Parallel Continuous Collision Detection for High-Performance GPU Cluster

Peng Du*
Nanyang Technological University, Singapore
Hangzhou Dianzi University, China

Elvis S. Liu†
Nanyang Technological University, Singapore

Toyotaro Suzumura‡
IBM Thomas J. Watson Research Center, USA

Abstract

Continuous collision detection (CCD) is a process to interpolate the trajectory of polygons and detect collisions between successive time steps. However, primitive-level CCD is a very time-consuming process especially for a large number of moving polygons. Over the years, a number of approaches have been proposed to improve the computational efficiency of CCD by culling out the non-colliding primitives before exact overlap tests. These approaches have two fundamental disadvantages. First, they are mainly designed for self- and pairwise CCD and thus the performance gain would be limited when they are applied to large-scale scenes that contain thousands of moving polygons. Second, they are designed as sequential processes appropriate for execution on a single processor. Therefore, deploying them on high-performance parallel computing systems would not increase their computational efficiency.

In this paper, we present a parallel CCD algorithm, which aims to accelerate N-body CCD culling by distributing the load across a high-performance GPU cluster. Our implementation integrates frameworks such as Message Passing Interface and CUDA, which is particularly suitable for large-scale distributed simulations. Experimental results, based on simulations conducted on a supercomputer, demonstrate that our approach is more computationally efficient than existing sequential CCD approaches.

Keywords: Continuous Collision Detection; Parallel Collision Detection; GPU Cluster; Supercomputer; Load Balancing; N-body

Concepts: •**Computing methodologies** → **Collision detection;**

1 Introduction

A virtual environment (VE) is a three-dimensional (3D) computer simulation that provides sensory information in such a way that users can visualize, explore, and interact with virtual entities in the environment. In recent years, large-scale VE simulations, such as military simulations [Neyland 1997; Perumalla et al. 2002] and distributed virtual environments [Steed and Oliveira 2010], have been developed to support hundreds of thousands, if not millions, of entities. This enables the application of parallel and distributed simulations (PDS) [Fujimoto 2000], which facilitates workload sharing by distributing the objects among a cluster of workstations.

One of the major challenges of large-scale PDS is providing scalable 3D collision detection service, which is a technique designed to detect collision of geometric polygons. Traditional discrete collision detection (DCD) is a process to detect collisions at every simulation time step, while continuous collision detection (CCD) is a technique to detect not only the discrete collisions, but also the missing collisions between discrete time steps. CCD has been widely used in physical simulations [Tang et al. 2012], motion planning [Lee et al. 2014], and virtual assembly [Du et al. 2015] to avoid missing collisions. However, this technique is usually more computationally intensive than DCD and involves decompositions of vertex/face (VF) and edge/edge (EE) intersection tests at the primitive-level.

Over the years, a number of approaches have been proposed, which sought to improve the computational efficiency of CCD by culling out the non-colliding primitives before exact overlap tests [Tang et al. 2008; Tang et al. 2010b; Tang et al. 2011a; Wong et al. 2013]. These approaches have two major disadvantages when applying on large-scale PDS. First, they are mainly designed for self- and pairwise collisions. Therefore, the performance gain would be limited when they are applied to large-scale VE simulations that contain thousands of objects (i.e. N-body scenes). Second, they are designed as sequential processes appropriate for execution on a single processor, which is unsuitable for PDS since their workload cannot be shared among the cluster of workstations. Although parallel DCD for distributed-memory systems has been discussed in the literature [Lawlor and Kalée 2002], the need of high-performance parallel CCD is still largely unmet.

Main contributions: To improve the performance of N-body CCD and facilitate workload sharing in PDS, this paper presents a parallel N-body CCD acceleration approach. This approach is designed especially for high-performance GPU clusters. It first employs a spatial decomposition method to distribute the entities across a GPU cluster (hereafter referred to as nodes), and then uses a number of efficient GPU-based culling methods to get the colliding entity pairs. We have implemented our approach with Message Passing Interface (MPI) and CUDA and performed experimental evaluation on the supercomputer TSUBAME [GSIC 2016], which is ranked No. 31 on the TOP500 list [TOP500 2016] (as of October 2016). Experimental results demonstrate that our approach is more computationally efficient than existing sequential CCD approaches when applying on a high-performance GPU cluster.

The rest of this paper is organized as follows. Section 2 briefly reviews the related work of collision detection. Section 3 describes an GPU-based N-body CCD approach. Section 4 presents a parallel CCD algorithm for distributed-memory systems. Section 5 evaluates the performance of the proposed algorithm. Finally, Section 6 concludes this paper and briefly describes the future work.

*e-mail:dupengtomas@gmail.com
†Corresponding author, e-mail:elvisliu@ntu.edu.sg
‡e-mail:tsuzumura@us.ibm.com

I3D '17, March 04-05, 2017, San Francisco, CA, USA
ISBN: 978-1-4503-4886-7/17/03...$15.00
DOI: http://dx.doi.org/10.1145/3023368.3023384

2 Related Work

In this section, we briefly review the prior work of collision detection, particularly CCD, N-body DCD, and parallel collision detection approaches.

2.1 Continuous Collision Detection

Over the years, many efficient culling algorithms have been proposed to improve the performance of CCD, which can be divided into high-level culling and low-level culling. High-level culling indicates the culling method works on polygons, while the low-level culling works on primitives.

High-level culling: A number of culling methods aim to bound the polygon by simple-shaped bounding volumes (BVs), in order to reduce the complexity of CCD. Representative examples include spheres [Bradshaw and O'Sullivan 2004], axis-aligned bounding boxes (AABBs) [Bergen 1998], discrete oriented polytopes (k-DOPs) [Klosowski et al. 1998] and oriented bounding boxes (OBB). And then bounding volume hierarchy (BVH) is built for the scene filled with thousands of virtual entities [Tang et al. 2008] to speedup collision detection.

Low-level culling: A class of culling methods is to remove redundant primitive pairs (VF or EE) and in turn reduces the primitive tests for the potentially colliding set (PCS). Representative-triangles [Curtis et al. 2008] and orphan sets [Tang et al. 2008] are two influential methods for culling redundant primitives. The former assigns each primitive to a unique triangle to guarantee no redundant collision pairs would be processed. The latter builds an orphan sets for adjacent collision pairs since they are difficult to cull out. Furthermore, many efficient culling methods for the PCS are based on separating axis. Representative examples include continuous separating axis [Tang et al. 2011b], deforming non-penetration filter [Tang et al. 2010a], and non-collinear filter [Du et al. 2012].

2.2 N-body Collision Detection

Some existing approaches use simple-shaped BVs for fast N-body culling, but they are all designed for DCD process. Most notably, a sorting-based technique known as sweep and prune (S&P) [Cohen et al. 1995] has been proven to be very efficient in N-body culling, especially in an environment of high spatial coherence. Liu et al. extended this method by using GPU for performance acceleration. They combine a space decomposition method and S&P for N-body culling [Liu et al. 2010]. In our previous work [Du and Liu 2016], we proposed an N-body CCD approach, which is designed especially for rotational rigid bodies. These approaches, however, cannot be directly deployed on a computer cluster since they only work on shared-memory systems, which are supposed to be executed on a single computer. As the number of entities grows, using these algorithms does not satisfy the scalability requirement since the single computer may eventually become a bottleneck.

2.3 Parallel Collision Detection

A number of algorithms have been proposed to exploit parallelism to accelerate the collision detection process. Most of them are designed for shared-memory platforms. Kim et al. proposed a hybrid CCD to utilize the computational capabilities of a multi-core CPU and a GPU [Kim et al. 2009]. Tang et al. proposed another parallel algorithm based on fine grained dynamic task assignment, which exploits temporal coherence to guarantee load balance for self- and pairwise CCD [Tang et al. 2010b; Tang et al. 2011a]. Zhang and Kim proposed a collision detection approach for many-core platforms [Zhang and Kim 2014]. It utilizes a partitioning approach called 'p-partition front' to enable even partitioning of the workload and avoid complex dynamic load balancing. Pan and Manocha proposed a GPU-based method [Pan and Manocha 2011], which employs a clustering scheme and collision-packet traversal to perform collision queries on multiple configurations simultaneously. Lawlor and Kalée presented a space decomposition method for DCD in distributed-memory system, which uses trivial data transfer to maintain load balance [Lawlor and Kalée 2002].

The CCD algorithms reviewed so far are some of the most efficient approaches. However, most of them are designed for pairwise CCD or should be deployed on shared-memory single computers. The needs of a parallel N-body CCD approach for high-performance distributed-memory systems are largely unmet. In this paper, we present a parallel CCD algorithm, which aims to accelerate N-body culling by distributing its load across a GPU cluster (i.e. a distributed-memory system). This algorithm employs a dynamic partitioning method to balance the load in the system and utilize a series of culling methods to enhance the computational efficiency.

3 GPU-based N-body Continuous Collision Detection

As discussed previously in Section 2, most existing CCD acceleration approaches are designed for self- and pairwise CCD, and therefore the performance gain would be limited when they are applied to large-scale scenes that contain thousands of moving entities on a parallel computing system. In this section, we present the design of a GPU-based N-body CCD algorithm, which can efficiently determine a potentially colliding set (PCS) in a scene that contains a large number of entities. This approach is designed for single-node shared-memory system that consists of multi-core GPUs and CPUs and exploits parallelism and multi-threading capabilities to enhance its computational efficiency. It also forms the basis of the proposed distributed-memory CCD algorithm presented in Section 4.

3.1 Overview

The GPU-based N-body CCD process is given in Alg. 1. In the initialization stage, the algorithm first computes the center and radius of the objects as described in Section 3.2. During runtime, it projects the trajectory of entities onto a Cartesian coordinate axis according to their motion equation. Next, it performs a GPU-based parallel S&P to efficiently obtain all intersected trajectories, which are regarded as the PCS. It then builds a BVH for each entity in the PCS and traverses the bounding volume traversal tree (BVTT) [Du et al. 2015] for further culling. Finally, it employs an interval-iteration method to solve the primitive collision equation [Redon et al. 2002].

3.2 Construction of Sphere Bounding Volume

For any rigid body (entity) composed of small triangles, a fixed size Sphere Bounding Volume (SBV) can be constructed in the initialization stage. The construction algorithm first calculates the centre P_c of the entity by averaging the vertex coordinates of the triangles. This is given in Eq. 1.

$$P_c = \frac{\sum_{i=1}^{n} (A_i + B_i + C_i)}{3n} \tag{1}$$

where A_i, B_i, and C_i are the three vertices of the triangle, and n is the number of triangles in the entity. The max distance L_{max}

Algorithm 1 GPU-based N-body CCD

```
1: Create a SBV for each entity
2: for each time step do
3:     Project the trajectory of the SBVs onto the Cartesian coordinate axis and create CBVs
4:     Perform GPU-based S&P to obtain a PCS
5:     for each colliding trajectory pair in the PCS do
6:         Perform stream-based pairwise CCD (Alg. 2)
```

between the centre P_c and the entity's vertex P_i can be calculated by Eq. 2.

$$L_{max} = max\{||P_c - P_i||, 1 \leq i \leq m\} \quad (2)$$

where m is the number of vertices. Although this is a conservative method, it can guarantee that the size of the SBV will always remain unchanged during runtime, regardless of the orientation of the entity. Hence, we can use a continuous bounding volume (CBV) to bound the trajectory of the SBV during the time interval.

3.3 GPU-based Parallel Sweep and Prune

The original S&P approach [Cohen et al. 1995] is designed for DCD only. In this section, we present an extension of the original approach in order to support the parallel CCD process based on GPU.

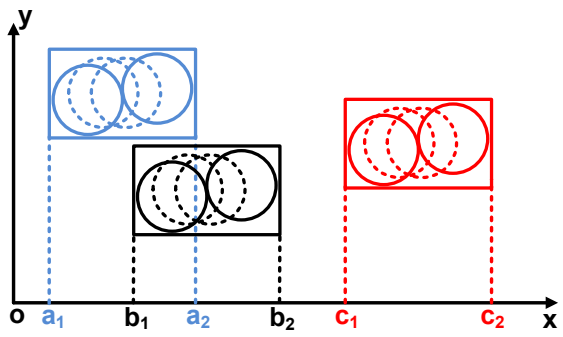

Figure 1: *CBVs are projected onto a coordinate axis.*

The parallel CCD approach begins by projecting the CBVs (continuous bounding volume, which are formed by the SBVs' trajectories) onto a coordinate axis (see Fig. 1). A list L containing the projections end points is constructed. By sorting L, it can determine which projections overlap. The original S&P uses insertion sort for this process. However, this approach is not suitable for single-node multi-core GPU systems, since it would be difficult to parallelize the insertion sort. To solve this problem, a GPU-based radix sort [Liu et al. 2010] is employed, which can determine the overlaps for each projection in a parallel manner. Specifically, a GPU core (as well as a thread) is used to determine a potentially colliding set (PCS) $P_i = \{(O_i, O_j), M_i > m_j, m_i < M_j\}$ for each CBV projection by sweeping L (m_i and M_i are the minimum and maximum boundaries of O_i). All P_i are then combined to obtain a final PCS $P = \cup P_i$.

3.4 Primitive Test

After the PCS is obtained from the previous stage, we employ a parallel stream-based approach to perform primitive-level CCD, in order to obtain the exact CCD results. In this approach, the geometric data are represented as streams. The underlying functional modules used in the algorithm (i.e. BVs and BVHs updates and BVH traversals) are mapped to computational kernels.

Five types of data streams are used in this process, namely collision pair stream S_c, transformation matrix stream S_{tm}, vertex stream S_v, BV stream S_{bv}, and BVH stream S_{bvh}.

- Collision pair stream S_c: after S&P traversal, all collision pairs in the PCS are put into S_c.
- Vertex stream S_v: it contains the geometric coordinates of the vertices of the components; in order to perform CCD between two discrete time steps, two vertex streams S_{v_n} and $S_{v_{n+1}}$ are used to store the vertex coordinates in the time interval $[t_n, t_{n+1}]$; in addition, S_{v_0} is used to represent the initial vertex coordinate stream.
- BV stream S_{bv} and BVH stream S_{bvh}: all bounding volumes of triangles are represented by S_{bv}; all components of an entity are enclosed by a single BVH S_{bvh}; S_{bv} and S_{bvh} are updated at each simulation time step based on S_v.
- Transformation matrix stream S_{tm}: when the coordinates of entities change, the transform matrices S_{tm} are sent to GPU, and a new vertex stream $S_{v_{n+1}}$ is computed with S_{v_0} and S_{tm}.

The algorithm of the parallel stream-based CCD is given in Alg. 2. It decomposes the process into several computation kernels, including S_v, S_{bv} and S_{bvh} update kernels, and S_c collision detection kernel. Due to the data independence of the sub-processes, all computation kernels can be processed on the GPU cores in a parallel manner.

Algorithm 2 Streams-based pairwise CCD.

```
1: for each simulation time step do
2:     Update S_v with S_tm
3:     Update S_bv and S_bvh with S_v
4:     for each collision pair c in S_c do
5:         if the leaf nodes of two S_bvh in c intersect then
6:             Return true
7:         else
8:             Return false
```

4 Parallel CCD for Distributed-Memory Systems

In this section, we present the design principles of the parallel CCD algorithm for distributed-memory systems. This algorithm aims to be deployed on a high-performance computing system that contains multiple working nodes. It employs a novel spatial decomposition method to distribute the workload of CCD across the nodes in order to enhance the overall runtime efficiency. It can also perform load-balancing at runtime based on dynamic task assignment.

4.1 Space Decomposition

With the use of spatial decomposition, the virtual space can be partitioned in multiple sub-spaces, which can localize the CCD tasks. Over the years, a number of non-uniform decomposition method such as k-d tree and octree have been proposed [Samet 2006]. However, uniform decomposition is the simplest and most suitable strategy for parallel implementation and dynamic task assignment.

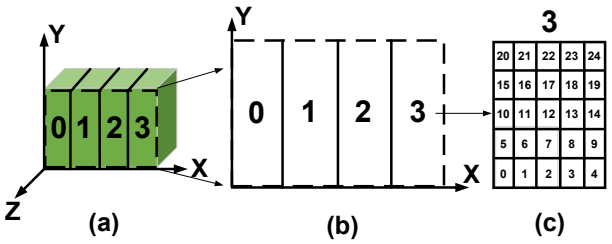

Figure 2: *Spatial decomposition: the virtual space is decomposed into n uniform sub-spaces (Fig. (b)). Each sub-space is then divided into $m \times m$ cells (Fig. (c)).*

Therefore, we adopted uniform decomposition in our implementation.

After decomposing the virtual space, in each sub-space the CCD process would become an individual sub-problem, and therefore the single-node GPU-based CCD process described in Section 3 can be applied.

In the initialization phase, the parallel algorithm uses a two-level space decomposition approach to partition the virtual space. Assume that the distributed-memory system that hosts the virtual environment has n nodes. The algorithm first decomposes the virtual space into n uniform sub-spaces, and labels them with ID i, $0 \leq i \leq n-1$. Each of these sub-spaces is distributed to the node at runtime (see Fig. 2(b)).

The algorithm then divides the sub-space into $m \times m$ cells in the X-Y plane, where m is a user-defined granularity. It labels the cells with ID j from the bottom-left to the top-right of the sub-space, where $0 \leq j \leq m \times m - 1$ (Fig. 2(c)). Hence, a cell can be located by the combination of IDs i and j.

4.2 Load Balancing

The load of the nodes may become uneven over time due to the arbitrariness of the entities' movement pattern. Therefore, load-balancing should be carried out in order to maximize the utilization of computational resources.

The basic idea of our load-balancing approach is to transfer the cells from a heavily loaded node to a less loaded node. It first sorts the nodes by the number of colliding entity pairs in the node. It then transfers the cells from the most loaded node to the least loaded node if the difference between them is larger than a pre-defined threshold. This is an adaptive process, which allows only one cell transfer at each time step, in order to avoid a drastic change of workload.

4.3 Entity Migration

Entity may need to be transferred between nodes during runtime. When an entity moves out of its current cell, we first check whether the target cell is on the same node. If this is not the case, we put the entity into a migration list and remove the entity from its current cell. After all migrating entities are obtained, we transfer them to the corresponding target nodes according to the migration list.

If an entity lies on the border of two cells, it would be assigned to both of them. Moreover, if the two cells are on two different nodes, the entity would exist on both of the nodes. Since we use lock-step synchronization in our implementation, there would be no ambiguity in the collision detection results.

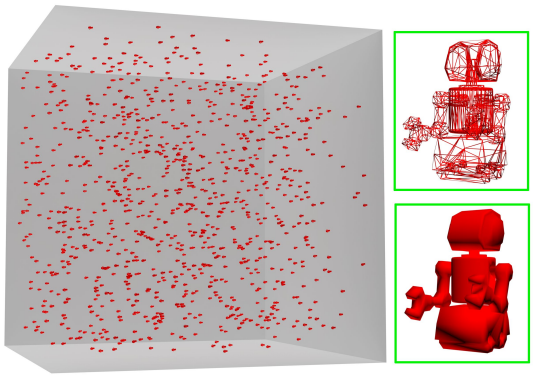

Figure 3: *Nanorobots simulation. In this simulation, a large mount of nanorobots move randomly in an virtual space. Each robot consists of $0.3K$ vertices and $0.7K$ triangles. There are many inter-object collisions.*

Furthermore, during the load-balancing process, if a cell is transferred to another node, all entities that it contains would be put into the migration list.

5 Experimental Evaluation

We have implemented and performed an experimental evaluation for the proposed parallel CCD algorithm on the supercomputer TSUBAME [Shimokawabe et al. 2011; GSIC 2016], which operates at the GSIC Center at the Tokyo Institute of Technology and is currently ranked No. 31 on the TOP500 list (as of October 2016). Each node of TSUBAME is comprised of 2 $\times$ 6-core Intel Xeon-E5670 CPUs and 3 $\times$ Nvidia Tesla K20X GPUs (1.31TFLOPS each) with 54GB main memory. The GPU computations of our algorithm were implemented with CUDA and the message passing protocols were implemented based on OpenMPI [MPI 2015]. Furthermore, lock-step approach were implemented to synchronize the working nodes.

5.1 Benchmarks and Approaches

Since currently there is no widely used benchmark for N-body CCD in a scene that contains thousands of entities, our evaluation is based on a nanorobots simulation [Liu 2015]. This simulation is comprised of a scene of hundreds of thousands of moving nanorobots (rigid bodies), with each nanorobot consists of $0.3K$ vertices and $0.7K$ triangles (see Fig. 3). A simple collision response mechanism is also implemented (There are many inter-object collisions, when collision occurs, the direction of entity's velocity is reversed).

Three approaches were implemented and tested in the nanorobots simulation to evaluate the performance of the proposed algorithm:

- Parallel GPU-based N-body CCD for distributed-memory systems with load balancing (PGCCD+LB): This approach is the algorithm that is proposed in this paper, which distributes the workload of CCD across multiple nodes in a high-performance computing system, and carries out GPU-based N-body and primitive-level cullings in order to efficiently obtain a final collision set. Dynamic load-balancing based on the approach described in Section 4.2 is performed to better utilize the computational resources in the computing system.
- Parallel GPU-based N-body CCD for distributed-memory systems without load balancing (PGCCD): This approach is

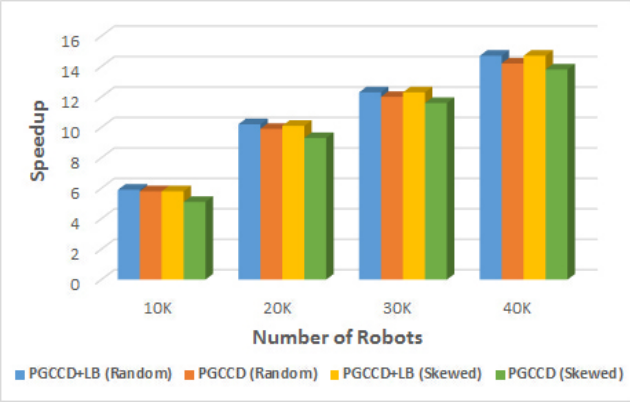

Figure 4: *Speedup of PGCCD over PCCCD (Number of Robots varies)*

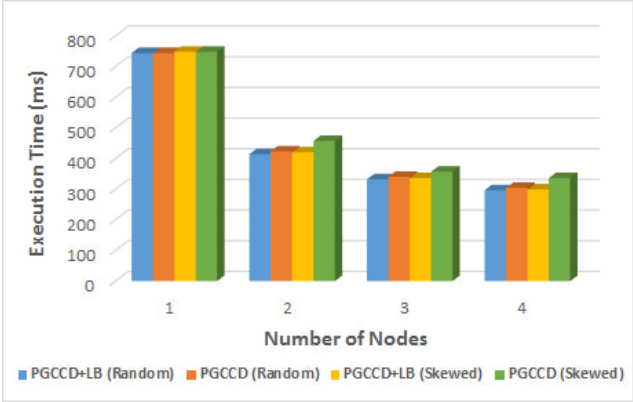

Figure 5: *Execution Time of PGCCD (Number of Nodes varies)*

the same as PGCCD+LB, except that the load-balancing process is not performed.

- Parallel CPU-based N-body CCD for distributed-memory systems (PCCCD): This approach is an extension of Lawlor's algorithm [Lawlor and Kalée 2002]. We chose it as a subject of comparison since, among the algorithms we review in Section 2, it is the only approach that is designed for distributed-memory systems. However, since Lawlor's original approach is a DCD algorithm, for the sake of fairness, we converted it to a CCD algorithm that supports primitive-level CCD. Its workload distribution mechanisms remain completely unchanged.

- GPU based N-body CCD for a shared-memory single computer: To the best of our knowledge, GPU based N-body CCD has not been proposed in the literature. The closest approach is [Liu et al. 2010], which performs GPU-based N-body DCD on a shared-memory single computer, but omits the process of primitive-level CCD. Therefore, comparing this approach with our approach would not be fair. We instead compared the performance of PGCCD on a single node and multiple nodes, in order to demonstrate the strength of the proposed algorithm over similar single-node shared-memory GPU-based algorithms such as Liu et al.'s approach.

Furthermore, in order to evaluate the algorithms under even and uneven distribution of workload, we define two movement patterns for the entities.

- Random Movement Environment: The entities are uniformly distributed to the scene and move randomly during runtime.

- Skewed Environment: The entities tend to move to the eight corners of the scene during runtime. When they reach a corner, they would stay for a period of time before moving to a random new corner. It is expected that more entities will concentrate at the corners of the scene over time.

5.2 Number of Entities

The first set of experiments evaluates the runtime efficiency of the proposed approach in both random movement and skewed environments, with the number of robots extending from 10,000 to 40,000. In the simulations, 4 nodes of TSUBAME were used and the predefined threshold of load-balancing was set to 15%.

Since the execution time of PCCCD obtained from the experiments is many times larger than that of the proposed algorithm, it would be difficult to include the results of both approaches in the same figure. Therefore, we calculated the speedup of our approaches over PCCCD, which is shown in Figure 4. It is not difficult to see that the performance of PGCCD+LB and PGCCD is much better than that of PCCCD, as both approaches have a large speedup over PCCCD in all cases. This indicates that the proposed approach is more scalable than PCCCD (i.e. the extension of Lawlor's approach) when deploying on a high-performance computing system.

Moreover, PGCCD+LB is only slightly faster than PGCCD in the random movement environment, which implies that although load-balancing can improve the performance of the proposed approach, its effect is not very significant in this setting. This may be due to the fact that load imbalance rarely happens in a scene of random moving entities.

Furthermore, we can see from the results that if load-balancing is not employed, the speedup of PGCCD in the skewed environment is slightly worse than that of the random movement environment. This confirms our expectation that the performance of PGCCD would be affected if the workload is uneven among nodes. On the other hand, if load-balancing is employed, the performance of PGCCD in both environments becomes similar. This also indicates that the improvement of PGCCD+LB over PGCCD is more significant in the skewed environment than in the random movement environment.

5.3 Number of Nodes

The second set of experiments evaluates the runtime efficiency of the proposed approach in random movement and skewed environments, with the number of working nodes extending from 1 to 4. The number of robots was set to 40,000 in the experiments.

Figure 5 shows the results of this evaluation. Since the execution time of PCCCD is many times larger than the proposed approach, its results are not included in the figure. The first observation is that the execution time of PGCCD+LB and PGCCD drops gradually with the increase in the number of working nodes. In particular, PGCCD+LB was able to complete the whole CCD process for 100,000 entities in $300ms$. This suggests that the proposed algorithm is scalable when running on a high-performance computing system.

Furthermore, as mentioned previously, the results of PGCCD on a single node represents the performance of similar shared-memory GPU-based algorithms such as [Liu et al. 2010]. The fact that PGCCD on four nodes has a speedup of 2.6 over PGCCD on a single node suggests that the proposed algorithm has much better

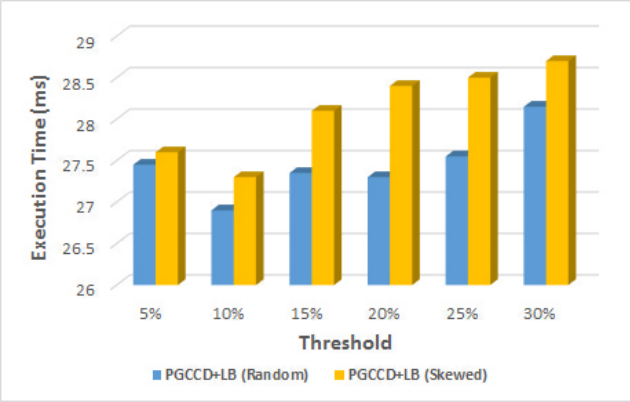

Figure 6: *Execution Time of PGCCD+LB (Threshold varies)*

performance than the shared-memory algorithm when deploying on a high-performance computing system.

However, the parallel efficiency of PGCCD drops significantly when more nodes are involved in the experiments. We can see from the results that when we increased the number of nodes from 3 to 4, the execution time reduction of PGCCD in all cases was less than 10%. Therefore, it can be expected that even if we add more nodes to the experiments, the performance gain of PGCCD would be insignificant.

5.4 Threshold

In the last set of experiments, we evaluate the performance of the proposed load-balancing method based on different threshold values. In our implementation, the threshold, which extends from 5% to 30%, was used to control the percentage of workload difference between the most and least loaded nodes. We used four nodes in this evaluation and the number of robots was set to 10,000.

It is important to point out that the choice of threshold is a trade-off and should always be application dependent. We measured the execution time of PGCCD+LB in random movement and skewed environments, which is given in Fig. 6. The first observation is that the computational overhead of the skewed environment is higher than that of the random movement environment even when load-balancing is performed, due to the fact that it would be more difficult to balance the workload in the former case. However, the overhead can be reduced by carefully choosing the value of threshold. It can be observed that there is a significant computational overhead when the threshold is smaller than or larger than 10%. Apparently, if a small threshold is chosen, the chance of cell transfer would be increased. On the other hand, a large threshold makes the load-balancing process more tolerant to workload imbalance, resulting in poor utilization of resources. Therefore, it can be concluded that, the optimal threshold is 10% for the current setting of experiments.

6 Conclusion

CCD is a process to detect collisions of polygons between successive time steps, which is usually more time-consuming than a DCD process. Over the years, a number of CCD approaches have been proposed, which sought to improve its computational efficiency. However, they are mainly designed for self- and pairwise CCD. Moreover, they are either designed as sequential process, or are only suitable to be executed on shared-memory single computers. Therefore, deploying them on high-performance parallel computing systems would not increase their computational efficiency.

In this paper, we present a parallel CCD acceleration approach, which is designed for distributed-memory GPU clusters. Our approach exploits parallelism by distributing the load of CCD across a cluster of nodes and employs a dynamic partitioning method to balance the load in the system. We also employ a number of GPU-based culling methods to enhance the computational efficiency of the CCD process. We have implemented our approach with MPI and CUDA and performed experimental evaluations on the supercomputer TSUBAME. Experimental results demonstrate that our approach is more computationally efficient than existing sequential/shared-memory single-node approaches, and therefore is more suitable to be applied to distributed simulation based virtual environment.

For future work, we will extend our parallel CCD approach to support deformable 3D models, which may involve a large number of self-collisions. Moreover, we will investigate other techniques to improve the efficiency of GPU-based parallel CCD, such as replacing the parallel S&P method by spatial decomposition methods.

Acknowledgments

This research has been partially supported by JST CREST (Core Research for Evolutionary Science and Technology) and National Nature Science Foundation of China (No. 61502130).

References

BERGEN, G. V. D. 1998. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 1–14.

BRADSHAW, G., AND O'SULLIVAN, C. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics 23*, 1–26.

COHEN, J., LIN, M., MANOCHA, D., AND PONAMGI, M. 1995. I-collide: An interactive and exact collision detection system for large-scale environments. In *I3D*.

CURTIS, S., TAMSTORF, R., AND MANOCHA, D. 2008. Fast collision detection for deformable models using representative-triangles. In *Proceedings of the ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM Press, 61–69.

DU, P., AND LIU, E. S. 2016. RNCCD: Rotation-aware N-body Continuous Collision Detection. In *Proceedings of the 33rd Computer Graphics International (CGI 16)*.

DU, P., TANG, M., AND TONG, R. 2012. Fast continuous collision culling with deforming non-collinear filters. *Computer Animation and Virtual Worlds 23*, 6-8, 375–383.

DU, P., ZHAO, J., PAN, W., AND WANG, Y. 2015. GPU accelerated real-time collision handling in virtual disassembly. *Journal of Computer Science and Technology 30*, 3, 511–518.

FUJIMOTO, R. M. 2000. *Parallel and Distributed Simulation Systems*. John Wiley and Sons, Inc.

GSIC. 2016. *TSUBAME 2.5 User's Guide*. Tokyo Institute of Technology, September.

KIM, D., HEO, J. P., HUH, J., KIM, J., AND YOON, S. E. 2009. HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. *Computer Graphics Forum 28*, 1791–1800.

KLOSOWSKI, J., HELD, M., MITCHELL, J., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics 4*, 21–37.

LAWLOR, O. S., AND KALÉE, L. V. 2002. A voxel-based parallel collision detection algorithm. In *ICS*.

LEE, J., KWON, O., ZHANG, L., AND YOON, S. 2014. A selective retraction-based RRT planner for various environments. *IEEE Transactions on Robotics 30*, 4, 1002–1011.

LIU, F., HARADA, T., LEE, Y., AND KIM, Y. J. 2010. Real-time collision culling of a million bodies on graphics processing units. In *SIGGRAPH ASIA*.

LIU, E. S. 2015. On the Scalability of Agent-based Modeling for Medical Nanorobotics. In *Proceedings of Winter Simulation Conference (WSC) 2015*.

MPI, O., 2015. A high performance message passing library. http://www.open-mpi.com/.

NEYLAND, D. L. 1997. *Virtual Combat: A Guide to Distributed Interactive Simulation*. Stackpole Books.

PAN, J., AND MANOCHA, D. 2011. Gpu-based parallel collision detection for real-time motion planning. *Algorithmic Foundations of Robotics IX, volume 68 of Springer Tracts in Advanced Robotics*, 211–228.

PERUMALLA, K., FUJIMOTO, R., MCLEAN, T., AND RILEY, G. 2002. Experiences applying parallel and interoperable network simulation techniques in on-line simulations of military networks. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 21*, 279–288.

SAMET, H. 2006. *Foundations of Multidimensional and Metric Data Structures.* Kaufmann.

SHIMOKAWABE, T., TAKAKI, T., ENDO, T., YAMANAKA, A., MARUYAMA, N., AOKI, T., NUKADA, A., AND MATSUOKA, S. 2011. Peta-scale phase-field simulation for dendritic solidification on the tsubame 2.0 supercomputer. In *International Conference for High Performance Computing*.

SMED, J., KAUKORANTA, T., AND HAKONEN, H. 2002. A Review on Networking and Multiplayer Computer Games. Tech. Rep. 454, Turku Centre for Computer Science, April.

STEED, A., AND OLIVEIRA, M. F. 2010. *Networked Graphics: Building Networked Games and Virtual Environments*. Morgan Kaufmann.

TANG, M., CURTIS, S., YOON, S. E., AND MANOCHA, D. 2008. Interactive continuous collision detection between deformable models using connectivity-based culling. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*. ACM Press, Stony Brook, New York, 25–36.

TANG, M., MANOCHA, D., AND TONG, R. F. 2010. Fast continuous collision detection using deforming non-penetration filters. In *Proceedings of the ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM Press, 7–14.

TANG, M., MANOCHA, D., AND TONG, R. F. 2010. MCCD: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models 72*, 7–23.

TANG, M., MANOCHA, D., LIN, J., AND TONG, R. F. 2011. Collision-streams: fast GPU-based collision detection for deformable models. In *Proceedings of the ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM Press, 63–70.

TANG, M., MANOCHA, D., YOON, S. E., DU, P., HEO, J. P., AND TONG, R. F. 2011. VolCCD: Fast continuous collision culling between deforming volume meshes. *ACM Transactions on Graphics 30*, 111–125.

TANG, M., MANOCHA, D., OTADUY, M., AND TONG, R. 2012. Continuous penalty forces. *ACM Transactions on Graphics 31*, 4, 107:1–107:9.

TOP500, 2016. TOP500 Supercomputer, October.

WONG, S., LIN, W., HUNG, C., HUANG, Y., AND LII, S. 2013. Radial view based culling for continuous self-collision detection of skeletal models. *ACM Transactions on Graphics 32*, 4, 114:1–114:10.

ZHANG, X., AND KIM, Y. 2014. Scalable collision detection using p-partition fronts on many-core processors. *IEEE Transactions on Visualization and Computer Graphics 20*, 447–456.