Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

A FAST PARALLEL MATCHING ALGORITHM FOR CONTINUOUS INTEREST MANAGEMENT

Elvis S. Liu
Georgios K. Theodoropoulos

School of Computer Science
University of Birmingham
Edgbaston, Birmingham, B15 2TT, United Kingdom

ABSTRACT

In recent years, the scale of distributed virtual environments (DVEs) has grown rapidly in terms of number of participants and virtual entities. Many DVEs employ interest management schemes to reduce bandwidth consumption and thus enhance the scalability of the system. Most of the existing interest management approaches, however, have a fundamental disadvantage - they perform interest matching at discrete time intervals. As a result, they would fail to report events between consecutive time-steps of simulation which leads to incorrect simulations. In this paper, we present a new algorithm for interest matching which aims to capture missing events between discrete time-steps. This algorithm facilitates parallelism by distributing the workload of matching process across multiple processors. Since it is increasingly common to deploy commercial DVE applications on shared-memory multiprocessor machines, using the parallel algorithm for these applications would be more suitable than the existing serial algorithms.

1 INTRODUCTION

A distributed virtual environment (DVE) can be characterised as a type of distributed simulations that allows multiple users to interact with each other in real-time even though they are at geographically different locations. In recent years, the scale of DVE has grown rapidly in terms of number of users and virtual entities. Using scalable data distribution mechanisms becomes one of the major requirements of the DVE system. Therefore, data filtering schemes, or interest management, have been developed to satisfy this requirement.

The basic idea of interest management is to have all participants receive only the data that are of interest to them. The interest matching process is essential for most of the interest management schemes which determines what data should be sent to the participants as well as what data should be filtered. The following are the major design requirements of interest matching algorithms:

- *Filtering Precision*: As the number of participants and virtual entities increase, state broadcasting could consume significant network resources. The algorithm should provide precise message filtering mechanisms which ensure the participants receive the minimal set of data that are of interest to them.
- *Runtime Efficiency*: If the cost in terms of computational resources of interest matching is too high, it would be unsuitable for real-time applications such as multiplayer online games (MOGs) (Smed et al. 2002) for which runtime performance is important. Therefore, the algorithm should provide a way to efficiently minimise the computational overhead of the matching process.
- *Capturing Ability*: Since DVE participants rely on the interest matching process to determine what events (i.e., messages) to receive, if the algorithm fails to report some of the events, the participants would most likely render incorrect scenes or perform incorrect simulations. Therefore the algorithm should have the ability to capture most of the events generated by the DVE.

Complex filtering mechanisms can precisely filter irrelevant messages, but they usually introduce significant computational overhead. In other words, there is often a trade-off between filtering precision and runtime efficiency. Over the years, a number of efficient interest matching algorithms were proposed which sought to reduce the computational overhead and, at the same time, to maintain the high precision of message filtering. These algorithms have been shown to meet their runtime performance requirements; however, they have a fundamental disadvantage - they perform interest matching at discrete time intervals. This might lead to missing events in large-scale DVEs that contain virtual entities of greatly varying types. For example, if an entity moves around the virtual space at a high speed such that the distance travelled is sufficiently large per time-step, it might move across a small entity without notice. In other words, the events between two consecutive time-steps are ignored by these algorithms. If DVE participants cannot receive the missing events, they would most likely perform incorrect rendering or simulations. The illusion of presence in DVE would break down seriously when this happens.

In (Liu and Theodoropoulos 2010) we presented a new algorithm for continuous interest matching and demonstrated its effectiveness in terms of capturing ability. The new algorithm aims to capture missing events between discrete time-steps of simulation. In this paper we are concerned with enhancing the runtime efficiency of this algorithm through parallelisation. Leveraging on our previous work (Liu and Theodoropoulos 2009), we present an extension of our continuous interest matching algorithm that can be executed on shared-memory multiprocessors to enhance the overall runtime efficiency of the matching process.

The remainder of this paper is organised as follows. Section 2 briefly reviews the background and related work of interest management schemes and interest matching algorithms. Section 3 describes the missing event problem and the existing solutions. For the sake of completeness, Section 4 presents an outline of the pairwise space-time interest matching algorithm introduced in (Liu and Theodoropoulos 2010). Section 5 discusses how the algorithm of Section 4 is exploited to develop a parallel algorithm for space-time interest matching. Section 6 evaluates the performance of our approach and existing approaches by experimental results. Finally, Section 7 concludes this paper and briefly describes our future work.

2 RELATED WORK

This section briefly reviews the related work of interest management schemes and interest matching algorithms.

2.1 Interest Management Schemes

Interest management has been studied extensively in many fields such as military simulations, DVEs, and MOGs. Numerous schemes have been proposed throughout the years, and can be classified into three main categories: zone-based (sometimes called cell-based), aura-based and class-based schemes.

The zone-based schemes (Macedonia et al. 1995, Barrus et al. 1996) partition the virtual world into a number of zones, where participants are connected to some of the zones and receive only the updates that are generated from them. However, irrelevant data would still be sent to the participants if the size of zones is not properly chosen. The aura-based schemes (Greenhalgh and Benford 1995) 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 much more precise message filtering mechanism than the zone-based approaches; however, more computational effort is required for testing the overlap status for the auras. The class-based schemes (Fujimoto 2000) allow participants to subscribe to attributes of an object class, indicating they wish to receive notification whenever that attribute of any object instance of that class is modified.

Some systems such as the latest version of DIVE (Benford, Bowers, Fahlen, and Greenhalgh 1994) and MASSIVE-3 (Purbrick and Greenhalgh 2000) attempt to combine different zone-based and aura-based schemes for fine-grained filtering as well as reducing computational overheads. The High-Level Architecture (HLA) (DMSO 1998) supports both class-based and value-based filtering. The Data Distribution Management (DDM) services of the HLA provide a flexible mechanism for publishing, and subscribing to, areas of interest though multidimensional routing space. Participants (federates) can specify what data they are going to publish or subscribe to by creating update regions and subscription regions. In the HLA interest matching process, an object is said to be of interest to a federate if and only if there is at least one of the object's attributes is subscribed to by the federate; and at least one update region associated with the object overlaps at least one subscription region of the federate.

2.2 Interest Matching Algorithms

Most of the existing interest matching algorithms are designed to solve the "trade-off" between runtime efficiency and filtering precision. They usually adopt aura-based filtering which ensures the participants receive the minimal set of data that are of interest to them. In addition, they provide a way to efficiently reduce the computational overhead of the matching process.

Van Hook, Rak, and Calvin (1994) pointed out in an early paper that the matching process of the object-based approach (which is similar to the aura-based approach) could be computationally intensive. To solve this problem, one might use crude grid-based filtering to cull out many irrelevant entities before a more compute-intensive procedure is carried out for finer discrimination. In a later paper, Van Hook, Rak, and Calvin (1997) proposed a clustering approach by multidimensional binary trees to reduce the computational cost of the matching process. Morgan, Storey, and Lu (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 objects that may be interacting than existing collision detection algorithms and provided performance figures to show that it is scalable. Raczy, Tan, and Yu (2005) proposed a sort-based DDM matching algorithm. The basic idea is to use the dimension reduction approach (Cohen et al. 1995) which reduces the multidimensional overlap test to a one-dimensional problem. Specifically, the algorithm projects the regions on each axis and uses heap-sort to sort the projections in order to find out the overlap information. The computational complexity of such sorting is $O(nlogn)$. Pan et al. (2007) also proposed a sort-based matching algorithm and argued that it has better storage and computational scalability than Raczy et al.'s algorithm in many cases. Liu, Yip, and Yu (2005) and Liu, Yip, and Yu (2006) proposed a HLA-compatible matching algorithm based on caching. It also exploits the concept of dimension reduction; however, instead of finding the overlap information every time, it caches the matching results of the previous time-steps. In environments where entities make relatively small movements, interest matching can be processed in linear time.

These are by far the fastest interest matching algorithms. However, they can be considered as "discrete algorithms" since they perform matching at discrete time intervals and thus lack the ability to capture events between time-steps.

3 THE MISSING EVENT PROBLEM

The "Missing event" 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.

As described in (Liu and Theodoropoulos 2010), there are two simple solutions for this problem. The first solution is to specify *large* auras, which can capture more missing events but with a drawback of generating irrelevant messages. Another solution is to reduce the time-step of simulation and increase the frequency of discrete interest matching. This is however a very time consuming process. Furthermore, additional overhead would be introduced when the frequency of aura update is increased. Morgan et al. proposed a predictive approach (Morgan and Lu 2003) to solve this problem, which is a message exchange policy based on predictive modeling of entity movements. It aims to avoid missing interactions by varying message exchange between nodes based on the likelihood that entities will influence each other in the near future. The performance of predictive interest management is, however, unclear since the authors have not published the evaluation (in terms of runtime efficiency and the ability of capturing missing events) of this approach. In our previous work (Liu and Theodoropoulos 2010), we presented a continuous interest matching algorithm which aims to capture missing events by performing space-time overlap test for aura pairs. Although this approach requires additional computational effort, we employ a sorting method to cull out the aura pairs that are unlikely to overlap and thus significantly reducing this overhead.

In this paper, we present a new algorithm which further enhances the runtime efficiency of continuous interest matching by exploiting parallelism. The design of the proposed algorithm is based on our two-phase parallel interest matching approach (Liu and Theodoropoulos 2009). We expect that the runtime efficiency of the proposed algorithm will be significantly enhanced when running on shared-memory multiprocessor computers. In the next section, we review the pairwise space-time matching algorithm. This algorithm forms part of our parallel algorithm for multi-region space-time matching which is described in Section 5.

4 PAIRWISE SPACE-TIME INTEREST MATCHING

The pairwise space-time interest matching approach is designed for aura-based filtering and aims to find out the interaction of a pair of auras between two consecutive time-steps of simulation. This approach uses swept volumes (Abdel-Malek, Blackmore, and Joy 2006) to bound the trajectory of the auras over each time interval. If the swept volumes overlap, it then carries out a divide-and-conquer algorithm to efficiently find out whether the two auras in question actually overlap at a certain time.

For consistency and simplicity, we use the terminology of HLA such as "extent", "subscription region" and "update region" in the rest of this paper.

4.1 Swept Volume

A swept volume is a bounding volume that bounds the motion of an arbitrary virtual entity along an arbitrary path over a time interval. The use of swept volume has been studied extensively in computer graphics and robotics. Some formal definitions of swept volume can be found in an early paper (Cameron 1985).

Exact computation of swept volumes could be time consuming, especially when entities undergo rotations and are geometrically complex. However, this can be greatly simplified if we apply swept volumes on DDM-based interest matching, since all regions in DDM are axis-aligned and do not rotate. In the rest of this subsection, we extend and modify the formal definitions of (Cameron 1985), in order to apply them on axis-aligned regions.

Formally, a region can be regarded as a set of points $\boldsymbol{R}$. We assume the existence of a polynomial function $\vec{\boldsymbol{\tau}}(t)$, which describes the motion of $\boldsymbol{R}$ such that it transforms a point $\vec{\boldsymbol{q}} \in \boldsymbol{R}$ to the position $\vec{\boldsymbol{p}}$ at time t. Hence, we define $\boldsymbol{A}_t$ as a set of points occupied by $\boldsymbol{R}$ at any time t:

$$\boldsymbol{A}_t = \{\vec{\boldsymbol{p}} \mid (\exists \vec{\boldsymbol{q}})(\vec{\boldsymbol{q}} \in \boldsymbol{R}, \vec{\boldsymbol{p}} = \vec{\boldsymbol{q}} + \vec{\boldsymbol{\tau}}(t))\} \quad (1)$$

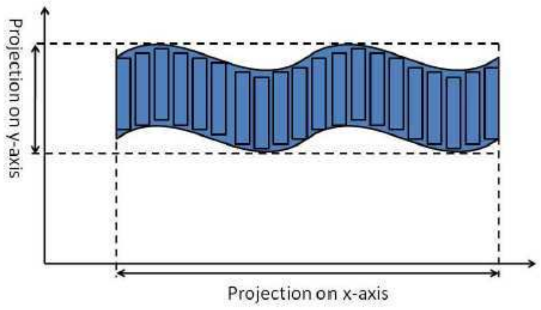

Figure 1: Orthogonal Projections of Swept Volume

Let $\boldsymbol{SV}$ denote the swept volume generated by the motion of $\boldsymbol{R}$ over the time interval $[a,b]$. $\boldsymbol{SV}$ can be regarded as the union of all $\boldsymbol{A}_t$ within $[a,b]$:

$$\boldsymbol{SV}([a,b]) = \bigcup_{t\in[a,b]} \boldsymbol{A}_t = \{\vec{\boldsymbol{p}} \mid \vec{\boldsymbol{p}} \in \boldsymbol{A}_t, \exists t \in [a,b]\} \tag{2}$$

$\boldsymbol{SV}$ can also be expressed by the union of smaller swept volumes:

$$\boldsymbol{SV}([a,b]) = \bigcup_{i=1}^{n} \boldsymbol{SV}(I_i) \tag{3}$$

where I_i represents the subinterval of $[a,b]$, such that I_i = $[t_{i-1},t_i]$ for $a = t_0 < t_i < ... < t_n = b$.

In order to find out whether two regions, $\boldsymbol{R}_U$ and $\boldsymbol{R}_S$, overlap within the time interval $[a,b]$, we need to perform a space-time intersection test for the sets of points which each region occupies along its path of motion. i.e., the two regions overlap within $[a,b]$ if and only if there exists a point $\vec{\boldsymbol{p}}$ that lies within both sets of points occupied by $\boldsymbol{R}_U$ and $\boldsymbol{R}_S$, respectively, at certain time $t \in [a,b]$:

$$\exists(\vec{\boldsymbol{p}},t)(\vec{\boldsymbol{p}} \in \boldsymbol{A}_t^U, \vec{\boldsymbol{p}} \in \boldsymbol{A}_t^S, t \in [a,b]) \Leftrightarrow \bigcup_{t\in[a,b]} (\boldsymbol{A}_t^U \cap \boldsymbol{A}_t^S) \neq \emptyset \tag{4}$$

where $\boldsymbol{A}_t^U$ and $\boldsymbol{A}_t^S$ are the sets of points occupied by $\boldsymbol{R}_U$ and $\boldsymbol{R}_S$, respectively, at time t.

4.2 Approximate Solution of Overlap Test

Since (4) cannot be easily computed numerically, an approximate solution for the overlap test is used. According to (3), $\boldsymbol{SV}$ can be split into some smaller swept volumes by splitting $[a,b]$ into some subintervals I_i. Hence $\boldsymbol{A}_t^U \cap \boldsymbol{A}_t^S$ can be approximated by $\boldsymbol{SV}_U(I_i) \cap \boldsymbol{SV}_S(I_i)$ if I_i are small, such that:

$$\bigcup_{t\in[a,b]} (\boldsymbol{A}_t^U \cap \boldsymbol{A}_t^S) \subseteq \bigcup_{i=1}^{n} (\boldsymbol{SV}_U(I_i) \cap \boldsymbol{SV}_S(I_i)) \tag{5}$$

Furthermore, the swept volume itself can be approximated by an axis-aligned swept volume (AASV) formed by its orthogonal projections. Figure 1 illustrates the orthogonal projections of a swept volume in 2D space.

Although it is a loose bound, using AASV approximation has two benefits. Firstly, computing AASV is much faster than computing the actual swept volume. Moreover, the AASV can be easily integrated with the *dimension reduction* approach, which will be discussed in the next section.

Without loss of generality, we let $x(t)$ be the x-component of $\vec{\boldsymbol{\tau}}(t)$, which describes the motion of $\boldsymbol{R}$'s x coordinate. To calculate the x-projection of the AASV of $\boldsymbol{R}$, we must find the greatest and least values of $x(t)$ on $[a,b]$. This requires finding all the relative maxima and relative minima there and then compare all these values together with $x(a)$ and $x(b)$. Finally, choose the greatest and least values among them.

Let $\boldsymbol{AV}$ denotes the AASV formed by the orthogonal projections of a swept volume $\boldsymbol{SV}$, such that $\boldsymbol{SV} \subseteq \boldsymbol{AV}$, we can rewrite the approximate solution of overlap test (5) as:

$$\bigcup_{i=1}^{n}(\boldsymbol{SV}_U(I_i) \cap \boldsymbol{SV}_S(I_i) \subseteq \bigcup_{i=1}^{n}(\boldsymbol{AV}_U(I_i) \cap \boldsymbol{AV}_S(I_i)) \tag{6}$$

where $\boldsymbol{AV}_U$ and $\boldsymbol{AV}_S$ denote the AASVs of $\boldsymbol{SV}_U$ and $\boldsymbol{SV}_S$, respectively. Hence, by approximation, we can determine that the two regions, $\boldsymbol{R}_U$ and $\boldsymbol{R}_S$, overlap within $[a,b]$ if:

$$\bigcup_{i=1}^{n}(\boldsymbol{AV}_U(I_i) \cap \boldsymbol{AV}_S(I_i)) \neq \emptyset \tag{7}$$

4.3 Divide-and-Conquer Algorithm

The test of (7) would be accurate if I_i are small. However, this also results in a large number of $\boldsymbol{AV}(I_i)$, and thus much overhead in computing swept volumes and performing set intersections. Therefore, instead of computing all $\boldsymbol{AV}_U(I_i) \cap \boldsymbol{AV}_S(I_i)$ over the entire time interval $[a,b]$, a divide-and-conquer algorithm is used to avoid unnecessary set intersections.

The divide-and-conquer algorithm first test $\boldsymbol{AV}_U([a,b])$ and $\boldsymbol{AV}_S([a,b])$ for an overlap. If an overlap occurs, it then splits the time interval $[a,b]$ into two equal subintervals and splits each of the AASVs into two smaller ones accordingly. This process continues recursively until no overlap occurs or the tested subinterval is smaller than a user defined threshold. The performance of this algorithm for different values of threshold is evaluated in Section 6.

The pseudo-code of the divide-and-conquer algorithm is given in Algorithm 1 (lines 7-24), which forms part of the scalable sorting algorithm.

5 PARALLEL ALGORITHM FOR MULTI-REGION SPACE-TIME INTEREST MATCHING

In this section, we present the principles of our parallel algorithm for multi-region space-time interest matching. This algorithm aims to support matching between n update regions and m subscription regions, and capture interactions between discrete time-steps. We extend the design of (Liu and Theodoropoulos 2009) which divides the matching process into two phases. When it runs on a shared-memory multiprocessor machine, the parallel algorithm is expected to have better computational performance than the serial approach introduced in (Liu and Theodoropoulos 2010).

5.1 The First Phase

In the first phase the algorithm employs a spatial data structure called flat subdivision (Overmars 1992) to efficiently decompose the virtual space into a number of subdivisions. A work unit (WU) is defined as the interest matching process within a space subdivision. During runtime, WUs are distributed across different processors.

A hash table is used to store the space subdivisions and the AASVs. The basic steps to construct the hash table are as following:

1. Decompose the d-dimensional virtual space into flat subdivisions
2. Determine the index k for each subdivision (For $d = 3$, index would be $k(x,y,z), \forall x,y,z \in \mathbb{Z}$)
3. Construct a hash table with the indexes k, so that each slot of the hash table represents a space subdivision
4. Compute the AASV for all update and subscription regions.
5. Hash all AASVs into the hash table based on the hash value $H(v)$, where v is the vertex of the AASVs. The hash function H is defined as follows:

   $H : \mathbb{R}^n \to \mathbb{Z}^n, H(x_i) = \lfloor \frac{x_i}{l_i} \rfloor, i = 1,2,...,n$

   Here, x_i and l_i are the coordinate of a AASVs's vertex and the length of space subdivision respectively, on the corresponding dimension i.

Note that if not all vertices of an AASV are hashed into the same slot, then the AASV exists in multiple subdivisions (slots). Furthermore, a hash table collision indicates that there are at least two AASVs in the corresponding space subdivision.

The hash table is constructed at the initialisation stage. During runtime, the AASVs may be frequently modified. Therefore, the algorithm needs to perform rehashing for the AASVs 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.

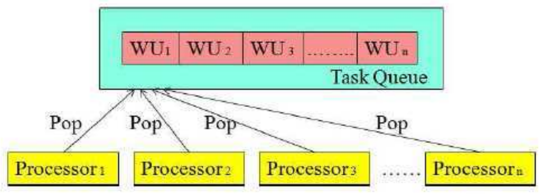

Figure 2: Task Queue for WU Distribution

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. Figure 2 illustrates how the processors and task queue interact during runtime.

Since every space subdivision is isolated during the matching process, lock mechanisms for each WU are unnecessary. Locking is only required when a processor tries to fetch a WU from the task queue.

5.1.1 Load Balancing

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 AASVs reside in a single space subdivision. In this situation, a single processor would be responsible for the matching of all regions.

5.2 The Second Phase

In the second phase, the processors fetch WUs from the task queue one by one and perform interest matching for them. This procedure is repeated until the task queue is empty. When a WU is being processed, a sorting algorithm based on dimension reduction is used to determine the overlapping status of the regions.

During the initialisation stage, the algorithm first constructs a list of AASV's extent end-points for each dimension. By sorting these lists, it can determine which extents overlap. Since the lists are presumed to be in random order, quick-sort would be a good choice for this stage. The complexity of the sorting process is $O((m+n)log(m+n))$ where m is the number of subscription regions and n is the number of update regions.

During runtime, the algorithm re-sorts the lists in order to find out the change of matching results. We can reduce the computational complexity of this process by caching the sorted lists and matching results from the previous time-step and exploiting temporal and geometric coherence (Cohen, Lin, Manocha, and Ponamgi 1995). If this is exploited, the coordinates of extent end-points would change minimally between consecutive time-steps. Therefore, the lists are already nearly sorted before the re-sort operation is performed.

We use insertion-sort to re-sort the lists at runtime. During this process, the overlap statuses are only modified when the insertion-sort performs a *swap*. When this happens, the algorithm carries out pairwise space-time overlap test for the two AASVs in question. Since temporal and geometric coherence is exploited, the chance of swap is extremely small. The computational complexity of re-sorting the lists using insertion-sort is $O(n+m+s)$ for each dimension, where s is the number of swaps.

Details of sorting process are given in Algorithm 1. During the sorting process, every iteration of the algorithm removes an end-point from the *EList* (line 4), inserting it into the correct position in the already-sorted list (line 33), until no end-points remain. If insertion (swap) occurs, the algorithm then carries out the divide-and-conquer approach to determine whether the two regions that contain the two end-points in question overlap (lines 7-24). The whole process is repeated until all *EList* are sorted.

One important difference between this algorithm and the sorting algorithm of (Liu and Theodoropoulos 2009) is that we use swept volumes instead of regions in this approach. The size of update and subscription regions are rarely changed during runtime; however, the size of their AASVs may change frequently due to the arbitrariness of the regions' trajectory. This does not affect the precision of finding potentially overlapped AASVs, but it might increase the chance of swap during the insertion-sort process. We will evaluate the runtime performance of this approach in the next section.

Algorithm 1: Re-Sort the Lists of End-points

```
Data: $[a,b]$: the time interval
Data: $EList$: a list of extent end-points of the corresponding dimension
Data: $Stack$: a stack storing a pair of AASVs and a subinterval $I$
Data: $isOverlapped$: a boolean variable indicating whether the two regions overlap at a certain time
Result: A result list storing the pairs of regions that overlap each other
begin
    foreach dimension do
        for $i \leftarrow 1$ to $Size(EList)-1$ do
            $temp \leftarrow EList[i]$;
            $j \leftarrow i-1$;
            while $j \geq 0$ AND $EList[j] > temp$ do
                $isOverlapped \leftarrow false$;
                $Stack.push$ (AASV containing $EList[i]$,AASV containing $EList[j]$,$[a,b]$);
                while $Stack$ is not empty do
                    $\boldsymbol{AV}_1 \leftarrow Stack.top.\boldsymbol{AV}_1$;
                    $\boldsymbol{AV}_2 \leftarrow Stack.top.\boldsymbol{AV}_2$;
                    $I \leftarrow Stack.top.I$;
                    $Stack.pop()$;
                    if $\boldsymbol{AV}_1 \cap \boldsymbol{AV}_2 \neq \emptyset$ then
                        if $I$ is too small then
                            $isOverlapped \leftarrow true$;
                            break;
                        else
                            Split $I$ into two equal subintervals $I_1$ and $I_2$;
                            $Stack.push$ ($\boldsymbol{AV}_1(I_1)$,$\boldsymbol{AV}_2(I_1)$,$I_1$);
                            $Stack.push$ ($\boldsymbol{AV}_1(I_2)$,$\boldsymbol{AV}_2(I_2)$,$I_2$);
                        end
                    end
                end
                if $isOverlapped$ then
                    Store this region pair to the result list
                else
                    Remove this region pair from the result list
                end
                $EList[j+1] \leftarrow EList[j]$;
                $j \leftarrow j-1$;
            end
            $EList[j+1] \leftarrow temp$;
        end
    end
end
```

6 PERFORMANCE EVALUATION

In this section we present the evaluation of the proposed parallel algorithm for continuous interest matching. We carried out several sets of experiments to compare the performance of six approaches, namely:

1. Discrete interest matching by brute-force (DIM)
2. Continuous interest matching by brute-force (CIM)
3. Discrete interest matching by scalable insertion-sort algorithm (SDIM)
4. Continuous interest matching by scalable insertion-sort algorithm (SCIM)
5. Discrete interest matching by parallel algorithm (PDIM)
6. Continuous interest matching by parallel algorithm (PCIM)

The DIM approach performs interest matching for all n update regions and m subscription regions at discrete time intervals. The CIM approach also performs interest matching for all region pairs; however, instead of testing the overlap status at discrete time intervals, it performs pairwise space-time matching as presented in Section 4 in order to capture the missing events. The SDIM approach is similar to the sorting approach presented in (Liu, Yip, and Yu 2006). We chose it as a comparison target because it is theoretically the fastest (linear time in the general case) algorithm among all serial matching algorithms. The SCIM approach is presented in our previous

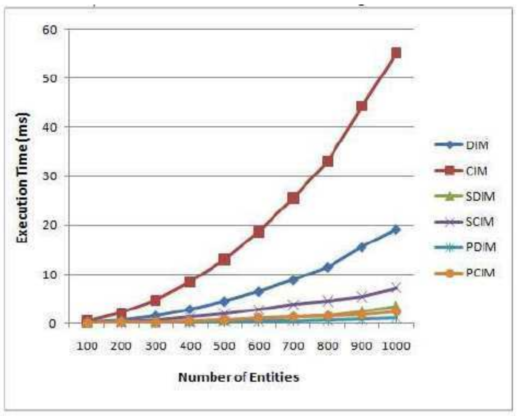

Figure 3: Comparing the Runtime Efficiency of Interest Matching Approaches (Number of Entities varies)

work (Liu and Theodoropoulos 2010). It uses the scalable sorting algorithm 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. The PDIM approach is the parallel algorithm for discrete interest matching, as presented in (Liu and Theodoropoulos 2009). The PCIM approach is the new algorithm proposed in this paper. It performs space-time interest matching similar to the CIM approach. In addition, it exploits parallelism by distributing the workload across multiple processors.

Of the three major design requirements described in Section 1, our performance evaluation only focuses on the runtime efficiency of the six approaches. The filtering precision of the six approaches is similar since they all adopt aura-based scheme. Furthermore, the capturing ability of PCIM is similar to CIM and SCIM since they employ swept volumes and the divide-and-conquer algorithm to perform space-time interest matching. In (Liu and Theodoropoulos 2010), we have showed by experimental evidence that the continuous algorithms (CIM and SCIM) have better capturing ability than the discrete algorithms.

The six algorithms were implemented in C++ with the use of multi-threading. All of the tests were run on an Intel Core 2 Quad Q8400 2.66GHz with 4GB main memory and used the following experimental set-up:

Entity Movement: All entities move in a random direction and undergo linear translational motion (i.e., the degree of $\vec{\boldsymbol{\tau}}(t)$ is equal to 1). The speed factor (SF) represents the average speed of the entities in proportion to its extents. The value of SF varies in different sets of experiments.

- *Entity Distribution*: The entities are distributed randomly across the virtual space.
- *Number of Dimensions*: The algorithms were tested in 3-dimensional space.
- *Number of Regions*: An update region and a subscription region are associated with each moving entity.

6.1 Number of Entities

The first set of experiments compares the runtime efficiency of the DIM, CIM, SDIM, SCIM, PDIM and PCIM with the number of virtual entities extending from 100 to 1000. The value of SF was set to 20 and the threshold of continuous algorithms was set to $\delta t/32$, where δt is the time-step of simulation.

Figure 3 shows the execution time of the six approaches. It is not difficult to see that PDIM has the best performance among discrete algorithms, where DIM has the poorest performance. The difference becomes significant when the number of entities is gradually increased. Similarly, PCIM has the best performance among continuous algorithms, where CIM has the poorest performance. These observations indicate that the parallel approaches (PCIM and PDIM) scale much better than the serial approaches when running on a shared-memory multiprocessor machine. Furthermore, the continuous algorithms require more computational effort than discrete algorithms under the same matching approach (brute-force, sorting, or parallelism). The discrepancy is due to the overhead introduced by the divide-and-conquer algorithm as well as the computation of AASVs.

6.2 Speed Factor

The second set of experiments compares the runtime efficiency of the six algorithms with the value of SF extending from 2 to 20. The number of entities was set to 1000 and the threshold of continuous algorithms was set to $\delta t/32$.

Figure 4 shows the execution time of the six approaches. The results indicate that when the value SF increases, CIM would spend slightly more time on the matching process. This is because the AASVs become bigger when the

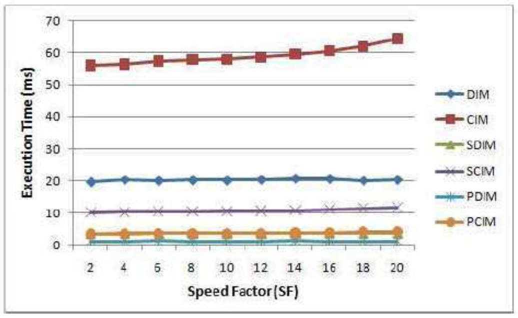

Figure 4: Comparing the Runtime Efficiency of Interest Matching Approaches (Speed Factor varies)

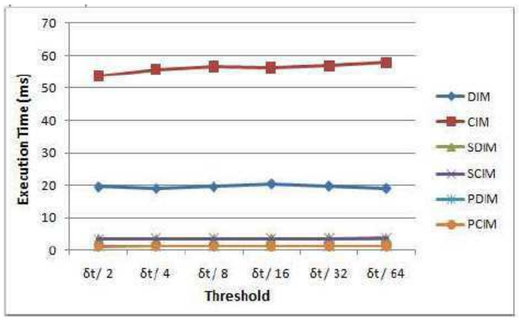

Figure 5: Comparing the Runtime Efficiency of Interest Matching Approaches (Threshold varies)

entities move at high speed, resulting in an increase of the chance of overlap, and thus more computational effort is spent on the divide-and-conquer process. We expected that the same effect would be applied on SCIM and PCIM too; however, since they are much faster than CIM, the results show that the computational overhead generated by increasing the value SF is insignificant for these two algorithms.

The execution time of discrete algorithms is not affected by the change of SF value since they only perform interest matching at discrete time intervals.

6.3 Threshold of Continuous Algorithms

The third set of experiments compares the runtime efficiency of the six algorithms with the threshold of continuous algorithms extending from $\delta t/2$ to $\delta t/64$. The number of entities was set to 1000 and the SF was set to 20.

Figure 5 shows the execution time of the six approaches. The results indicate that when the value of threshold decreases, the CIM approach would spend slightly more time on the matching process. This is because choosing a smaller threshold would cause the divide-and-conquer algorithm to perform more recursions, resulting in an increase of computational overhead. We expected that the same effect would be applied on SCIM and PCIM too; however, since they are much faster than CIM, the experimental results show that the computational overhead becomes insignificant.

The execution time of discrete algorithms are not affected by the change of threshold value since they do not employ the divide-and-conquer method.

6.4 Speedup

The last set of experiments compares the runtime efficiency of the six approaches when running on different number of processors. We only used one Intel Q8400 as testbed, in the case of number of required processors less than 4, some of the physical cores were disabled. The number of working threads was equal to the number available cores. The number of entities, the speed factor, and the threshold of continuous algorithms were set to the constant value of 1000, 20, and $\delta t/64$, respectively.

Figure 6 shows the execution time of the six approaches. Since the brute-force and sorting approaches are designed for serial processing, the execution time of these algorithms did not change significantly when the number

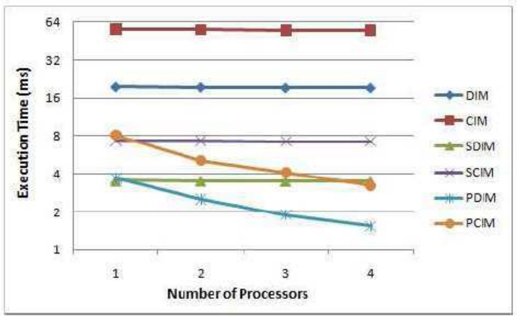

Figure 6: Comparing the Runtime Efficiency of Interest Matching Approaches (Number of Processors varies)

of active processors increased. The execution time of parallel algorithms, however, decreased gradually due to increased number of active processors; this suggests that the proposed algorithm is scalable when running on multiprocessor machines.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we present a parallel algorithm for continuous interest matching. The interest matching process is essential for message filtering services such as HLA DDM which enhance scalability of the DVE system. Most of the interest matching approaches in the literature are designed to solve the trade-off between runtime efficiency and filtering precision. However, they all perform matching at discrete time intervals. As a consequence, they would fail to report events between two consecutive time-steps when the entities move rapidly. To overcome this limitation, our approach aims to capture the missing events by using swept volumes to bound the trajectory of the regions and perform space-time overlap tests over each time interval. Although this process requires additional effort to compute the swept volumes, we use a parallel algorithm to distribute the workload of interest matching across multiple processors.

The experimental results of this paper and the remarks in our previous work (Liu and Theodoropoulos 2009) and (Liu and Theodoropoulos 2010) lead to three important conclusions. Firstly, the proposed algorithm has a very good filtering precision which is similar to all aura-based matching algorithms. Secondly, the proposed algorithm significantly enhances the runtime efficiency of continuous interest matching when running on shared-memory multiprocessor machines such as multi-core processors. Since it is increasingly common to deploy commercial DVE applications such as MOGs on multiprocessor machines, using the proposed algorithm for these applications is more suitable than the serial algorithms. Finally, the experimental results presented in (Liu and Theodoropoulos 2010) show that the continuous algorithm can capture most of the events that are ignored by the discrete algorithms. Hence, we conclude that the proposed algorithm satisfies all three major design requirements of interest matching algorithms, as described in Section 1.

Although our discussion so far is centred on shared-memory multiprocessor machines, the parallel interest matching algorithm can also be extended and applied on cluster of servers. In this case, the machines can no longer exchange data through the shared-memory and therefore additional synchronisation protocols will be required. We are currently designing and implementing these protocols.

REFERENCES

Abdel-Malek, K., D. Blackmore, and K. Joy. 2006. Swept volumes: Foundations, perspectives, and applications. International Journal of Shape Modeling. *International Journal of Shape Modeling* 12 (1): 87–127.

Barrus, J. W., R. C. Waters, and D. B. Anderson. 1996. Locales and beacons: Efficient and precise support for large multi-user virtual environments. In *Proceedings of IEEE VRAIS*, 204–213.

Benford, S., J. Bowers, L. E. Fahlen, and C. Greenhalgh. 1994. Managing mutual awareness in collaborative virtual environments. In *Proceedings of ACM VRST*, 223–236.

Cameron, S. 1985. A study of the clash detection problem in robotics. In *Int. Conf. Robotics and Automation*.

Cohen, J. D., M. C. Lin, D. Manocha, and M. K. Ponamgi. 1995. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proceedings of Symposium on Interactive 3D Graphics*.

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.

Fujimoto, R. M. 2000. *Parallel and distributed simlation systems*. John Wiley and Sons, Inc.

Greenhalgh, C., and S. Benford. 1995. Massive: a collaborative virtual environment for teleconferencing. *ACM transactions on Computer Human Interactions* 2 (3): 239–261.

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

Liu, E. S., and G. K. Theodoropoulos. 2009. 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)*. Singapore.

Liu, E. S., M. Yip, and G. Yu. 2006. Lucid Platform: Applying HLA DDM to Multiplayer Online Game Middleware. *ACM Computers in Entertainment* 4 (4): 9.

Liu, E. S., M. K. Yip, and G. Yu. 2005. Scalable Interest Management for Multidimensional Routing Space. In *Proceedings of the ACM VRST 2005*, 82–85.

Macedonia, M. R., M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. 1995. Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. In *Proceedings of IEEE VRAIS*, 2–10.

Morgan, G., and F. Lu. 2003. Predictive interest management: An approach to managing message dissemination for distributed virtual environments. In *Proceedings of the First International Workshop on Interactive Rich Media Content Production: Architectures, Technologies, Applications, Tools 2003*.

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*.

Overmars, M. H. 1992. Point location in fat subdivisions. *Information Processing Letters* 44 (5): 261–265.

Pan, K., S. J. Turner, W. Cai, and Z. Li. 2007. An Efficient Sort-Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment. In *Proceedings of PADS '07*, 70–82. Washington, DC, USA.

Purbrick, J., and C. Greenhalgh. 2000. Extending locales: Awareness management in massive-3. In *Proceedings of the IEEE Virtual Reality 2000 Conference*, 287. Washington, DC, USA: IEEE Computer Society.

Raczy, C., G. Tan, and J. Yu. 2005. A sort-based DDM matching algorithm for HLA. *ACM Transactions on Modeling and Compututer Simulation* 15 (1): 14–38.

Smed, J., T. Kaukoranta, and H. Hakonen. 2002. A review on networking and multiplayer computer games. In *Proceedings of Int. Conf. on Application and Development of Computer Games in the 21st Century*, 1–5.

Van Hook, D. J., S. J. Rak, and J. Calvin. 1997. Approaches to RTI Implementation of HLA Data Distribution Management Services. In *Proc. of 15th Workshop on Standards for the Interoperability of Distributed Simulations*.

Van Hook, D. J., S. J. Rak, and J. O. Calvin. 1994. Approaches to relevance filtering. In *Proceedings of Eleventh Workshop on Standards for the Interoperability of Distributed Simulations*, 26–30.

AUTHOR BIOGRAPHIES

Elvis S. Liu is a PhD student in the School of Computer Science at the University of Birmingham (UK). Before studying at Birmingham, he worked as a lecturer at the Hong Kong Institute of Vocational Education, and a research associate at the Hong Kong Polytechnic University. He has several years of experience in developing online-game engine and teaching online-game development courses. He received a Bachelor of Science from the University of Hong Kong, and a Master of Science from the University of Newcastle (UK). His email address is <E.S.Liu@cs.bham.ac.uk>.

Dr. Georgios K. Theodoropoulos is currently a Reader in the School of Computer Science at the University of Birmingham (UK). He received a Diploma in Computer Engineering from the University of Patras (Greece) and an M.Sc and Ph.D. in Computer Science from the University of Manchester (UK). His current research is in the areas of parallel and distributed simulation, distributed virtual environments, Grid computing and Peer-to-Peer systems. His email address is <G.K.Theodoropoulos@cs.bham.ac.uk>.