2010 International Conference on Intelligent Systems, Modelling and Simulation

Offloading Region Matching of Data Distribution Management with CUDA

Shih Hsiang Lo and *Yeh Ching Chung*
Computer Science Department
National Tsing Hua University
Hsinchu, Taiwan
albert@sslab.cs.nthu.edu.tw and
ychung@cs.nthu.edu.tw

Fang Ping Pai
Aeronautical Systems Research Division
CHUNG_SHAN Institute of Science and Technology
Taichung, Taiwan
fppai118@gmail.com

Abstract—**Data distribution management (DDM) aims to reduce the transmission of irrelevant data between High Level Architecture (HLA) compliant simulators by taking their interesting regions into account (i.e. region matching). In a large-scale simulation, computation intensive region matching would have a direct impact on the simulation performance. To deal with the high computation cost of region matching, the whole process of region matching is offloaded to graphical processing units (GPUs) based on Computer Unified Device Architecture (CUDA). Two approaches are proposed to perform region matching in parallel. Several metrics, including different numbers of regions, different sizes of regions and different distributions of regions, are used in the experimental tests. The experimental results indicate that the performance of region matching on a GPU can be improved more than one or two orders of magnitude in comparison with that on a CPU.**

Region Matching; CUDA; Data Distribution Management; High Level Architecture

I. Introduction

Data distribution management (DDM) is one of run-time infrastructure (RTI) managements in the High Level Architecture [1]. DDM is designed to reduce unnecessary transmission between HLA compliant simulators (termed federates) during simulation and is essential to support the execution of a large-scale simulation. According to the DDM services defined in the HLA interface specification, a federate can clearly states a set of data requirements (ranges), which is defined as a region in the HLA. Specifically, federates can publish data in the interesting regions (termed publication regions) over which they are willing to send data. Similarly, federates can subscribe data in the interesting regions (termed subscription regions) from which they are willing to receive data. Afterwards, RTI will deliver data from publication federates to the interested subscription federates only if the corresponding publication regions overlap the corresponding subscription regions. The process of finding overlaps between publication and subscription regions is called region matching.

When a significant number of regions are involved in a simulation, it takes a considerable amount of time to perform region matching. Furthermore, performing computation intensive region matching affects the time to perform other managements in RTI, e.g. Object Management and Time Management. Thus it could become the performance bottleneck of the whole RTI system. Several region matching approaches [2-9] have reduced the overhead of region matching under certain circumstances. However, the computation cost taken by region matching is still high for time-constrained applications and little research has been conducted on performing region matching in parallel. To address this performance issue, we make region matching run in parallel and also offload the whole execution of region matching to graphical processing units (GPUs) based on Compute Unified Device Architecture (CUDA) [10]. CUDA is a general-purpose and widely-use parallel computing architecture. It leverages GPUs in computing data-parallel and/or thread-parallel jobs. Many applications have been successfully implemented with CUDA, such as [11-12].

In this paper, we present two parallel region matching approaches running on GPUs. One is parallel region-based approach; the other is parallel grid-based approach. In the parallel region-based approach, each new or modified region (termed updated region) is assign to a CUDA thread in order to recalculate the overlapped information. In the parallel grid-based approach, each updated region is assigned to a CUDA thread as well. Each CUDA thread is responsible for mapping the updated region on to a set of grid cells according to the ranges of the region and the grid cell size. Then each CUDA thread only needs to compare the updated region with other regions in that set of grid cells. In both approaches, the computing part is not only shifted to GPUs but also the region information (e.g. the ranges of regions) and the overlapped information (e.g. the overlapped regions) are maintained in the device memory for DDM.

To evaluate the proposed approaches, we implemented these approaches using CUDA in our RTI system that follows IEEE 1516 standard [1]. Several metrics, including different numbers of regions, different sizes of regions and different distributions of regions, are used in the experimental tests. The experimental results show that the speedup of region matching can reach one or two orders of magnitude.

The organization of this paper is as follows. In Section II, we briefly discuss the region matching approaches reported in the literature. In Section III, parallel region matching approaches are presented. In Section IV, the experimental results are presented in details. We conclude our work in Section V.

II. Related work

A. Region-based Approach

In [13-14], publication regions are compared with all subscription regions to find overlaps between these regions. The idea is straightforward and the computation cost is quadratic with the number of regions. In a large-spatial simulation, this approach takes much of time to compare unrelated regions (i.e. the regions are not overlapped). On the other hand, it can achieve high performance when all regions are highly overlapped [3, 15]. In [6], a part of DDM computation is shifted to network processors. The region matching and the multicast management of DDM are executed in a Myrinet network card. As a result, there is a coarse-grained task parallelism between DDM and other managements of the RTI system.

B. Grid-based Approach

In [11-13], the authors proposed a method to reduce the region matching cost of the region-based approach. This approach partitions an *N*-dimensional space into grid cells and maps all regions on to these grid cells. Then each region is only compared with the other regions that mapped on to the same grid cells. However, an inappropriate grid cell size results in poor performance. When a grid cell size is large, unnecessary computation overhead is introduced. On the other hand, when a grid cell size is small, this could lead to redundant computations. To solve the problem, a cost model was proposed in [7] to estimate the region matching cost and then the grid cell size can be dynamically adjusted based on the matching cost model.

C. Sort-based Approach

Several region matching approaches using sorting technique were proposed in [3, 9, 16]. In [3], the end points in each dimension of all regions are first sorted and recorded in a sorted list. The sorted list is then scanned to get the overlapped information in each dimension. This approach is not designed for the simulations where regions will be modified at run-time. For this reason, a dynamic sort-based algorithm for a large-spatial simulation is presented in [16]. When a region is updated, the dynamic sort-based algorithm shifts the end points of this region from old positions to new positions and then scans the sorted end points within a dynamic range. In [9], the authors proposed a P-Pruning algorithm for DDM. The principle of this approach is similar to the work in [3, 16].

D. Partition-based Approach

This approach first splits an *N*-dimensional space into fixed-size partitions and then adjusts the partition space as needed. In [17], the authors proposed a region matching approach that clusters an *N*-dimensional space into varied-size partitions based on region access patterns as well as the location of simulation object. In theory, the execution of region matching can be evenly distributed to different hosts. A simple partition-based approach using quad-tree structure in helping dynamic partition was proposed in [8].

III. Parallel Region matching approaches

A. Preliminary

To run CUDA programs, the input data is copied from the host to the device (i.e. any CUDA-Enabled product), then the GPU execute CUDA kernels and finally the output data is copied back to the host. In order to avoid too much data copied between the host and the device, the whole execution of region matching is offloaded to GPUs and additionally the data related to performing region matching is maintained in the device, including all region information, all overlapped information etc. Fig 1. illustrates the offloading model of region matching between the host and the device. When regions are added or modified (i.e. updated regions) by federates, only the information of updated regions is copied from the host to the device. Then, the Region Matching procedure is executed in the device so as to update the overlapped information related to these updated regions. After finishing the procedure, the updated overlapped information is copied back to the host.

Based on the offloading model, we consider the qualities of different region matching approaches. Among the region matching approaches reported in the literature, the region-based and the grid-based approaches are efficiently feasible to be implemented in the CUDA programming model. For the region-based approach, each region simply compares all regions to find overlaps for the region. The comparison operations for each region can be executed in parallel. For the grid-based approach, the operation of mapping regions can be executed in parallel as long as the region information can be recorded in the grid cells concurrently. Atomic instructions provided in the CUDA programming model are used to guarantee it. After the operation of mapping regions, the operation of matching regions is similar to that of the region-based approach. For the sort-based approaches, it is essential that the ranges of regions are sorted at each dimension. It is possible to parallel the sorting operation. However, at run-time, the operation of shifting (moving) data in a list greatly affects the performance of region matching. The reason is that if many CUDA threads have to move a set of data in a list, only one CUDA thread can carry out the operation. For the partition-based approach, executing recursive functions or processing tree data structure are required to support the dynamic partition mechanism. In fact, CUDA is inappropriate to run the applications having these two operations.

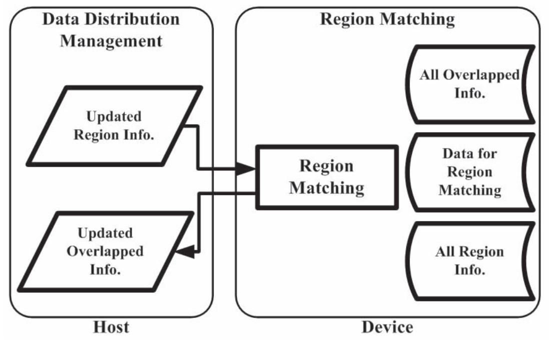

Figure 1. The offloading model of region matching.

B. *Parallel Region-based Approach*

This approach consists of four phases to perform region matching. The four phases are described as follows:

Phase 1: The information of updated publication and subscription regions is copied to the global memory of the device.

Phase 2: The overlapped information related to the updated regions is set as non-overlapped. Each updated region is assigned one CUDA thread. For each CUDA thread, it is necessary to reset the overlapped information of the assigned updated region and also to clear those regions which the assigned region overlapped before as non-overlapped.

Phase 3: Similar to Phase 2, a CUDA thread is assigned to process an updated region. Each CUDA thread will sequentially read all region information and compare the assigned region with all regions to find overlaps while all CUDA threads run in parallel as shown in Fig. 2. Fig. 2 shows how each CUDA thread loads and compares the region information. In the CUDA programming model, a thread block contains a set of threads. The thread block size in Fig. 2 is M and CUDA Threads 1, 2, …, and M are within the same block. The details of this phase are described in the following.

First, every CUDA thread within the same block loads the region information of some region from the global memory to the shared memory. As shown in Fig. 2, CUDA Thread 1 loads the first region r_1 to the shared memory, CUDA Thread 2 loads the second region r_2 to the shared memory, and so forth. In the CUDA model, every CUDA thread within the same block can share region information via the shared memory and access to the shared memory is far faster than access to the global memory. In this way, the number of loading region information is reduced to M (M^2 originally) and the region information is reused for M times. Only the information about M regions is loaded into the shared memory at a time because of the very limited size of the shared memory.

After the loading, every CUDA thread within the same block sequentially carries out the comparison operation for the assigned region and M regions that have been loaded into the shared memory. The principle of the comparison operation is that the lower and upper end points of a region are compared with those of the other region at each dimension. If two regions are overlapped with each other, the overlapped information of these two regions is updated. Atomic instructions provided in the CUDA programming model are used to make sure each CUDA thread will write to different positions of the overlapped information in case of the concurrent access of the same overlapped information.

When every CUDA threads within the same block complete the comparisons for M regions, the information of next M regions will be loaded into the shared memory in order to find overlaps with these M regions. The process repeats until every CUDA thread completes the comparison operations for all regions.

Phase 4: The updated overlapped information is copied back from the global memory of the device to the memory of the host.

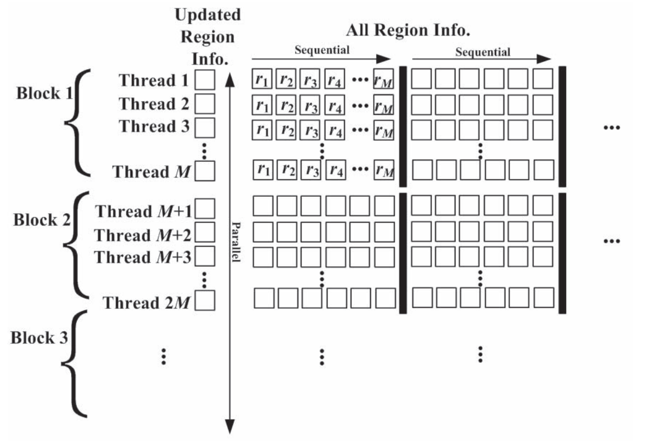

Figure 2. A schematic figure of the parallel region-based approach.

C. *Parallel Grid-based Approach*

Initially, an N-dimensional space is partitioned into a set of grid cells and each grid cell has the equal size. Consider the performance of region matching, these grid cells are created and maintained in the device. This approach consists of five phases to perform region matching. The five phases are described as follows:

Phase 1: The information of updated publication and subscription regions is copied into the global memory of device.

Phase 2: This phase is the same as Phase 2 described in the parallel region-based approach.

Phase 3: Each updated region is assigned one CUDA thread. The updated publication (subscription) region is mapped on to a set of grid cells, denoted as C, according to the ranges of this publication (subscription) region. The identity of the updated publication (subscription) region is recorded in the publication (subscription) list of each grid cell in C. Atomic instructions are used to make sure each CUDA thread will write to different positions of the publication or subscription list in case of the concurrent access of the same list. Fig. 3 shows an illustration of mapping regions. In Fig. 3, $PubList(c_i)$ and $SubList(c_i)$ represents the publication and subscription lists of grid cell c_i, respectively. For example, at grid cell c_{14}, publication region p_2 and subscription region s_2 are both mapped on to grid cell c_{14}. Note that if an updated region is a new one (i.e. not mapped before), the region is simply mapped on to grid cells. Otherwise, the region is unmapped before mapping it.

Phase 4: Each updated publication (subscription) region is assign to one CUDA thread. All CUDA threads run in parallel. Each CUDA thread will compare the assigned publication (subscription) region with the regions contained in the subscription (publication) lists of C. Fig 4. shows an example of the execution. In Fig. 4, p_2 and s_1 are assigned to CUDA Thread 1 and 2, respectively. CUDA Thread 1 will find the regions in the subscription lists of grid cell c_8,c_9,c_{13} and c_{14}, to carry out comparison operations. CUDA

Thread 2 will find the regions in the publication lists of grid cell c_{12},c_{13},c_{17} and c_{18}, to carry out comparison operations. That is, p_2 is compared with s_1 and s_2 and at the same time s_1 is compared with p_2.

Phase 5: the overlapped information is copied back from the global memory of the device to the memory of the host.

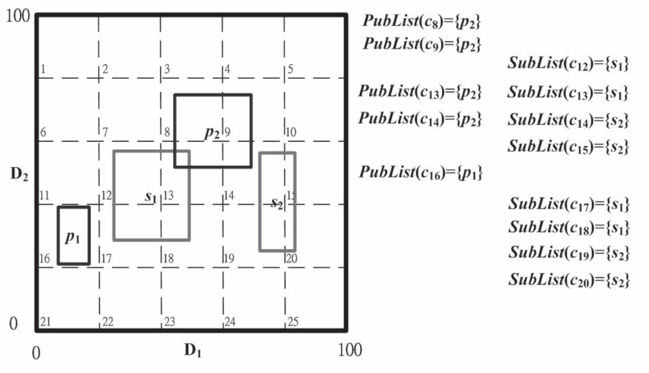

Figure 3. Four regions mapped on to grid cells and the publication and subscription lists of these grid cells.

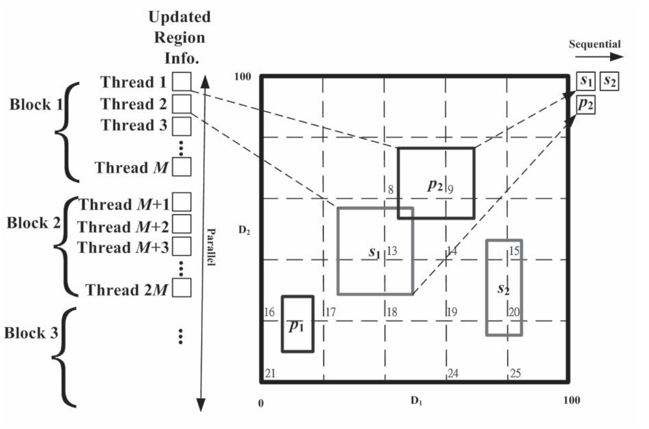

Figure 4. A schematic figure of the parallel grid-based approach

IV. Experiment

In the section, we evaluate the performance of the proposed region matching approaches running on GPUs. We use one machine with one CUDA-enabled product (Geforce 9600 GT) as our experimental platform. This machine has one quad-core processor 2.4 GHz CPU, 6 Gbytes DRAM and running Linux operating system with kernel version 2.6.28. Geforce 9600 GT has 8 CUDA processors and each CUDA processors contains eight CUDA cores. In this paper, each CUDA thread block contains 256 CUDA threads.

Two scenarios were simulated to evaluate the performance of the parallel region-based (REGION) and the parallel grid-based (GRID) approaches on the quad-core processor and the Geforce 9600 GT. Each simulated scenario has at most 32768 simulation regions that were distributed to a 10000×10000 2-dimensional battlefield. The numbers of publication and subscription regions are the same. In this paper we only show the simulation cases in 2-dimensional space. Our approaches also can be extended to the simulation in an *N*-dimensional space. In each simulated scenario, the sizes of the publication and the subscription regions of a simulation object are the same. For each time step, the position of each simulated region is randomly moved toward North, South, East or West direction. During the simulation, the moving distance is set to half the region size. For all simulation cases, the time of region matching is averaged over a period of 30 time steps.

A. Scenario I: Uniform Distribution of Regions

In this scenario, all simulated regions are uniformly (randomly) distributed to a 2-dimensional battlefield. The number of regions simulated is from 2048 to 32768. The region sizes simulated are *RS*=10×10 (small region size) and *RS*=100×100 (large region size). The grid cell size of the parallel grid-based approach is set to 100×100 for each test case. Fig. 5 and 6 show the speedup of region matching under *RS*=10×10 and *RS*=100×100, respectively. In each figure, the *x*-axis represents the number of regions simulated and the *y*-axis represents the speedup compared with the region matching time taken by the CPU. From Fig. 5 and 6, we have the following remark.

Remark 1: For REGION(9600 GT), the performance is up to 200× more than that of a CPU (i.e. the quad-core processor). For GRID(9600 GT), the performance is up to 150× more than that of a CPU. More the number of simulated regions increases, higher the speedup can be reached. The one reason is that the time to access the global memory of the device can be covered by arranging other CUDA thread blocks to use CUDA processor if the number of CUDA thread block is sufficiently large. The other reason is that shared memory is exploited to make CUDA threads access data efficiently. The results indicate that the proposed parallel region matching approaches can perform region matching for many regions effectively.

Fig. 7 and 8 show the region matching time for the proposed approaches under *RS*=10×10 and *RS*=100×100, respectively. In each figure, the *x*-axis represents the number of regions simulated and the *y*-axis represents the time to perform the parallel region matching approaches on the GPU. From Fig. 7 and 8, we have the following remarks.

Remark 2: We can see that the performance of the parallel grid-based approach is better than that of the parallel region-based approach. That is because the parallel region-based approach compares regions with many unrelated regions. It proves that, in this scenario, the parallel grid-based approach can reduce the unnecessary comparisons between unrelated regions and also execute in parallel effectively.

Remark 3: The performance of the parallel grid-based approach is sensitive to region size. The reason is that a large region is mapped on to more grid cells in comparison to a small region. As a result, atomic instructions are more often used (in Phase 3 of the parallel grid-based approach) to guarantee the correctness of the concurrent access of the publication and/or subscription lists.

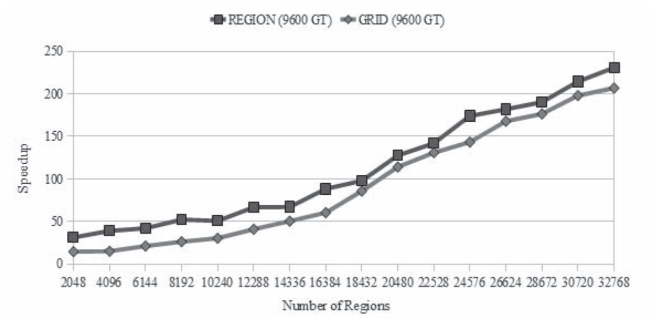

Figure 5. The speedup of region matching under *RS*=10×10.

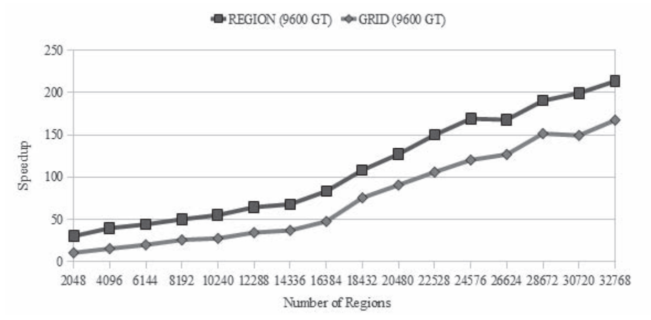

Figure 6. The speedup of region matching under *RS*=100×100.

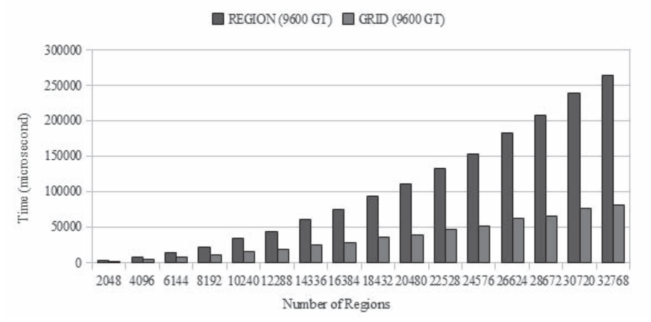

Figure 7. The region matching time under *RS*=10×10.

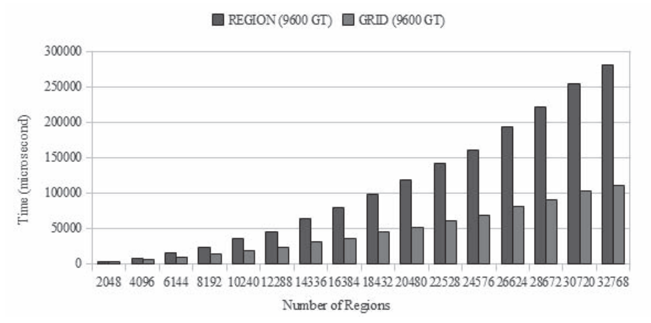

Figure 8. The region matching time under *RS*=100x100.

B. Scenario II: Non-Uniform Distribution of Regions

In this scenario, some of regions are distributed around three crowded areas and other regions are uniformly distributed to this space. Three crowded areas are located around coordinates (1000, 1000), (5000, 5000) and (9000, 9000). The number of regions simulated is from 2048 to 32768. The region sizes simulated are *RS*=10×10 and *RS*=100×100. For each simulation case, 20 percentages of all regions are around the three crowded areas. The numbers of regions around three crowded areas are nearly the same. The grid cell size of the parallel grid-based approach is set to 100×100 for each test case. Fig. 9 and 10 show the speedup of the region matching approaches under *RS*=10×10 and *RS*=100×100, respectively. In each figure, the *x*-axis represents the number of regions simulated and the *y*-axis represents the speedup compared with the region matching time taken by a CPU. From Fig. 9 and 10, we have the following remark.

Remark 4: In Fig. 9 and 10, the performance of the region-based approach achieves about 25× whereas the performance of the parallel grid-based approach is about 5×. In this scenario, the speedup results are not good as those shown in Scenario I. The main reason is that many CUDA threads would access the same overlapped information of the regions around the crowded area. This required atomic instructions to make sure the correctness of the concurrent access of the overlapped information. Using atomic instructions is harmful for the performance of both approaches. However, the speedup of the parallel region-based approach still achieves substantial performance improvement on 9600 GT.

Fig. 11 and 12 show the region matching time of REGION (9600 GT) and HYBRID (9600 GT) under *RS*=10×10 and *RS*=100×100, respectively. In each figure, the *x*-axis represents the number of regions simulated and the *y*-axis represents the time to perform region matching approaches running on the GPU. From Fig. 11 and 12, we have the following remark.

Remark 5: We can observe the performance of the parallel region-based approach is better than that of the parallel grid-based approach. For the parallel region-based approach, since some of regions are around the crowded areas, the occurrences of comparing unrelated regions are reduced, that is, the efficiency of comparing region is improved. However, for the parallel grid-based approach, it takes more time to map regions at the crowded areas and therefore it uses atomic instructions more as described in Remark 3.

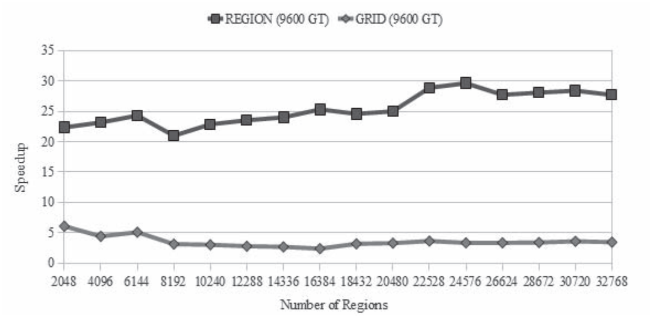

Figure 9. The speedup of region matching for three crowded areas under *RS*=10×10.

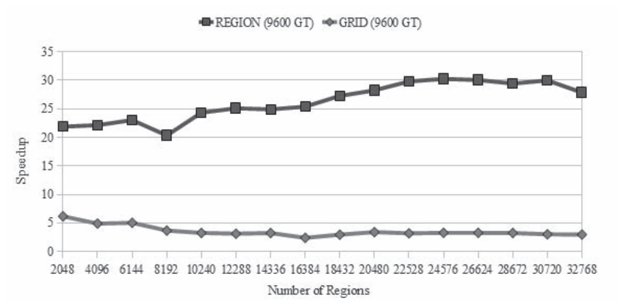

Figure 10. The speedup of region matching for three crowded areas under *RS*=100×100.

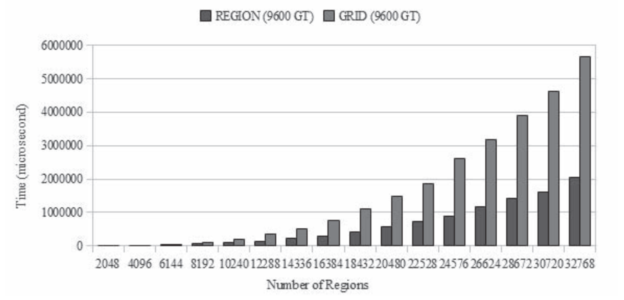

Figure 11. The region matching time for three crowded areas under *RS*=10×10.

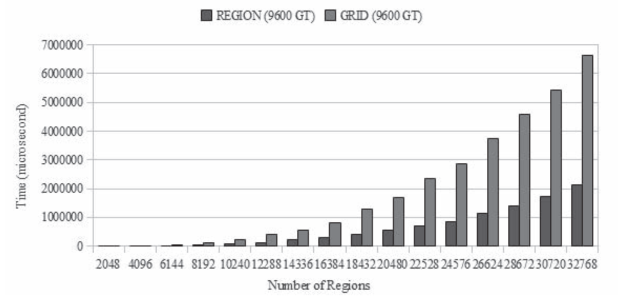

Figure 12. The region matching time for three crowded areas under *RS*=100×100.

V. CONCLUSION

DDM services in the HLA RTI provide a good mechanism to reduce unnecessary transmission and irrelevant reception over the network. To ensure only necessary data are transmitted, region matching for publication and subscription regions is indispensable. In this paper we focus on region matching of DDM and have proposed two region matching approaches with fine-grained data parallelism. We implemented both approaches in our RTI system. Two scenarios, uniform and non-uniform distributions of regions, are considered to evaluate the performance of the proposed approaches. The results indicate that the proposed region matching approaches based on CUDA can greatly reduce the time to perform region matching. Further performance analysis of the proposed approaches will be showed in the near future.

ACKNOWLEDGMENT

The work of this paper is partially supported by CHUNG-SHAN Institute of Science & Technology under contract XW98141P and National Science Council under contract 98-2623-E-007-007-D. The authors would like to thank anonymous referees for their comments.

REFERENCES

[1] IEEE, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Framework and Rules," vol. 1516-2000, ed, 2000.

[2] G. Tan, Z. Yusong, and R. Ayani, "A hybrid approach to data distribution management," in Distributed Simulation and Real-Time Applications, 2000. (DS-RT 2000). Proceedings. Fourth IEEE International Workshop on, 2000, pp. 55-61.

[3] C. Raczy, G. Tan, and J. Yu, "A sort-based DDM matching algorithm for HLA," ACM Trans. Model. Comput. Simul., vol. 15, pp. 14-38, 2005.

[4] A. Boukerche, N. McGraw, C. Dzermajko, and K. Lu, "Grid-Filtered Region-Based Data Distribution Management in Large-Scale Distributed Simulation Systems," in Proceedings of the 38th Annual Simulation Symposium, 2005, pp. 259-266.

[5] R. Ayani, F. Moradi, and G. Tan, "Optimizing cell-size in grid-based DDM," presented at the Proceedings of the fourteenth workshop on Parallel and distributed simulation, Bologna, Italy, 2000.

[6] A. Santoro and R. M. Fujimoto, "Offloading Data Distribution Management to Network Processors in HLA-Based Distributed Simulations," Parallel and Distributed Systems, IEEE Transactions on, vol. 19, pp. 289-298, 2008.

[7] S. H. Lo, C. A. Chiu, D. Y. Hong, F. P. Pai, and Y. C. Chung, "MGRID: A Modifiable-Grid Region Matching Approach for DDM in the HLA RTI," in Sping Simulation Multiconference, San Diego, CA, 2009.

[8] O. Eroglu, H. A. Mantar, and F. E. Sevilgen, "Quadtree-based approach to data distribution management for distributed simulations," presented at the Proceedings of the 2008 Spring simulation multiconference, Ottawa, Canada, 2008.

[9] P. Gupta and R. K. Guha, "A Comparative Study of Data Distribution Management Algorithms," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, vol. Volume 4 Number 2, 2007.

[10] CUDA Zone -- The resource for CUDA developers. Available: http://www.nvidia.com/object/cuda_home.html

[11] L. Nyland, M. Harris, and J. Prins, Fast N-Body Simulation with CUDA, GPU Gems 3 ed.: Addison-Wesley, 2007.

[12] S. A. Manavski, "CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography," in Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on, 2007, pp. 65-68.

[13] D. J. Van Hook and J. O. Calvin, "Data Distribution Management in RTI 1.3," in Proceedings of the Spring Simulation Interoperability Workshop, Orlando, 1998.

[14] D. Wood, "Implementation of DDM in the MAK High Performance RTI," presented at the Proceedings of the Simulation Interoperability Workshop.

[15] J. Y. C Raczy, G Tan, SC Tay, R Ayani, "Adaptive data distribution management for HLA RTI," presented at the Proceedings of 2002 European Simulation Interoperability, 2002.

[16] K. Pan, S. J. Turner, W. Cai, and Z. Li, "An Efficient Sort-Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment," presented at the Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, 2007.

[17] B. L. Kumova, "Dynamically adaptive partition-based data distribution management," in Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on, 2005, pp. 292-300.