World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:11, No:3, 2017

# Collision Detection Algorithm Based on Data Parallelism

Zhen Peng, Baifeng Wu

***Abstract*—**Modern computing technology enters the era of parallel computing with the trend of sustainable and scalable parallelism. Single Instruction Multiple Data (SIMD) is an important way to go along with the trend. It is able to gather more and more computing ability by increasing the number of processor cores without the need of modifying the program. Meanwhile, in the field of scientific computing and engineering design, many computation intensive applications are facing the challenge of increasingly large amount of data. Data parallel computing will be an important way to further improve the performance of these applications. In this paper, we take the accurate collision detection in building information modeling as an example. We demonstrate a model for constructing a data parallel algorithm. According to the model, a complex object is decomposed into the sets of simple objects; collision detection among complex objects is converted into those among simple objects. The resulting algorithm is a typical SIMD algorithm, and its advantages in parallelism and scalability is unparalleled in respect to the traditional algorithms.

***Keywords*—**Data parallelism, collision detection, single instruction multiple data, building information modeling, continuous scalability.

## I. Introduction

COMPUTING technology is moving into the era of the parallel computing. While it is difficult to further raise the frequency of processor chip, the number of processor cores can continue to be increased. How to make use of the computation ability of these processor cores more and more will be an obstacle for traditional programmers. SIMD technology is a solution for the problem. With the technology a program can directly utilize larger computing power by increasing the number of processor cores continuously without increasing the program complexity. Thus, it can be one of development directions of future computing technology.

For some of existing algorithms, due to the complex control structure and data dependency, it is hard to change into an effective SIMD algorithm.

In a traditional parallel program, a process is divided into many subprocesses delivered to multiple processors. But because of dependency usually exist among subprocesses, and the degree of parallelism is limited. What is more, the decomposition process is hardly sustained when we face larger amounts of data.

In this paper, based on the idea of data parallelism we propose a new processing model. The decomposition object in this model is the data rather than the processing procedure. Such decomposition is capable of leading to unlimited parallelism, and the increasing amount of data can be easily dealt with more processor cores.

Zhen Peng and Baifeng Wu are with the School of Computer Science, Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China (e-mail: pengz14@fudan.edu.cn, bfwu@fudan.edu.cn).

In this paper, we take the precise collision detection in BIM as an example, introduce the construction process of such an SIMD algorithm and analyze its performance and scalability.

## II. Related works

The Building Information Modeling (BIM) is a technology of designing the building with the digital model. In the design phase, the precise collision detection can help us to find design error in time and provide hints for optimizing or modifying design. Such a technique can usually remarkably reduce design cost and improve design efficiency.

Existing parallel algorithms of the collision detection are classified into three categories: the parallel hierarchical bounding box algorithm, the parallel space partition algorithm and the parallel image space algorithm.

An approach for fast bounding volume hierarchy (BVH) construction based on tree structure on GPU is proposed in [1]. A method that partitions scene data, and decomposes and updates BVH concurrently is introduced in [2]. The SIMD-DOP method based on the discrete orientation polytope (k-DOP) and optimized by SIMD instructions is introduced in [3]. A new k-DOP method improved by genetic algorithms is proposed in [4]. Building balanced BVH tree with the divide and conquer strategy improves the efficiency of the parallel traversal of the bounding box tree in [5], [6]. A clustering scheme and a collision-packet traversal to perform efficient collision queries is presented in [7]. A method with the dynamic execution model and its associated experimental runtime system is presented in [8]. The OBB and AABBBVH tree with some other technologies has been used in [9]-[14]. The p-partition front method in [15] partitions and distributes the workload of BVH traversal among multiple processing cores in many-core platform without the need for dynamic balancing.

The octree node pair is used to determine the level of parallelization combining with load balancing algorithm in [16]. An octree grid and octree subdivision improved by a two-stage scheme is used to reduce the number of tests in the broad phase in [17]. A technique for dynamic construction of octree is presented in [18]. A G-Octree structure dividing the scene space owning the advantages of both the uniform grid and octree is proposed in [19]. A new octree-based proxy is proposed which divides the scene mesh in [20]. The approach is proposed which divides space into a sparse grid of regular axis-aligned voxels then objects are sent to all voxels they intersect in [21].

It introduces an approach based on image–space interference tests in [22]. It presents a novel and fast algorithm with the A-buffer, in which the depth values of texel are used to compute colliding sets in [23]. The algorithm on the basis of RECODE and RAPID algorithm is proposed in [24].

There are some problems in these algorithms. First, accurate locations and shapes of the collision cannot be obtained. Second, the parallelism of the algorithm is the parallelism of the control flow rather than data flow, which limits the degree of parallelism. Third, when data amount rises, it is difficult for user to get more computation power by increasing the number of processor cores. In these existing algorithms there are severe dependency among data, much of the computation is spent on the maintenance of dependency. Such algorithms are usually complex in program structure and low in scalability.

## III. The Summary of the Algorithm

The three steps of the algorithm are the following. In the first step, the models for collision detection are partitioned into sets of cubes respectively. Then in the second step, for each pair of cubes belonging to different models, a collision detection is played. Since the shapes of these cubes are simple and identical, such a collision detection is extremely easy. Finally, in the third step, the collision results of the detection process in the second step are merged to generate the collision results of models. As long as the cubes from these models are small enough, these collided cubes can approximate the final collision results. Thus it constitutes a data parallel computing process which includes the parallel partition of the models into cubes, the parallel collision detection of all pairs of cubes belonging into different models and the parallel reduction of the detection result. The process formed in three steps is the typical data parallel computation process which can be undertaken by thousands of independent parallel threads which means the greater computing power can be easily obtained via increasing the number of processor cores without changing programs; that is so-called continuous scalability.

### *A. The Parallel Partition*

Different models are divided into their respective collections of cubes on the different processor cores. There are no correlations and also data exchanges among processor cores. The precision of the computation is defined as the length of cube's edge. Cubes have same length but different positions. Cubes in the collection constitute a new component which approximately expresses the model. The higher the precision is, the more approximate the new component and the model are. So the partition algorithm, Partition (The Model, The Precision), which generates the collection of cubes, $C_m$, is defined. On the assumption, $P_u$ denotes the processor core and $u$ is an unique serial.

**Algorithm 1**. Partition

```
$m_1, m_2, \ldots, m_N$ [input] total models
$P_u$ [input] the processor core $u$
$a$ [input] The Precision
$C_m$ [output] the collection of cubes of $m$
1: **For all** $u \in [1, N]$
2: $m$ is divided into many cubes which construct $C_m$ by uniform grid.
3: **End**
```

The method of partitioning the model is uniform grid in Algorithm 1. If the time of generating each cube is $t_1$, it is obvious that the time of Algorithm 1 is proportional to $|C_m|$, that is $|C_m|\ t_1$. Due to the arbitrariness of $m$, $|C_m|$ also means average. So, the total time that different cores execute Algorithm 1 concurrently is also $|C_m|\ t_1$.

After the partitioning process, each pair of cubes of different models forms a two tuple. Assuming, we have $N(N > 2)$ models. For model $m(0<=m<=N)$, its corresponding number of cubes is $|C_m|$. For each pair of models, $m_i$ and $m_j$, $C_{m_i}$ and $C_{m_j}$ can be generated. There are $|C_{m_i}||C_{m_j}|$ number of two tuples can be generated. So, there are total $\sum_{i=1}^{N}\sum_{j=i+1}^{N}|C_{m_i}||C_{m_j}|$ number of two tuples.

Now, it is time to generate total two tuples. On the PRAM, where there are infinite number of processor cores, the algorithm Pre-Generation and Generation which generating total two tuples are defined.

**Algorithm 2**. Pre-Generation

```
$m_1, m_2, \ldots, m_N$ [input] total models
$C_{m_1}, C_{m_2}, \ldots, C_{m_N}$ [input] collections of models
$P_u$ [input] the processor core $u$
$Pair$ [output] the collection of the number of two-tuples
1:**Forall** $u \in [1, N(N-1)/2]$
2: Obtaining models, $m_i$ and $m_j$, and also..and $C_{m_j}$ through $u$
3: $Pair_u = |C_{m_i}||C_{m_j}|$
4:**End**
```

On the PRAM, $N(N-1)/2$ processor cores execute the Algorithm 2 concurrently and output the collection $Pair$, where the element denotes the quantity of two-tuples that can be produced from cubes of two models. The second and third step in Algorithm 2 will be done in constant time. So, the time of Algorithm 2 is constant. The collection $Prefix\_Pair$ where the element $Prefix\_Pair_u = \sum_{i=1}^{u} Pair_i$, and total count of two-tuples, $TotalPairCount = Prefix\_Pair_N$, are defined. The $Prefix\_Pair$ collection will be evaluated by the parallel prefix summation and its time is $log(N(N-1)/2)$ [25].

**Algorithm 3**. Generation

```
$m_1, m_2, \ldots, m_N$ [input] total models
```

```
$C_{m_1}, C_{m_2}, \ldots, C_{m_N}$ [input] collections of models
$P_u$ [input] the processor core $u$
$Pair$ [input] the collection of the number of two-tuples
$Prefix\_Pair$ [input]the prefix summation of $Pair$
$D$ [output] the collection of two-tuples
1: **Forall** $u \in [1, N(N-1)/2]$
2:   **For** $j = 1$ **to** $Pair_u$ **do**
3:     $t = j + Prefix\_Pair_{u-1}$
4:     Obtaining two collections, $C_{m_x}$ and $C_{m_y}$ ,from $Pair_u$
5:     Obtaining the jth two tuple, $Cube_{j1}$ and $Cube_{j2}$ ,of two tuples that $C_{m_x}$ and $C_{m_y}$ generate
6:     $D_t = \{u, Cube_{j1}, Cube_{j2}\}$
7:   **End**
8: **End**
```

At least $TotalPairCount$ processor cores are required to practice the collision detection of two-tuples as fast as possible. Therefore, when it is generating the collection of two-tuples, Algorithm 3 also sets corresponding two-tuple for each processor core in the third and sixth step. In Algorithm 3, steps from 3 to 6 will be done in constant time. So, its execution time is proportional to the average size of set $Pair_u$ , that is $|C_m|^2$ .

### *B. The Parallel Collision Detection*

Now, the collision between two cubes in each two-tuple in the collection $D$ from the Algorithm 3 will be detected. On the assumption, there is the algorithm which detects the collision between two cubes whose time is $t_2$ . Hence, the algorithm Detection detecting the collision concurrently is defined.

**Algorithm4**. Detection

```
$P_u$ [input] the processor core $u$
$D$ [input] the collection of two-tuples
$R$ [output] the collection of results of the detection
1: **Forall** $u \in [1, TotalPairCount]$
2:   $\{u, Cube_1, Cube_2\} = D_u$
3:   Detecting the collision between $Cube_1$ and $Cube_2$ ;the result=1 if they collided, or 0
4:   $R_u = \{u, Cube_1, Cube_2\} \cup$ the result
5: **End**
```

In PRAM, $TotalPairCount$ number of processor cores run the Algorithm 4 concurrently and gain the consequence $R$ . In Algorithm 4, steps from 2 to 4 can be done in constant time and the step 3 costs the majority of time. So, the time of Algorithm 4 is $t_2$ .

### *C. The Parallel Reduction*

At last, it is time to reduce $R$ from the Algorithm 4 and extract all two-tuples where two cubes are collided. If positions of these two-tuples in the final results can be calculated in advance, the procedure of extraction can be done concurrently. In Algorithm 4, $R$ can be considered as the collection of 1 and 0, where $R$ is 1 indicates whose two cubes are collided, otherwise $R$ is 0. The collection $Prefix\_R$ is defined where the element $Prefix\_R_u = \sum_{i=1}^{u} R_i$ which denotes the position in final collection when $R_u$ is 1. The $Prefix\_R$ collection can be evaluated by the parallel prefix summation algorithm [25] and its time is $log(TotalPairCount)$ . So, the algorithm Reduction extracting these two-tuples concurrently is defined.

**Algorithm5**. Reduction

```
$P_u$ [input] the processor core $u$
$R$ [input] the collection of results of the detection
$Prefix\_R$ [input] the prefix summation of $R$
$O$ [output] the collection of all collided two-tuples
1: **Forall** $u \in [1, TotalPairCount]$
2:   **If** $R_u = 1$ **Then**
3:     $t = Prefix\_R_u$
4:     $\{u, Cube_1, Cube_2\} = R_u$
5:     $O_t = \{u, Cube_1, Cube_2\}$
6:   **End**
7: **End**
```

In PRAM, $TotalPairCount$ number of processor cores run Algorithm 5 concurrently and gain the consequence $O$ . $O$ is the collection of all collided two-tuples. It is able to gain details of the collision from two-tuples in $O$ . In Algorithm 5, steps from 2 to 5 have be done in constant time. Hence, the time of Algorithm 5 is constant.

### *D. The Time of the Algorithm*

Algorithms 1 – 5 and discussion of the construct of the new algorithm are proposed in the paper. The proposed algorithm and its time are discussed on PRAM with infinite number of processor cores. Parts and their time of the new algorithm are listed in Table I. So, the total time $T_1$ of the new algorithm is the sum of the time of each part.

TABLE I
THE TIME OF THE ALGORITHM

| Parts | Time |
|---|---|
| **Algorithm 1**. Partition | $\lvert C_m \rvert t_1$ |
| **Algorithm 2**. Pre-Generation | *Constant* |
| Obtaining $Prefix\_Pair$ | $\log(N(N-1)/2)$ |
| **Algorithm 3**. Generation | $\lvert C_m \rvert^2$ |
| **Algorithm 4**. Detection | $t_2$ |
| Obtaining $Prefix\_R$ | $\log(TotalPairCount)$ |
| **Algorithm 5**. Reduction | *Constant* |

$$T_1 = |C_m| t_1 + log(N(N-1)/2) + |C_m|^2 + t_2 + log(TotalPairCount) + \text{Constants} \tag{1}$$

$$TotalPairCount = Prefix\_Pair_N = \sum_{i=1}^{N} Pair_i = N|C_m|^2 \tag{2}$$

Combining (1) and (2):

$$T_1 = 3log(N) + 2log(|C_m|) + |C_m| t_1 + |C_m|^2 + t_2 + \text{Constants} \quad (3)$$

## IV. The Summary of the Algorithm in Reality

In the discussion of algorithms, each processor core only handles a piece of data and the number of processor cores is equal to pieces of data. On the assumption there are $Q$ number of processor cores, each processor core should now handle multiple pieces of data. Thus, it is necessary to make modifications on formulas and the time of the new algorithm. Take the Algorithm 1 for example, it is obtained after modification. Algorithms 2 to 5 are modified in the same manner and not described in detail any further. Furthermore, $Prefix\_Pair$ and $Prefix\_R$ are calculated by the parallel prefix summation algorithm [25]. Because of the Brent theorem [26], its time is $log(M + M/Q)$ where $M$ is the amount of data under such circumstances. The time of algorithms after modification is listed in Table II. So, the total time $T_2$ of the new algorithm is still the sum of the time of each part.

$$\begin{aligned} T_2 = {} & (N/Q)|C_m| t_1 + (N(N-1)/(2Q)) + \\ & log(N(N-1)/2 + N(N-1)/(2Q)) + \\ & (N(N-1)/(2Q))|C_m|^2 + (TotalPairCount/Q)t_2 + \\ & log(TotalPairCount + TotalPairCount/Q) + \\ & TotalPairCount/Q \end{aligned} \quad (4)$$

Combining (2) and (4):

$$K_1 = \left(1 + |C_m|^2\right)/2 \quad (4)$$

$$K_2 = |C_m|^2 + |C_m|^2 t_2 + |C_m| t_1 \quad (5)$$

$$K_3 = log|C_m|^2 \quad (6)$$

$$T_2 = \left(N^2/Q\right)K_1 + (N/Q)K_2 + log\left((Q+1)^2/\left(2Q^2\right)\right) + 3logN + K_3 \quad (7)$$

It is obvious that (5)-(7) are constants. In (8), $T_2$ consists of the parallel part which is $\left(N^2/Q\right)K_1 + (N/Q)K_2 + log\left((Q+1)^2/\left(2Q^2\right)\right)$ and the serial part which is $3logN + K_3$. When the number of processor cores, $Q$, approaches infinity, the parallel part can be accelerated linearly and finally becomes zero. The serial part is just the logarithm function of $N$.

**Algorithm 1'**. Partition

```
$m_1, m_2, \ldots, m_N$ [input] total models
$P_u$ [input] the processor core $u$
$a$ [input] The Precision
$C_m$ [output] the collection of cubes of $m$
1: For all $v \in [1, Q]$
2:   For j=0 to $(N/Q+1)$ do
3:     If $(u = j*Q+v) <= N$ Then,
4:       $m$ is divided into many cubes which construct $C_m$ by uniform grid method.
5:     End
6:   End
7: End
```

## V. Design and Analysis of the Experiment

### A. The Target and Design of the Experiment

In what follows, we use $N$, $Q$ and $T$ to respectively denote the amount of data, the number of processor cores and the execution time.

The target of the experiment is to verify the relationship that is the increase of $N$ can be handled by the increase of $Q$ with the execution time $T$ unchanging. $N$ can be adjusted flexibly.

TABLE II
THE TIME OF THE ALGORITHM AFTER MODIFICATION

| Parts | Time |
|---|---|
| **Algorithm 1'**. Partition | $(N/Q)\|C_m\|t_1$ |
| **Algorithm 2'**. Pre-Generation | $(N(N-1)/(2Q))$ |
| The calculation of $Prefix\_Pair$ | $log(N(N-1)/2 + N(N-1)/(2Q))$ |
| **Algorithm 3'**. Generation | $(N(N-1)/(2Q))\|C_m\|^2$ |
| **Algorithm 4'**. Detection | $(TotalPairCount/Q)t_2$ |
| The calculation of $Prefix\_R$ | $log(TotalPairCount + TotalPairCount/Q)$ |
| **Algorithm 5'**. Reduction | $TotalPairCount/Q$ |

TABLE III
GPU

| Name | $Q$ | GPU clock frequency (MHz) |
|---|---|---|
| Tesla C2075 | 448 | 1147 |
| Tesla K20m | 2496 | 706 |
| KeplerGK20A | 192 | 852 |

We choose three GPUs with a different number of processor cores. Three key parameters of the GPU are shown in Table III.

The algorithm is implemented on the CUDA platform. The implementation of the algorithm on different GPUs is identical, including the partition granularity.

### B. Results and Analysis

On different GPUs, with the addition of $N$, the corresponding algorithm execution time $T$ is shown as Tables IV-VI. In Tables IV-VI, the execution time has been divided by the respective GPU clock frequency. After multiple polynomial fittings between $N$ and $T$, we find the quadratic fitting can lead to good fitting results. So the experimental equations on different GPUs between $T$ and $N$ are determined by the quadratic fitting and shown in Table VII.

TABLE IV
THE EXECUTION TIME ON TESLA C2075

| $N$ | $T$ (MS) | $N$ | $T$ (ms) |
|---|---|---|---|
| 448 | 2.115142 | 3136 | 47.203162 |
| 896 | 5.737569 | 3584 | 59.519289 |
| 1344 | 11.052878 | 4032 | 74.368611 |
| 1792 | 17.526462 | 4480 | 90.558515 |
| 2240 | 26.039730 | 4928 | 107.930014 |
| 2688 | 35.292662 | 5376 | 127.180430 |

TABLE V
THE EXECUTION TIME ON TESLA K20M

| $N$ | $T$ (MS) | $N$ | $T$ (ms) |
|---|---|---|---|
| 312 | 1.163411 | 3432 | 45.284932 |
| 624 | 2.846401 | 3744 | 52.853101 |
| 936 | 5.174316 | 4056 | 61.248362 |
| 1248 | 8.110800 | 4368 | 70.055490 |
| 1560 | 11.691257 | 4680 | 79.540202 |
| 1872 | 15.882692 | 4992 | 89.618648 |
| 2184 | 20.590834 | 5304 | 100.404845 |
| 2496 | 25.807761 | 5616 | 111.825580 |
| 2808 | 31.716484 | 5928 | 123.614233 |
| 3120 | 38.157489 | | |

TABLE VI
THE EXECUTION TIME ON KEPLER GK20A

| $N$ | $T$ (MS) | $N$ | $T$ (ms) |
|---|---|---|---|
| 192 | 2.675388 | 1920 | 109.851498 |
| 384 | 6.337382 | 2112 | 158.259225 |
| 576 | 12.864418 | 2304 | 215.330858 |
| 768 | 23.818332 | 2496 | 183.237438 |
| 960 | 36.066012 | 2688 | 211.913586 |
| 1152 | 41.133188 | 2880 | 242.920610 |
| 1344 | 56.061404 | 3072 | 275.532809 |
| 1536 | 72.166813 | 3264 | 312.487933 |
| 1728 | 89.868976 | 3456 | 377.754658 |

Now, we keep $T$ unchanged and consider the relationship between $Q$ and $N$. The corresponding $N$ in Table VIII, IX, X can be got from the same $T$ in (9), (10) and (11) by gradually increasing $T$.

Based on Tables VIII, IX, X, on the same $T$, we get the relationship between $Q$ and $N$, as in Fig. 1.

TABLE VII
THE EXPERIMENTAL EQUATIONS ON GPUS

| Name | The calculation equation for the time $T$ |
|---|---|
| Tesla C2075 | $(3.887136E-06)N^2+0.002715N+0.211335$ (8) |
| Tesla K20m | $(3.06991E-06)N^2+0.002634N+0.070202$ (9) |
| KeplerGK20A | $(2.71559E-05)N^2+0.010828N-2.138942$ (10) |

TABLE VIII
THE CORRESPONDING $T$ FOR THE $N$ ON TESLA C2075

| $T$ (ms) | $N$ | $T$ (MS) | $N$ |
|---|---|---|---|
| 10 | 1276 | 110 | 4977 |
| 20 | 1934 | 120 | 5213 |
| 30 | 2441 | 130 | 5440 |
| 40 | 2869 | 140 | 5658 |
| 50 | 3247 | 150 | 5868 |
| 60 | 3588 | 160 | 6072 |
| 70 | 3902 | 170 | 6269 |
| 80 | 4195 | 180 | 6461 |
| 90 | 4470 | 190 | 6647 |
| 100 | 4729 | 200 | 6828 |

TABLE IX
THE CORRESPONDING $T$ FOR THE $N$ ON TESLA K20M

| $T$ (ms) | $N$ | $T$ (MS) | $N$ |
|---|---|---|---|
| 10 | 1420 | 110 | 5570 |
| 20 | 2155 | 120 | 5836 |
| 30 | 2723 | 130 | 6091 |
| 40 | 3203 | 140 | 6336 |
| 50 | 3627 | 150 | 6573 |
| 60 | 4010 | 160 | 6802 |
| 70 | 4363 | 170 | 7023 |
| 80 | 4692 | 180 | 7239 |
| 90 | 5000 | 190 | 7448 |
| 100 | 5293 | 200 | 7652 |

TABLE X
THE CORRESPONDING $T$ FOR THE $N$ ON KEPLER GK20A

| $T$ (ms) | $N$ | $T$ (MS) | $N$ |
|---|---|---|---|
| 10 | 498 | 110 | 1842 |
| 20 | 725 | 120 | 1931 |
| 30 | 907 | 130 | 2016 |
| 40 | 1062 | 140 | 2097 |
| 50 | 1201 | 150 | 2176 |
| 60 | 1326 | 160 | 2252 |
| 70 | 1443 | 170 | 2326 |
| 80 | 1551 | 180 | 2398 |
| 90 | 1653 | 190 | 2468 |
| 100 | 1750 | 200 | 2536 |

In Fig. 1, the vertical axis is $Q$ and the horizontal axis is $N$. Each polyline represents the relationship between $N$ and $Q$ on the different time $T$. The polyline from the left to the right indicates the time gradually increases with the same interval.

The conclusion from Fig. 1:

1. For the specific time $T$, the increase of $N$ can be dealt with in the increase of $Q$, which maintains the execution time unchanged. For example, in the polyline ($T$ =30ms in Fig. 1), when $N$ =2500, it gets $Q$ =1000; when $N$ =2750, it gets $Q$ =2000. The time $T$ stays the same when this happens. Thus, the target of the experiment can be verified.

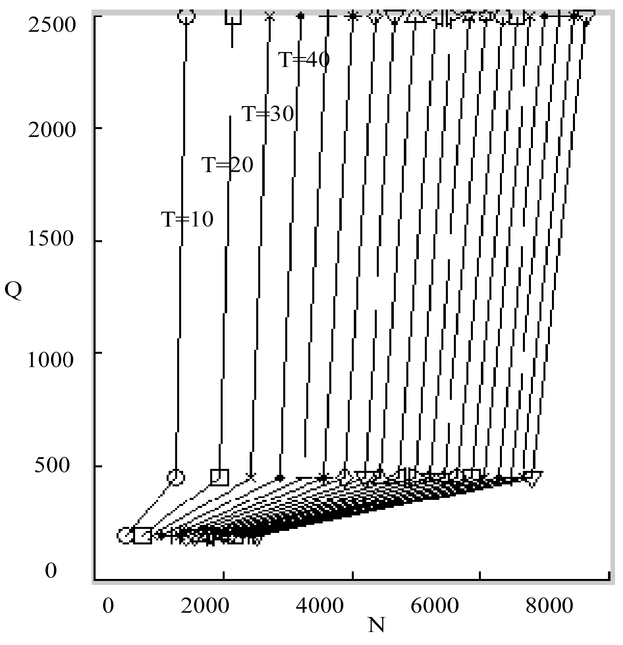

Fig. 1 The relationship between $Q$ and $N$ on the same time $T$

2. The average slopes of polylines decrease gradually from left to right. On the leftmost polyline, when $N$ increases by 1000, $Q$ increases by 2300 to keep $T$ unchanged. But, on the rightmost polyline, when $N$ increases by 5000, $Q$ also increases by 2300 to keep $T$ unchanged. Thus, from the left to the right in Fig. 1, it indicates that when $N$ increases, the same increase of $Q$ can cope with the larger increase of $N$. The continuous scalability of the algorithm is verified.
3. From left to right, polylines become more and more dense. It shows that the growth of $N$ can lead to the larger growth rate of $T$. Theoretically, the growth rate of $T$ is the first order partial derivative of $N$ in (8). The one or two order partial derivative of $N$ in (8) can be obtained.

$$\frac{\partial T_2}{\partial N} = \frac{2K_1}{Q} N + 3\frac{1}{N} + \frac{K_2}{Q} \tag{11}$$

$$\frac{\partial^2 T_2}{\partial N^2} = \frac{2K_1}{Q} - 3\frac{1}{N} \tag{12}$$

when $N > \sqrt{1.5Q / K_1}$ is established, the value of (13) becomes greater than zero, and the value of (12) is increasing. So, a $N'$ can be found, when $N > N'$ is established, the growth rate of $T$ becomes larger. Thus, the experiment results verify the point.

## VI. Conclusion

In BIM and other fields, the increasing amount of data gradually becomes the obstacle of computing technique. How to solve this problem by increasing processor core is a concern. In this paper, we take the precise collision detection in BIM as an example, and change a traditional serial execution based algorithm to a parallel algorithm with an unlimited extension in parallelism.

In the era of the multi-core and many-core techniques, the combination of data parallelism and the SIMD technique can show prominent advantages in controlling the program complexity and getting sustainable scalability, and will be one of the development trends for these applications.

## References

[1] X. Yang, T. M. Wang and D. Q. Xu, "Fast BVH construction on GPU," *Journal of Zhejiang University,* vol. 46, pp. 84-89, 2012.
[2] D. Peng, T. Min and A. T. Ruofeng, "Parallel Collision Detection on Multi-core Platform," *Jisuanji Fuzhu Sheji Yu Tuxingxue Xuebao/Journal of Computer-Aided Design and Computer Graphics,* vol. 23, pp. 833-838, 2011.
[3] M. Tang, M. Dinesh, R. F. Tong, "Parallel Collision Detection Between Deformable Objects Using SIMD Instructions," *Chinese Journal of Computers,* vol. 32, pp. 2042-2051, 2009.
[4] W. Zhao and L. Li, "A New K-DOPs Collision Detection Algorithms Improved by GA," in *International Conference on Wireless Communications and Applications*, 2011, pp. 58-68.
[5] W. Zhao, C. Chen and L. Li, "The Collision Detection Algorithm Based on the MapReduce of Cloud Computing," *System simulation technology and its application Changchun,* 2010, pp. 96-100.
[6] W. Zhao, C. Chen and L. Li, "Parallel Collision Detection Algorithm Based on OBB Tree and MapReduce," in *Entertainment for Education. Digital Techniques and Systems: 5th International Conference on E-learning and Games, Edutainment 2010, Changchun, China, August 16-18, 2010. Proceedings*, X. Zhang, S. Zhong, Z. Pan, K. Wong, and R. Yun, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 610-620.
[7] J. Pan and D. Manocha, "GPU-Based Parallel Collision Detection for Real-Time Motion Planning," *Springer Tracts in Advanced Robotics,* vol. 68, pp. 211-228, 2010.
[8] M. Anderson, M. Brodowicz, L. Dalessandro, J. DeBuhr, and T. Sterling, "A Dynamic Execution Model Applied to Distributed Collision Detection," in Supercomputing: 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings, J. M. Kunkel, T. Ludwig and H. W. Meuer, Eds. Cham: Springer International Publishing, 2014, pp. 470-477.
[9] P. Cai, C. Indhumathi, Y. Cai, J. Zheng, Y. Gong, S. L. Teng, and W. Peng, "Collision Detection Using Axis Aligned Bounding Boxes" *Springer Singapore,* 2014, pp. 1-14.
[10] R. Erbes, A. Mantel, E. Schömer, and N. Wolpert, "Parallel Collision Queries on the GPU," *Lecture Notes in Computer Science,* vol. 7686, pp. 84-95, 2013.
[11] Y. J., P. J. and B. M., "GPU-based collision detection for sampling-based motion planning," in *Ubiquitous Robots and Ambient Intelligence (URAI), 2013 10th International Conference on*, 2013, pp. 215-218.
[12] Y. Zou, X. Zhou, G. Ding, Y. He, M. Jia, "A GPGPU-Based Collision Detection Algorithm," in *Image and Graphics, 2009. ICIG '09. Fifth International Conference on,* 2009, pp. 938-942.
[13] X. Zhang, YJ. Kim, "Interactive Collision Detection for Deformable Models Using Streaming AABBs," *IEEE Transactions on Visualization and Computer Graphics,* vol. 13, pp. 318-329, 2007-01-01 2007.
[14] HA. Aly, QAE. Elawady, "A new narrow phase collision detection algorithm using height projection," in *Education and Research Conference (EDERC), 2010 4th European*, 2010, pp. 111-115.
[15] X. Zhang, YJ. Kim, "Scalable Collision Detection Using p-Partition Fronts on Many-Core Processors," *IEEE Transactions on Visualization and Computer Graphics,* vol. 20, pp. 447-456, 2014-01-01 2014.
[16] X. Liu and L. Cao, "Parallel Octree Collision Detection Based on MPI," *Journal of Computer-Aided Design & Computer Graphics,* pp. 184-187+192, 2007-02-28 2007.
[17] T. H. Wong, G. Leach and F. Zambetta, "An adaptive octree grid for GPU-based collision detection of deformable objects," *The Visual Computer,* vol. 30, pp. 729-738, 2014-01-01 2014.
[18] F. Tsuda, R. Nakamura, "A Technique for Collision Detection and 3D Interaction Based on Parallel GPU and CPU Processing," in *Games and*

*Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*, 2011, pp. 36-42.

[19] R. Xu, L. Kang, H. Tian., "A G-Octree Based Fast Collision Detection for Large-Scale Particle Systems," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, 2012, pp. 269-273.

[20] W. Fan, B. Wang, J. Paul, and J. Sun, "An octree-based proxy for collision detection in large-scale particle systems," *Science China Information Sciences,* vol. 56, pp. 1-10, 2013-01-01 2013.

[21] O. S. Lawlor and L. V. Kalée, "A voxel-based parallel collision detection algorithm,", 2002, pp. 285-293.

[22] G. Baciu, WSK. Wong, "Image-based techniques in a hybrid collision detector," *IEEE Transactions on Visualization and Computer Graphics,* vol. 9, pp. 254-271, 2003-01-01 2003.

[23] H. Jang and J. Han, "Fast collision detection using the A-buffer," *The Visual Computer,* vol. 24, pp. 659-667, 2008-01-01 2008.

[24] L. Wang, Y Shi, R. Li, "An image-based collision detection optimization algorithm," in *Signal and Information Processing (ChinaSIP), 2015 IEEE China Summit and International Conference on,* 2015, pp. 220-224.

[25] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, vol. 27, pp. 831-838, 1980-01-01 1980.

[26] R. P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," *Journal of the ACM,* vol. 21, pp. 201-206, 1974-01-01 1974.