Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

AN ANALYSIS OF PARALLEL INTEREST MATCHING ALGORITHMS IN DISTRIBUTED VIRTUAL ENVIRONMENTS

Elvis S. Liu

IBM Research
and University College Dublin
Dublin, IRELAND

Georgios K. Theodoropoulos

Institute of Advanced Research Computing
Durham University
Durham, UK

ABSTRACT

Interest management is a filtering technique which is designed to reduce bandwidth consumption in Distributed Virtual Environments. This technique usually involves a process called "interest matching", which determines what data should be filtered. Existing interest matching algorithms, however, are mainly designed for serial processing which is supposed to be run on a single processor. As the problem size grows, these algorithms may not be scalable since the single processor may eventually become a bottleneck. In this paper, a parallel approach for interest matching is presented which is suitable to deploy on both shared-memory and distributed-memory multiprocessors. We also provide an analysis of speed-up and efficiency for the simulation results of the parallel algorithms.

1 INTRODUCTION

A distributed virtual environment (DVE) is intended for multiple users to interact in a virtual environment in real-time even though they are at geographically different locations. In recent years, high bandwidth and low latency network environments have facilitated the development of large-scale DVE applications such as Massively Multiplayer Online Games (MMOGs) (Smed et al. 2002) which aim to support millions of participants. Solving the real-time rendering problem is only part of the effort in building such compelling DVEs. Today, one of the major challenges for large-scale DVE development is to provide scalable data distribution mechanisms.

To satisfy the scalability requirement of DVE, a technique called "interest management" has been introduced, which filters unneeded data before delivering updates to the appropriate participant for processing. This typically involves a process referred to as "interest matching", which matches the interests of data senders and receivers and hence determines what data should be sent to the participant. Over the last two decades, interest management and interest matching have been studied extensively in many fields such as military simulations, social DVEs, and MMOGs. Numerous schemes have been proposed which sought to reduce the computational overhead of the matching process. However, existing interest matching algorithms are mainly designed for serial processing which is supposed to be run on a single processor. As the problem size grows, these algorithms may not be scalable since the single processor may eventually become a bottleneck. Furthermore, most of the existing algorithms perform interest matching at discrete time intervals, which might fail to report events between consecutive time-steps of simulation.

In our previous work, we presented the preliminary design of parallel interest matching algorithm for shared-memory multiprocessors (Liu and Theodoropoulos 2009) and distributed-memory systems (Liu and Theodoropoulos 2011). These algorithms facilitate workload sharing by dividing the interest matching process among multiple processors. In this paper, we present a more detailed description of the theoretical concepts of the parallel algorithms. We also provide an analysis of speed-up and efficiency for their

simulation results. Finally, in order to solve the missing event problem, we present a framework to integrate the "space-time" interest matching approach (Liu and Theodoropoulos 2010) with the parallel algorithms.

2 BACKGROUND AND RELATED WORK

Aura-based interest management (Carlsson and Hagsand 1993) use auras to represent the interests of each participant. When auras overlap, a connection between the owners of the auras is established and messages are exchanged through the connection. This approach provides a precise message filtering mechanism; however, computational overhead would be introduced for testing the overlap status for the auras. In the High-Level Architecture (HLA) (DMSO 1998), the Data Distribution Management (DDM) services allow the participants to specify "update regions" and "subscription regions" to represent their interests. These regions are similar to the auras except they must be rectangular and axis-aligned when using in a two- or three-dimensional space.

The interest matching algorithms are designed to solve the "trade-off" between runtime efficiency and filtering precision for aura-based interest management. They have usually been applied on high precision filtering schemes, such as HLA DDM, and provide a way to efficiently reduce the computational overhead of the matching process. In an early paper (Van Hook et al. 1994), the authors pointed out that the matching process of the aura-based approach could be computationally intensive. To solve this problem, Van Hook et al. proposed a crude grid-based filtering approach to cull out many irrelevant entities before a more compute-intensive procedure is carried out for finer discrimination. Morgan et al. (2004) proposed a collision detection algorithm for aura-based interest matching. The algorithm uses aura overlap for determining spatial subdivision. The authors argued that it more accurately reflects the groupings of entities that may be interacting than existing collision detection algorithms. Recently, more robust matching algorithms (Raczy et al. 2005, Liu et al. 2005) based on dimension reduction were proposed, which reduces the multidimensional overlap test to a one-dimensional problem. These algorithms are designed specifically for HLA-compliant systems, and thus adopt the use of rectangular auras (i.e., regions of the HLA).

2.1 The Missing Event Problem

The interest matching algorithms discussed so far focus on enhancing the computational efficiency of the matching process; they might, however, lead to a "missing event". This problem occurs when an entity moves at a high speed such that the distance travelled is sufficiently large per time-step; it might move across another entity without notice.

In general there are two simple general solutions to this problem (Morse and Steinman 1997), but each of them must make a trade-off. The first solution is to specify *large-enough* auras, so that many of the missing events (generated by fast moving auras) could be captured. However, expanding the auras might result in the avatar seeing things he is not supposed to see, resulting in a bandwidth overheard. The second solution is to reduce the time-step of simulation and increase the frequency of performing discrete interest matching. However, this is a very time consuming process and unsuitable for real-time systems such as DVEs.

In (Liu and Theodoropoulos 2010), a space-time interest matching algorithm is presented. This algorithm aims to capture missing events by using Axis-Aligned Swept Volumes (AASVs), which bound the trajectory of auras over the time interval. Although this approach requires additional computational effort, a sorting method is employed to cull out the aura pairs that are unlikely to overlap and thus significantly reducing this overhead.

3 PARALLEL INTEREST MATCHING FOR SHARED-MEMORY MULTIPROCESSORS

This section describes a parallel interest matching algorithm which facilitates parallelism by distributing the workload of the matching process across shared-memory multiprocessors. The algorithm divides the matching process into two phases. In the first phase it employs a spatial data structure called uniform

subdivision to efficiently decompose the virtual space into a number of subdivisions. We define as work unit (WU) the interest matching process within a space subdivision. In the second phase, WUs are distributed across different processors and are processed concurrently.

For the sake of consistency, aura is hereafter referred to as "regions" as per the terminology of HLA DDM.

3.1 Spatial Decomposition

Uniform subdivision is a common spatial data structure which has long been used as a mean of rapid retrieval of geometric information. The idea of using hashing for subdivision directory was first described in an early article (Rabin 1976) and was later discussed more generally in (Bentley and Friedman 1979). This section presents the formal definitions of uniform subdivision, which leads to the discussion in the subsequent sections where they are used for hash indexing and rapid WU distribution.

Formally, the virtual space $\boldsymbol{S}$ can be define as a multidimensional point set that contains all entities in the virtual world. Therefore, all update or subscription regions can be regarded as the subsets of $\boldsymbol{S}$.

Definition 1. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension, for $d = 1,2,...,n$.

$$\boldsymbol{S} = \{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge SMIN_d \leq x_d < SMAX_d, \text{ for } d = 1,2,...,n\}.$$

Alternatively, $\boldsymbol{S}$ can be expressed as the Cartesian product of its one-dimensional boundaries.

Definition 2. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension, for $d = 1,2,...,n$.

$$\boldsymbol{S} = [SMIN_1, SMAX_1) \times [SMIN_2, SMAX_2) \times ... \times [SMIN_n, SMAX_n) = \prod_{d=1}^{n} [SMIN_d, SMAX_d).$$

The hashing approach requires decomposing $\boldsymbol{S}$ into uniform subdivisions. Each subdivision represents a slot in the hash table, which is labelled by a multidimensional hash table index.

Definition 3. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension. The boundary can be uniformly divided into N_d sub-boundaries with unit length L_d, such that

$$L_d = \frac{SMAX_d - SMIN_d}{N_d}$$

$\forall N_d \in \mathbb{Z}^+$, $\forall L_d \in \mathbb{R}^+$, for $d = 1,2,...,n$.

Definition 4. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1,2,...,n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The uniform subdivision $\boldsymbol{Z}$ of $\boldsymbol{S}$ is labelled by a multidimensional hash table index $(z_1, z_2, ..., z_n)$, such that

$$\boldsymbol{Z}(z_1, z_2, ..., z_n) = \{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge SMIN_d + z_d L_d \leq x_d < SMIN_d + (z_d + 1)L_d, \text{ for } d = 1,2,...,n\}$$

for $z_d = 0,1,...,N_d - 1$.

Similar to all axis-aligned point sets, the uniform subdivision can be expressed as the Cartesian product of its one-dimensional boundaries, which is given in **Definition 5**.

Definition 5. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1,2,...,n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The uniform subdivision $\boldsymbol{Z}$ of $\boldsymbol{S}$ can be defined as

$$\begin{aligned}\boldsymbol{Z}(z_1,z_2,...,z_n) &= [SMIN_1+z_1L_1, SMIN_1+(z_1+1)L_1)\times[SMIN_2+z_2L_2, SMIN_2+(z_2+1)L_2)\times... \\ &\quad \times[SMIN_n+z_nL_n, SMIN_n+(z_n+1)L_n) \\ &= \prod_{d=1}^{n}[SMIN_d+z_dL_d, SMIN_d+(z_d+1)L_d)\end{aligned}$$

for $z_d = 0,1,...,N_d-1$.

Theorem 1. Given a set of all hash table indices

$$\boldsymbol{HI} = \{(z_1,z_2,...,z_n) \mid z_d = 0,1,...,N_d-1 \wedge d = 1,2,...,n\}$$

where N_d is the number of subdivisions of space $\boldsymbol{S}$ in d dimension. Then, $\boldsymbol{S}$ can be expressed as the union of all uniform subdivisions, such that

$$\boldsymbol{S} = \bigcup_{k\in\boldsymbol{HI}} \boldsymbol{Z}(k).$$

The proof of **Theorem 1** is beyond the scope of this paper and therefore is omitted.

3.2 First Phase: Hashing

During the simulation, regions are hashed into the hash table. The algorithm uses the coordinate of a region's vertex as a hash key. Given a key k, a hash value $H(k)$ is computed, where $H()$ is the hash function. The hash value is an n-dimensional index which can be matched with the index of a space subdivision, and therefore indicating that which subdivision the vertex lies in. Hence, the regions with hash key k are stored in slot $H(k)$. The hash function is given in **Definition 6**.

Definition 6. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1,2,...,n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The hash function for transforming a key k_d into a hash value is defined as

$$H : \mathbb{R}^n \to \mathbb{Z}^n, H(k_d) = \lfloor \frac{k_d - SMIN_d}{L_d} \rfloor$$

There are two important properties of using a hash table for spatial decomposition. First, hash table collision means that regions in the same slot are potentially overlapped with each other; therefore, further investigation on their overlap status is required. This process is left to the second phase of the algorithm. Second, if a region lies in multiple space subdivisions, it would be hashed into all of them. The algorithm assumes that the size of region is much smaller than a space subdivision. Therefore, a region would exist in at most four slots in the two-dimensional space (at most eight slots in the three-dimensional space). This assumption ensures that the computational complexity of the hashing process would be bounded by a constant.

The hash table is constructed at the initialisation stage. During runtime, the position and size of regions may be frequently modified. Therefore, the algorithm needs to perform rehashing for the regions at every time-step. The complexity of this process is $O(n+m)$ where m is the number of subscription regions and n is the number of update regions.

Applying the parallel algorithm to space-time interest matching is straightforward. All that needs to be changed is to replace regions with AASVs. Therefore, hashing would be performed for the AASVs' vertices instead of the regions'.

3.3 Second Phase: Sorting

After the hashing stage, each slot of the hash table represents a WU which will be distributed across different processors. The algorithm then places the WUs on a task queue. Each processor fetches WUs from the queue and performs interest matching for the corresponding space subdivisions. Since only one processor has the authority to manage each space subdivision, there will be no ambiguous matching result. As discussed in (Dandamudi and Cheng 1995), the task queue approach is desired for task distribution and provides very good load sharing for shared-memory multiprocessor systems. When a processor finishes processing a WU, it would fetch another WU from the task queue immediately unless the queue is empty. Therefore, no processor would be idle until all WUs are fetched. The worst case happens only when all regions or AASVs reside in a single space subdivision. In this situation, a single processor would be responsible for the matching of all of them.

The spatial decomposition approach essentially transforms the large-scale interest matching process into several individual sub-problems. When a WU is being processed, each processor carries out a matching process only for the regions or AASVs within the WU. The matching process employs a sorting algorithm presented in (Liu and Theodoropoulos 2009) and (Liu and Theodoropoulos 2010) for discrete and space-time interest matching, respectively. This sorting algorithm makes use of the concept of dimension reduction and is theoretically the most efficient serial algorithm for interest matching.

4 PARALLEL INTEREST MATCHING FOR DISTRIBUTED-MEMORY SYSTEMS

This section presents an extension of the parallel algorithm proposed in the previous section (which is hereafter referred to as the "shared-memory algorithm"). The new parallel algorithm (which is hereafter referred to as the "distributed-memory algorithm") can be run on a cluster of computers that enables them to work simultaneously and thus increasing the overall runtime efficiency of the matching process. The distributed-memory algorithm is suitable to apply for both discrete and space-time interest matching. For the sake of simplicity, our discussion is first focused on the former.

4.1 Spatial Decomposition

Similar to the shared-memory algorithms, the distributed-memory algorithms employ the uniform subdivision approach to efficiently decompose the virtual world into a number of static space subdivisions. However, in contrast to shared-memory multiprocessors, all processors (which are hereafter referred to as *nodes*) participating in the matching process have their own private memory. Therefore, the task queue is no longer available for data sharing. As a result, each node must maintain a WU-node map, which contains the information of the space subdivisions that are currently being processed by the nodes.

Figure 1 illustrates an example of space subdivisions in two-dimensional space. In the figure, $Node_A$ is responsible for WU (0,2), (1,1), (1,2), and (2,2); $Node_B$ is responsible for WU (0,0 and (0,1); $Node_C$ is responsible for WU (1,0), (2,0), and (2,1).

At the initialisation stage, the WUs are evenly divided between the working nodes. The regions are distributed to different nodes according to the space subdivisions they reside in. If a region lies in multiple space subdivisions that are owned by different nodes, it would be distributed to all of them. Again, we assume that the size of region is much smaller than a space subdivision. Therefore, a region would exist in at most four nodes for the two-dimensional space (at most eight nodes for the three-dimensional space).

The position and size of a region may be modified dynamically during simulation. Whenever a region is modified, its owner node is responsible for determining whether the region in question is entering a space subdivision that is owned by another node. This is done by hashing the vertices of the region with the hash function given in **Definition 6**. The hash value is an n-dimensional key which can be matched with the index of space subdivision, and therefore indicating which subdivision the vertex lies in.

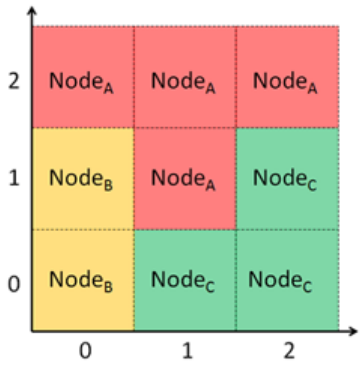

Figure 1: Space Decomposition in 2D.

4.2 Sorting and Matching

At the matching phase, each node processes all of the WUs that are assigned to it. If the WU has more than one region (i.e., hash table collision), A sorting process similar to the one presented in the previous section would be applied to determine the overlap status of the regions.

The shared-memory algorithm proposed in the previous section requires each WU maintains one endpoint list per dimension at the sorting stage. In contrast to this approach, in the distributed-memory algorithm the endpoint lists of different WUs are combined, as long as the WUs are owned by the same node. As a result, each node only maintains one endpoint list per dimension. Combining the endpoint lists does not affect the order of the regions' vertices, due to the fact that the regions are all in the same space.

4.3 Load-Balancing

Due to the arbitrariness of the entities' movement pattern, the workload of the nodes may become uneven during runtime. Since the shared task queue is no longer available as a mean of load-balancing, a load-balancing scheme should be carried out to redistribute the workload, in order to maximise the utilisation of computational resources.

The workload of interest matching is defined by the elapsed time to process a WU. After the matching process, each node broadcasts its workload to all other nodes. The node with the heaviest load then carries out a load-balancing algorithm to redistribute a portion of its workload to a neighbour node. The term "neighbour node" is defined as the owner of an adjacent space subdivision. For example, in Figure 1, $Node_C$ is a neighbour node of WU (0,0).

Algorithm 1 is employed to balance the workload among the nodes. This process is performed in an adaptive manner that only the ownership of one WU would be transferred at each time-step.

The reason for only transferring the ownership of WU to a neighbour node is to reduce the communication overhead. When a region moves across a border that the WUs on both sides of the border are owned by different nodes, the two nodes should exchange messages for the region's migration. If the algorithm transfers the ownership of WU to arbitrary nodes, it might increase the chance of creating isolated WUs (i.e., all adjacent WUs are owned by different nodes), and therefore increase the chance of region migration. On the other hand, if the WU is transferred to a neighbour node, after the transfer there would be at least one adjacent WU is owned by the same node. This reduces the communication overhead of border crossing.

5 PERFORMANCE EVALUATION

This section presents an evaluation of the proposed parallel interest matching algorithm. Several sets of experiments were carried out to compare the performance of four approaches, namely:

Algorithm 1: Workload Redistribution Algorithm (The Least Loaded Neighbour.

```
Data: WU: a list of WU owned by the current node
Data: λ: the threshold of workload difference
1  begin
2      Sort WU in descending order by the number of regions they contain;
3      for i ← 1 to Size(WU) − 1 do
4          if WU[i].HasNeighbourNode() then
5              node ← WU[i].LeastLoadNeighbour();
6              if (Load − leastLoadedNode.Load)/Load > λ then
7                  TransferOwnership(WU[i], node);
8              end
9          end
10     end
11 end
```

1. Discrete interest matching by sorting algorithm (SDIM)
2. Space-time interest matching by sorting algorithm (SCIM)
3. Discrete interest matching by distributed-memory algorithm (PDIM-DM)
4. Space-time interest matching by distributed-memory algorithm (PCIM-DM)
5. Discrete interest matching by shared-memory algorithm (PDIM-SM)
6. Space-time interest matching by shared-memory algorithm (PCIM-SM)

The SDIM approach is the efficient sorting approach presented in (Liu and Theodoropoulos 2009). It was chosen as a comparison target because it is theoretically the fastest (linear time in the general case) algorithm among all serial algorithms. The SCIM approach is an implementation of the efficient space-time interest matching presented in (Liu and Theodoropoulos 2010). It performs sorting to efficiently cull out the region pairs that are unlikely to overlap with each other, and carries out space-time matching for the remaining pairs. PDIM-SM and PCIM-SM are the parallel algorithms presented in Section 3, which exploit parallelism by distributing the workload across multiple processors. The PDIM-SM approach is designed for discrete interest matching, while the PCIM-SM approach performs space-time interest matching similar to the SCIM approach. PDIM-DM and PCIM-DM are the parallel algorithms presented in Section 4, which perform interest matching in a parallel manner where workload is shared by a cluster of nodes. The PDIM-DM approach is designed for discrete interest matching, while the PCIM-DM approach performs space-time interest matching similar to the SCIM approach.

This evaluation only focuses on the runtime efficiency of the interest matching approaches. The filtering precision of all six approaches is similar due to the fact that they all adopt an aura-based scheme. Therefore, performing further experiments to evaluate their filtering precision is unnecessary. Furthermore, the space-time algorithms' ability to capture missing events has already been evaluated in (Liu and Theodoropoulos 2010). It has been shown experimentally that the space-time algorithms have better event-capturing ability than the discrete algorithms.

5.1 Implementation and Experimental Set-up

In order to evaluate the performance of the interest matching approaches, we implemented a simulation using a World War II dogfight scenario (aerial combat between fighter aircrafts). In this scenario, the virtual entities (aircrafts) are equally divided into two teams, which engage each other in an aerial combat in a three-dimensional scene. The update and subscription regions that are associated with the aircraft are modified dynamically during the simulation.

Table 1: Experimental Set-up.

Scene Size	40km x 40km x 20km.
Entity Speed	The maximum speed (referred to as MS) of the aircraft was set to 500km/h in all experiments.
Entity Acceleration	The acceleration of the aircraft is set to $5m/s^2$. The gravitational acceleration is set to $9.8m/s^2$.
Entity Distribution	Initially, the two teams are placed at two opposite corners of the scene.
Subscription Region	One subscription region is associated with each of the aircraft. The size of the subscription region is approximately the line-of-sight of the pilot, which is set to 3km x 3km x 3km with the aircraft at the centre.
Update Region	One update region is associated with each of the aircraft. The size of the update region is approximately the size of the aircraft, which is set to 15m x 15m x 15m.
Entity Movement	Entity movements are based on the following four modes: • Search: The aircraft gradually reduces its speed to half of its maximum speed and searches for a new target. If a target appears, it then changes to engage mode; otherwise, it turns to a random new direction for every 2 minutes. • Engage: The aircraft accelerates to the maximum speed and chases its target for 2 minutes. It then changes to cool mode. • Cool: The aircraft gradually reduces its speed to half of its maximum speed and then flies for 1 minutes. It then changes to the search mode. In addition, the aircraft cannot engage any target in this mode but can be engaged by an enemy. • Evade: If an aircraft is being engaged, it accelerates to the maximum speed and turns to a new random direction for every 5 seconds until it is no longer being engaged. It then changes to the cool mode.
Execution Time Measurement	Average execution time of the matching algorithms was measured over 10,000 time-steps.

For the sake of simplification, no aircraft is destroyed during the simulation. The simulator and all six algorithms were implemented in C++. The experiments of the shared-memory algorithms were run on a workstation, which consists of two Intel Xeon E5634 2.4Ghz 6-core CPUs with 48GB main memory. For the distributed-memory algorithms, message communication was constructed based on MPI protocols, such as *MPI_Bcast*(), *MPI_Send*(), and *MPI_Recv*(); all processes were synchronised by the *MPI_Barrier*() call, which is a simple lock-step synchronisation protocol. The experiments of the distributed-memory algorithms were executed on the eScience Cluster at the Midland e-Science Centre. Each worker node has an Intel Xeon 3GHz processor with 2GB main memory. A Myrinet backplane is used to give 2+2Gbps programmable interconnection between the worker nodes.

5.2 Runtime Efficiency of Shared-Memory Algorithms (Number of Processor varies)

This set of experiments compares the runtime efficiency of SDIM, SCIM, PDIM-SM, and PCIM-SM when running on different number of processors. In the case of number of required processors less than 12, some of the physical cores of the two Intel E5634s were disabled. The number of working threads was equal to the number of available cores. The number of entities was set to the constant value of 1000.

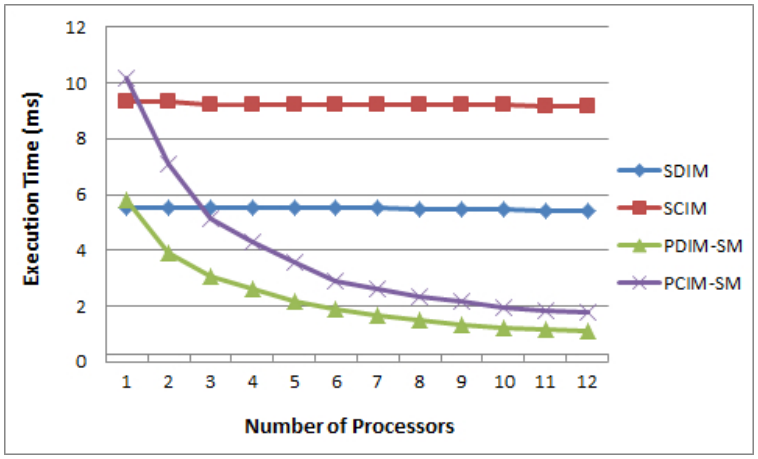

Figure 2: Execution Time of Shared-Memory Algorithms (Number of Processors varies).

Figure 2 shows the execution time of the four approaches. It is not difficult to see that the parallel algorithms (PCIM-SM and PDIM-SM) scale much better than the serial algorithms (SCIM and SDIM). Since SCIM and SDIM are designed for serial processing, the execution time of these algorithms did not change significantly when the number of active processors increased. The execution time of parallel algorithms (PDIM-SM and PCIM-SM), on the other hand, decreased gradually with increase in the number of active processors; this suggests that the proposed algorithm is scalable when running on a shared-memory multiprocessor machine.

Furthermore, the space-time algorithms require more computational effort than the discrete algorithms under the same matching approach (i.e., sorting or parallelism). The discrepancy is due to the overhead introduced by the divide-and-conquer algorithm as well as the computation of AASVs.

5.3 Execution time of Distributed-Memory Algorithms (Number of Processors varies)

The second set of experiments compares the runtime efficiency of SDIM, SCIM, PDIM-DM, and PCIM-DM when running on different number of processors (1-10). The number of entities was again set to constant (1000).

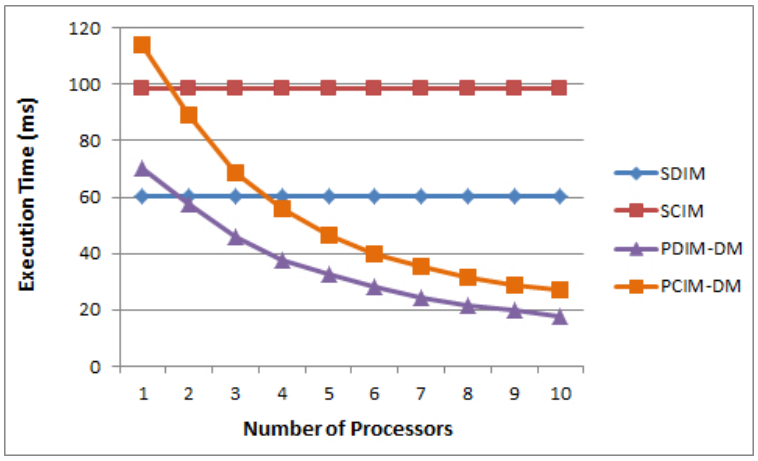

Figure 3: Execution Time of Distributed-Memory Algorithms (Number of Processors varies).

The results are shown in Figure 3. We can see from that graph that the execution time of the serial algorithms (SCIM and SDIM) is constant as they are designed for serial processing and are included only for reference. The execution time of the two parallel algorithms (PDIM-DM and PCIM-DM), on the other hand, decreased gradually with increase in the number of active processors. Therefore, this suggests that the two parallel algorithms, even with communication overhead, are more computationally scalable when running on a cluster of nodes. Nevertheless, similar to the shared-memory algorithms described in the previous subsection, the performance gain of PDIM-DM and PCIM-DM is not proportional to the number of processors. Their actual scalability of parallelism should be obtained through speed-up and efficiency analysis.

5.4 Analysis of Speed-up and Efficiency

As Lee (1980) has observed, a parallel algorithm rarely attain its maximum efficiency, this is due to both logical and physical constraints. For example, logical constraints may include intrinsic data-dependencies, control dependencies and operator precedences in the parallel algorithm, which force a serial chain of execution amongst the dependent operations, and thus limit the number of operations which may be executed in parallel. Physical constraints may include the control restrictions on the different types of operations which may be executed simultaneously, and the delays due to the communication and competition amongst the interacting components in the computer. Lee therefore defined certain measures to compare the effectiveness of various parallel algorithms, such as speed-up, efficiency, redundancy, utilisation, and quality. The first two measures are widely used today for analysis of parallelism (Parhami 1999).

In Lee's paper, the term "speed-up" refers to how much faster a parallel algorithm is compared to a corresponding serial algorithm, such that

$$S(n,p) = \frac{T(n,1)}{T(n,p)} \tag{1}$$

where n is the problem size, p is the number of processors, $T(n,1)$ is the execution time of the serial algorithm, and $T(n,p)$ is the execution time of the parallel algorithm with p processors.

An ideal speed-up (or linear speed-up) is obtained when $S(n,p) = p$. When running an algorithm with ideal speed-up, doubling the number of processors doubles the speed. Therefore it is considered extremely good scalability.

The term "efficiency" represents how well-utilised the processors are in solving the problem, compared to how much effort is wasted in communication and synchronisation. It is defined as

$$E(n,p) = \frac{S(n,p)}{p} = \frac{T(n,1)}{pT(n,p)}. \tag{2}$$

Note that a parallel algorithm with ideal speed-up and algorithms running on a single processor have an efficiency of 1.

The experimental results presented in this section allow us to analyse the speed-up and efficiency of proposed parallel algorithms based on the calculation of these two measures. Let T_C, S_C and E_C denote the execution time, speed-up and efficiency of PCIM-SM, respectively; and let T_D, S_D and E_D denote the execution time, speed-up and efficiency of PDIM-SM, respectively. The results of the analysis are given in Table 2. Note that the problem size n is set to constant (1000).

The results show that the efficiency of both PDIM-SM and PCIM-SM drops gradually when more processors are involved in the matching process. In particular, we can see that when all 12 processors are used in the experiments, the efficiency of both algorithms drops below 50%. Therefore, even if we add more processors to the experiments, the performance gain may not be significant.

Based on the same methodology, we carried out a similar analysis for the distributed-memory algorithms. Let T_C, S_C and E_C denote the execution time, speed-up and efficiency of PCIM-DM, respectively; and let

Table 2: Speed-up and Efficiency of the Shared-Memory Algorithms.

p	$T_D(n,p)$	$S_D(n,p)$	$E_D(n,p)$	$T_C(n,p)$	$S_C(n,p)$	$E_C(n,p)$
1	5.784	1	100%	10.142	1	100%
2	3.874	1.493	74.65%	7.102	1.428	71.40%
3	3.041	1.902	63.40%	5.127	1.978	65.94%
4	2.589	2.234	55.85%	4.318	2.349	58.72%
5	2.142	2.700	54.01%	3.542	2.863	57.27%
6	1.876	3.083	51.39%	2.894	3.504	58.41%
7	1.685	3.433	49.03%	2.601	3.899	55.70%
8	1.486	3.892	48.65%	2.356	4.305	53.81%
9	1.325	4.365	48.50%	2.176	4.661	51.79%
10	1.225	4.723	47.23%	1.952	5.196	51.96%
11	1.152	5.023	45.64%	1.844	5.500	50.00%
12	1.129	5.123	42.69%	1.784	5.685	47.37%

T_D, S_D and E_D denote the execution time, speed-up and efficiency of PDIM-DM, respectively. The results of the analysis are given in Table 3. The problem size n is again set to constant (1000).

Table 3: Speed-up and Efficiency of the Distributed-Memory Algorithms.

p	$T_D(n,p)$	$S_D(n,p)$	$E_D(n,p)$	$T_C(n,p)$	$S_C(n,p)$	$E_C(n,p)$
1	70.123	1	100%	114.235	1	100%
2	57.431	1.221	61.08%	88.899	1.285	64.25%
3	45.862	1.529	50.98%	68.692	1.663	55.43%
4	37.491	1.87	46.76%	55.779	2.048	51.21%
5	32.434	2.162	43.24%	46.193	2.473	49.46%
6	28.196	2.487	41.45%	39.817	2.869	47.82%
7	24.481	2.864	40.92%	35.29	3.237	46.24%
8	21.783	3.219	40.24%	31.609	3.614	45.18%
9	19.755	3.55	39.44%	28.898	3.953	43.92%
10	17.861	3.926	39.26%	26.91	4.245	42.45%

The analysis shows that the efficiency of both PDIM-DM and PCIM-DM drops gradually to about 40% when the number of processors is increased to 10, and the change in efficiency becomes less apparant when more processors are involved in the experiments. If we compare distributed-memory algorithms with the shared-memory algorithms, we can see that the former are about 10% less efficient than the latter when the same number of processors are involved in the simulations. It is important to note that, however, this comparison is not entirely fair since the processing power, cache size, and memory bandwidth of the two testbeds (Intel E5645 and eScience Grid) are significantly different.

6 CONCLUSIONS AND FUTURE WORK

This paper presents a parallel interest matching approach which can be applied on both shared-memory and distributed-memory multiprocessors. As reviewed in Section 2, existing interest matching algorithms are unsuitable to deploy on parallel systems. The proposed algorithm, on the other hand, facilitates workload sharing by dividing a large-scale interest matching problem into several smaller sub-problems and distributing them across multiple processors. Experimental evidence, based on a World War II dogfight scenario, has demonstrated that the parallel algorithms are more computationally efficient than the existing algorithms when running on a multiprocessor machine. Moreover, the parallel algorithms can be easily

combined with the space-time approach presented in our previous work, in order to capture the missing events between discrete time intervals. Although the space-time approach requires additional computational effort, running it within the parallel framework can further minimise its computational overhead.

Our future work will concentrate on testing and comparing the performance the proposed algorithms under different entity behaviors, number of nodes, and occupation density.

7 ACKNOWLEDGMENT

This work was supported, in part, by Lero - the Irish Software Engineering Research Centre (www.lero.ie).

REFERENCES

Bentley, J. L., and J. H. Friedman. 1979, December. "Data Structures for Range Searching". *ACM Computing Surveys* 11 (4): 397–409.

Carlsson, C., and O. Hagsand. 1993. "DIVE - A platform for multi-user virtual environments". *Computers & Graphics* 17 (6): 663–669.

Dandamudi, S. P., and P. S. P. Cheng. 1995. "A Hierarchical Task Queue Organization for Shared-Memory Multiprocessor Systems". *IEEE Transactions on Parallel and Distributed Systems* 6 (1): 1–16.

DMSO 1998. "High Level Architecture Interface Specification Version 1.3".

Lee, R. B.-L. 1980, August. "Empirical Results on the Speed, Efficiency, Redundancy and Quality of Parallel Computations". In *Proceedings of the 1980 International Conference on Parallel Processing*: IEEE Computer Society.

Liu, E. S., and G. Theodoropoulos. 2010, May. "A Continuous Matching Algorithm for Interest Management in Distributed Virtual Environments". In *Proceedings of the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2010)*.

Liu, E. S., and G. K. Theodoropoulos. 2009, October. "An Approach for Parallel Interest Matching in Distributed Virtual Environments". In *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009)*.

Liu, E. S., and G. K. Theodoropoulos. 2011, September. "A Parallel Interest Matching Algorithm for Distributed-Memory Systems". In *Proceedings of the 15th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2011)*.

Liu, E. S., M. K. Yip, and G. Yu. 2005. "Scalable Interest Management for Multidimensional Routing Space". In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*.

Morgan, G., K. Storey, and F. Lu. 2004. "Expanding Spheres: A Collision Detection Algorithm for Interest Management in Networked Games". In *Proceedings of the Entertainment Computing - ICEC 2004*.

Morse, K. L., and J. S. Steinman. 1997, March. "DATA DISTRIBUTION MANAGEMENT IN THE HLA - Multidimensional Regions and Physically Correct Filtering". In *Proceedings of the 1997 Spring Simulation Interoperability Workshop (SIW)*, 343–352.

Parhami, B. 1999. *Introduction to Parallel Processing: Algorithms and Architectures*. 1 ed. Springer.

Rabin, M. 1976. "Probabilistic algorithms". In *Proceedings of the Symposium on New Directions and Recent Results in Algorithms and Complexity*, 21–39. New York: Academic Press Inc.

Raczy, C., G. Tan, and J. Yu. 2005. "A Sort-Based DDM Matching Algorithm for HLA". *ACM Transactions on Modeling and Computer Simulation* 15 (1): 14–38.

Smed, J., T. Kaukoranta, and H. Hakonen. 2002, April. "A Review on Networking and Multiplayer Computer Games". Technical Report 454, Turku Centre for Computer Science.

Van Hook, D. J., S. J. Rak, and J. O. Calvin. 1994. "Approaches to Relevance Filtering". In *Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations*, 26–30.

AUTHOR BIOGRAPHIES

ELVIS S. LIU is a Research Fellow of Irish Research Council and IBM Research, Ireland. His current research is focused on simulation technologies for Exascale high-performance computing systems. He received a B.Sc. from the University of Hong Kong, an M.Sc. from the University of Newcastle (UK), and a Ph.D. from the University of Birmingham (UK). His email address is <elvisliu@ie.ibm.com>.

GEORGIOS K. THEODOROPOULOS is currently the Executive Director of the Institute of Advanced Research Computing at Durham University, UK where he holds a personal Chair in Computer Science. His research interests include distributed simulation and virtual environments, complex systems, distributed systems, info-symbiotic systems, Grid and Cloud computing. His e-mail address is <theogeorgios@gmail.com>.