads of a block loads the particles in the corresponding box into array A in the shared memory respectively and synchronizes to make sure that all n particles have been loaded.

Step 3: Each of the first n threads of a block applies for a local variable called d_acc to store the near-field force on the corresponding particle, and then reads sequentially the other n-threadIdx.x-1 (threadIdx.x is the index of a CUDA thread) particles from array A for computing. The results of the force and the counterforce of each interaction should update d_acc and array B respectively, and then synchronizes to make sure that all computations have been completed. For example, if A[i] is read, the result of counterforce would be used to update B[i]. At last each of the first n threads utilizes B [threadIdx.x] to update the acc[j] in the global memory and synchronizes again to make sure that the data of array B has been written to the global memory. The j represents the index of the thread's corresponding particle.

Step 4: The treads of a block are employed to load the particles in the first neighbor box of the corresponding box to array A to overwrite the previous data and synchronize to make sure that all particles in this neighbor box have been loaded, and then set each element of array B to zero.

Step 5: Each of the first n threads of a block sequentially read the particles in array A for computing. The results of the force and the counterforce of each interaction should update d_acc and array B respectively, and then synchronizes to make sure that all computations have been completed. At last each of the first n threads utilizes B [threadIdx.x] to update the acc[j] in the global memory and synchronizes again to make sure that the data of array B has been written to the global memory. The j represents the index of the threadIdx.x-th particle in the neighbor box calculated.

Step 6: If all neighbor boxes of a block's corresponding box have been once loaded to array A for computation, the computation is completed. Otherwise the algorithm goes back to the Step 4 and the particles of the next neighbor box would be loaded to shared memory to overwrite the previous data in array A for computation.

Step 7: Each of the first n threads of a block utilizes the corresponding local variable d_acc in the register to update acc[j] in the global memory respectively and. The j represents the index of the thread's corresponding particle.

Fig.6 and Fig.7 illustrate the data flows of the two procedures of calculating the near-field force on the 6 particles in box A. T_i represent the CUDA thread i of a block and this block's corresponding box is box A. The reason for creating array B to store the counterforce is to convert the write operation on the global memory to the write operation on the low-latency shared memory. And in step 3 and step 5 there exist the situations that many threads simultaneously updating the same variable, which may lead to an error result. Thus the atomic function in CUDA C [3] is employed to solve this problem. Because the atomic operation would serialize the updating operations of different threads on the same variable, the performance of this implementation would be harmfully influenced, which would be demonstrated in the results of the following numerical experiments.

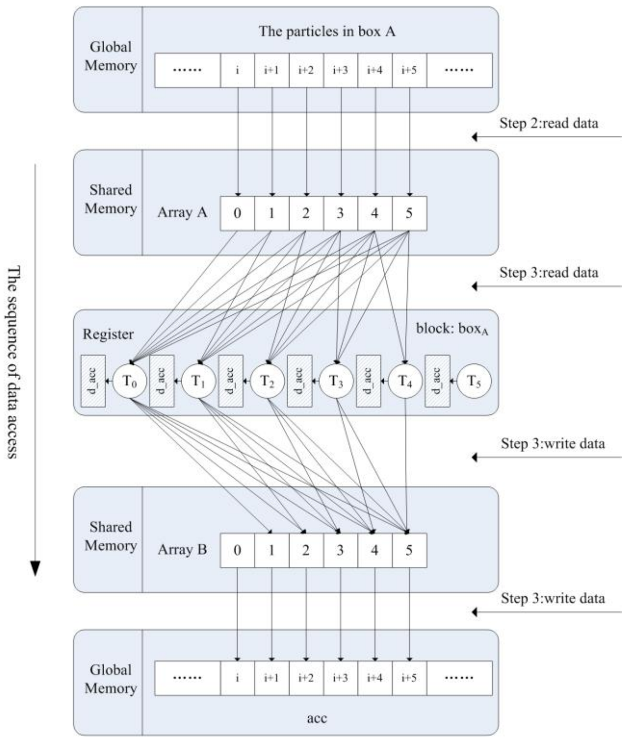

Figure 6. Data flow of calculating the particle's near-field force that comes from other particles in its father box

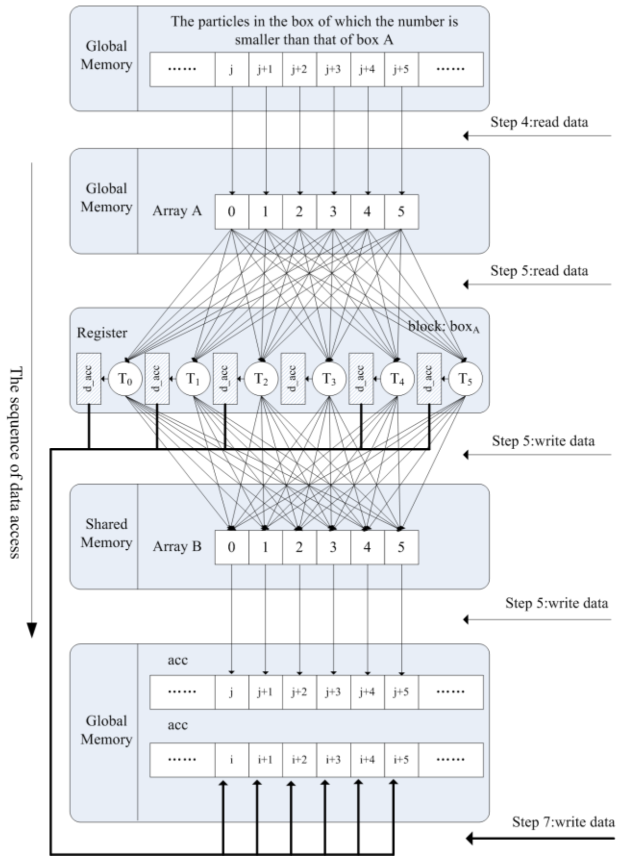

Figure 7. Data flow of calculating the particle's near-field force that comes from the particles in its father box's neighbor

IV. Theoretical Analysis and Numerical Experiments

In this section, we first conduct the quantitative analysis of the amount of the computation and data access of the previous BPB algorithm and improved BPB algorithm to show the improvement of the improved BPB algorithm, and then execute the numerical experiments to demonstrate the superiority of the improved BPB algorithm based on the Newton's third law.

A. Theoretical Analysis

Since the gravitational force between pair of particles is calculated, each interaction between a pair of particles contains the same number of floating point operations, which is 15 (2 subtractions, 4 additions, 7 multiplications, 1 division and 1 square root). The criterion to measure the amount of computation could be the number of interactions between a pair of particles. Furthermore, each write or read operation of the two different algorithms is conducted on a single-precision floating-point number. Thus the criterion to measure the amount of data access could be the times of data access.

TABLE I. Comparison of the Previous BPB Algorithm and the Improved BPB Algorithm from the Aspects of the Amounts of Computation and Data Access

Procedure		***Father box***	***Neighbor box***	***Return d_acc***
Previous BPB	***GR(times)***	3N	3BP	
	GW(times)			2N
	SR(times)	3N(P-1)	$3BP^2$	
	SW(times)	3N	3BP	
	RR(times)			2N
	RW(times)	2N(P-1)	$2BP^2$	
	CP(times)	N(P-1)	BP^2	
Improved BPB	***GR(times)***	3N	3BP/2	
	GW(times)	2N	BP	2N
	SR(times)	3N(P-1)/2+2N	$3BP^2/2+BP$	
	SW(times)	3N+2N+N(P-1)	$3BP/2+BP+BP^2$	
	RR(times)			2N
	RW(times)	2N(P-1)/2	BP^2	
	CP(times)	N(P-1)/2	$BP^2/2$	

B is the sum of all boxes' neighbors, ***P*** is the number of particles in each box, and ***N*** is the number of particles in the system. ***GR*** (global memory reading) and ***GW*** (global memory writing) respectively represent the number of the read or write operations on the global memory. ***SR*** (shared memory reading) and ***SW*** (shared memory writing) respectively represent the number of the read or write operations on the shared memory. ***RR*** (global memory reading) and ***RW*** (global memory writing) respectively represent the number of the read or write operations on the register. ***CP*** (computation) represents the number of interactions between a pair of particles. The procedure ***Father box*** is to calculate the particle's near-field force that comes from the other particles in its father box. The procedure ***Neighbor box*** is to calculate the particle's near-field force that comes from the particles in its father box's neighbor boxes. The procedure ***Return d_acc*** is to utilize the local variable d_acc in the register to update the array acc in the global memory.

Tab.1 analyzes the amounts of computation and data access in different procedures of the near-field computation in the previous BPB algorithm and the improved BPB algorithm. It shows clearly that the amount of computation is half reduced in the improved BPB algorithm. The change of the amount of data access in different procedures is discussed as follow:

In the procedure ***Father box***, because the array B in the shared memory is employed in the improved BPB algorithm to store the counterforce, the number of write operations on the shared memory is increased. Because the data of array B is used to update the array acc in the global memory, the number of the read operation on the shared memory and the number of the write operation on the global memory are increased. Furthermore, since the number of interactions between a pair of particles is half reduced in this procedure, the write operation on d_acc in register is also half reduced.

In the procedure ***Neighbor box***, because only B/2 neighbor boxes would be loaded from global memory to the shared memory for computing in the improve BPB algorithm, the numbers of read operation on global memory and write operation on the shared memory for loading neighbor boxes from global memory to the shared memory are half reduced. In addition, the number of read-write operations on the array A in the shared memory is also half reduced. However, the number of the write operations on array B in the shared memory is increased to store the counterforce. At last the number of read operation on the shared memory and the number of write operation on global memory are increased, for the data of array B in the shared memory is used to update the array acc in global memory.

In the procedure ***Return d_acc***, the amount of data access is not changed.

TABLE II. THE MACRO CHANGE OF THE AMOUNT OF COMPUTATION AND DATA ACCESS IN THE IMPROVED BPB ALGORITHM

Change of the amount of computation	**Change of the amount of data access**		
	Read-write operations on register(times)	***Read-write operations on shared memory(times)***	***Read-write operations on global memory(times)***
$-N(P-1)/2 - BP^2/2$	$-N(P-1)-BP^2$	$-N(P-1)-BP^2$	$2N - BP/2$
When tree_level=7, N=16384P, B=129540 and P≧1			
Reduced by P(145924P-16384)/2	Reduced by P(145924P-16384)	Reduced by P(146462P-89346)	Reduced by 32002P

The above analysis indicates that both increment and reduction of the amount of data access to multilevel GPU storage space exits in different procedures of the near-field computation in the improved BPB algorithm. Therefore, the macro analysis of the change of the amount of the computation and data access in the improved BPB algorithm is presented in the Tab.2. It shows that the amount of computation and the amount of data access to register are half reduced. However, the changes of the amounts of data access to the shared memory and global memory are determined by the height of the quadtree and the number of particles in each box. In the real numerical experiments, the height of the quadtree is set to 7, if there is at least one particle in each box, the amounts of data access to the shared memory and global memory are reduced, that is to say, both the amount of data access and the amount of computation are reduced, which lead to the superiority of the improved BPB algorithm. Furthermore, we can conclude that the improvement would be enhanced with the increment of the number of the particles in each box.

B. Numerical Experiments

In this section, the numerical experiments are executed and the experimental results of the implementations of the previous BPB algorithm and the improved BPB algorithm on GPU are compared and analyzed to demonstrate the superiority of the improved BPB algorithm based on the Newton's third law. The experiments are executed on a HP work station that consists of 8 cores (Intel Xeon E5506/2.13GHz quad-core) with 8GB memory and a single Tesla C1060. CentOS 5.4 is used as operating system. The compiler we used for the compilation of our GPU code is the NVIDIA CUDA compilation tools. The level of the quadtree is set to 7, thus there are 16384 boxes, and the particle distribution is uniform. The number of particles in each box ranges from 8 to 512.

The runtimes of the different implementations on CPU and GPU are recorded in Tab.3. The highest speedup of the improved BPB algorithm's implementation compared to the serial CPU implementation reaches 326. Fig.8 demonstrates the conclusion in the theoretical analysis that acceleration of the improved BPB algorithm is enhanced with the increment of the number of particles per box. However, the runtime of the improved BPB algorithm is not half or more reduced for the half reduced amount of computation and the reduced amount of data access. The reason is that the usage of the atomic function harmfully influenced the performance of the implementation on GPU. Nevertheless, the result of the experiments still definitely demonstrates the superiority of the improved BPB algorithm.

TABLE III. THE MACRO CHANGE OF THE AMOUNT OF COMPUTATION AND DATA ACCESS IN THE IMPROVED BPB ALGORITHM

Number of particles per box	**CPU(ms)**	**GPU(ms)**		**Speedup**	
		Previous BPB	***Improved BPB***	***Previous BPB***	***Improved BPB***
8	150.846	79.506	79.227	1.90	1.90
16	581.918	82.382	81.446	7.06	7.14
32	2291.87	90.248	88.130	25.40	26.01
64	9106.34	116.69	109.707	78.04	83.01
128	36426.1	234.54	202.768	155.31	179.64
256	145100	644.85	499.495	225.01	290.49
512	579568	2475.9	1774.183	234.08	326.67

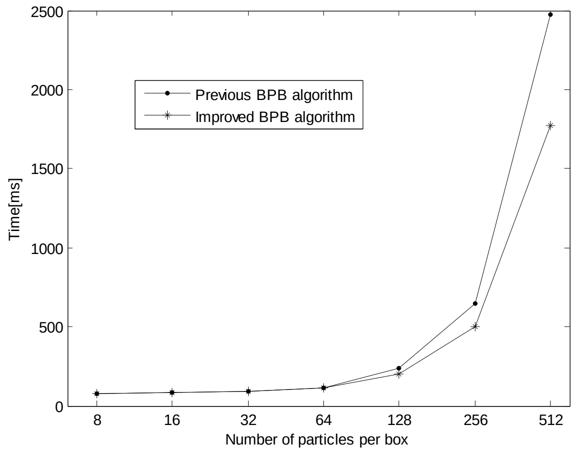

Figure 8. Comparison of the runtime of the previous BPB algorithm and the improved BPB algorithm on GPU

V. The Novel GPU Optimization Model of Transformation between Data Access and Computation.

The Newton's third law half reduces the amount of computation in the improved BPB algorithm, and the data accesses in these reduced computations are also deducted. However, the remaining computations need to undertake additional data access to deal with the counterforce, and this process is the transformation between data access and computation, which is a novel GPU optimization model. Through the theoretical analysis and numerical experiments, we have demonstrated that this GPU optimization model may optimize the performance of the algorithm mapping on GPU. Thus we can deep analyze an algorithm to find the possibility of the transformation between data access and computation to optimize its algorithm mapping on GPU. Because the data access always accompany with the computation, there are four situations of the transformation between data access and computation:

- If the computation is transformed to data access, the amount of computation is reduced and the amount of data access is increased.
- If the computation is transformed to data access, the amount of computation is reduced and the amount of data access is not changed or reduced.
- If the data access is transformed to computation, the amount of computation is increased and the amount of data access is reduced.
- If the data access is transformed to computation, the amount of computation is increased and the amount of data access is not changed or increased.

When the transformation belongs to the first three situations, the algorithm on GPU could be theoretically optimized. When the transformation belongs to the last situation, the performance would be harmfully influenced. Therefore, before the GPU optimization model of transformation between data access and computation is used to improve the algorithm mapping on GPU, the quantitative analysis of the amount of computation and data access should be conducted to judge which situation dose the transformation belong to.

VI. Conclusion

We have proposed a novel parallel algorithm for the near-field computation in N-body problem on GPU. This algorithm is based on Newton's third law and evolved from the BPB algorithm presented in our previous work. The theoretical analysis shows the improvement of the amount of the computation and data access in the improved BPB algorithm. The effect of this improvement is demonstrated by the numerical experiments. At last the GPU optimization model of transformation between data access and computation is proposed on the basis of the principle of the improvement in the improved BPB algorithm. Four situations of this model are discussed to help the researcher to determine whether the transformation in their algorithm can lead to the improvement of the implementation's performance on GPU.

In the future work, the far-field computation in N-body problem would be mapped on GPU, and the CUDA implementation would be transplanted to the multi-GPU computational platform.

Acknowledgment

This work is supported by National High-tech R&D Program of China under Grant 2009AA012201, Innovation Research program of Shanghai under Grant 12ZZ094, and Shanghai Academic Leading Discipline Project J50103.

References

[1] Jianchen Shan, Yongmei Lei and Jinshi Zhu, "The algorithm mapping of the near-field computation in N-body problem on GPU" in CCCA2011, Hong Kong, China, 2011, pp 326-330

[2] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," Journal of Computational Physics, vol. 73, pp. 325 - 348, 1987.

[3] "NVIDIA CUDA C Programming Guide Version 3.1.1" 7/21/2010 http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf

[4] Theodore Bially, "Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction." IEEE Transactions on Information Theory, IT-15(6):658-664, Nov 1969.

[5] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," vol. 324, pp. 446-449, 1986.

[6] Lars Nyland, Mark Harris, Jan Prins, "Fast N-Body Simulation with CUDA" GPU Gems, 2007, 3:677–695

[7] I. Lashuk, A. Chandramowlishwaran, H. Langston, T. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, "A massively parallel adaptive fast-multipole method on heterogeneous architectures," in SC '09, New York, NY, USA, 2009, pp. 58:1--58:12.

[8] T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri, and M. Taiji, "A novel multiple-walk parallel algorithm for the Barnes-Hut treecode on GPUs - towards cost effective, high performance N-body simulation," Computer Science, vol. 24, pp. 21-31, 2009.

[9] R. Beatson and L. Greengard, "A short course on fast multipole methods," Oxford University Press, 1997, pp. 1--37.

[10] J. Breitbart, "Case studies on gpu usage and data structure design" unpublished

[11] Felipe A. Cruz and L. A. Barba, "Characterization of the accuracy of the fast multipole method in particle simulations" Int. J. Numer. Meth. Engng., Vol. 79, No. 13. (2009), pp. 1577-1604.

[12] N. A. Gumerov and R. C. C. H. Duraiswami, "Fast multipole methods on graphics processors" J. Comput. Phys., vol. 227(2008), pp. 8290--8313.

[13] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, "42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence," in SC '09, New York, NY, USA, 2009, pp. 62:1--62:12.

[14] A. Kawai, T. Fukushige and J. Makino, "$7.0/Mflops Astrophysical N-Body Simulation with Treecode on GRAPE-5," SC Conference, vol. 0, p. 67, 1999.

[15] F. A. Cruz, M. G. Knepley and L. A. Barba, "PetFMM--A dynamically load-balancing parallel fast multipole library," CoRR, vol. abs/0905.2637, 2009.