A Use of Matrix with GVT Computation in Optimistic Time Warp Algorithm for Parallel Simulation

[1]Shalini Potham, [2]Syed S. Rizvi, [1]Khaled M. Elleithy, and [3]Aasia Riasat
[1]Computer Science and Engineering Department, University of Bridgeport, Bridgeport, CT 06601 USA
[2] Electronics Engineering Technology Department, ECPI University, Virginia Beach, VA 23462, USA
[3]Department of Computer Science, Institute of Business Management, Karachi, Pakistan
[1]{spotham, elleithy}@bridgeport.edu, [2]srizvi@ecpi.edu, [3]aasia.riasat@iobm.edu

Keywords: Discrete-event simulation, GVT computation, parallel simulation, Time management algorithm.

Abstract

One of the most common optimistic synchronization protocols for parallel simulation is the Time Warp algorithm proposed by Jefferson [12]. Time Warp algorithm is based on the virtual time paradigm that has the potential for greater exploitation of parallelism and, perhaps more importantly, greater transparency of the synchronization mechanism to the simulation programmer. It is widely believe that the optimistic Time Warp algorithm suffers from large memory consumption due to frequent rollbacks. In order to achieve optimal memory management, Time Warp algorithm needs to periodically reclaim the memory. In order to determine which event-messages have been committed and which portion of memory can be reclaimed, the computation of global virtual time (GVT) is essential. Mattern [2] uses a distributed snapshot algorithm to approximate GVT which does not rely on first in first out (FIFO) channels. Specifically, it uses ring structure to establish cuts C1 and C2 to calculate the GVT for distinguishing between the safe and unsafe event-messages. Although, distributed snapshot algorithm provides a straightforward way for computing GVT, more efficient solutions for message acknowledging and delaying of sending event messages while awaiting control messages are desired. This paper studies the memory requirement and time complexity of GVT computation. The main objective of this paper is to implement the concept of matrix with the original Mattern's GVT algorithm to speedups the process of GVT computation while at the same time reduce the memory requirement. Our analysis shows that the use of matrix in GVT computation improves the overall performance in terms of memory saving and latency.

1. INTRODUCTION

Time Warp [12] is an optimistic synchronization protocol that uses run-time detection of errors caused by out-of-order execution of portions of a parallel computation, and recovery using a rollback mechanism [8]. The main advantages of Time Warp protocol is that it offers the potential for greater exploitation of parallelism and, perhaps more importantly, greater transparency of the synchronization mechanism to the simulation programmer [13].

In order to efficiently use optimistic Time Warp algorithm, the algorithm must be able to reclaim memory and re-use it to schedule future events as well as support state saving operations as part of speculative event processing [4]. In order to reclaim memory (e.g., processed messages and snapshots of the logical process (LP's) state), and to allow the operations that cannot be roll backed (e.g., I/O), global virtual time (GVT) is defined [9]. GVT is a lower bound of the timestamp of any rollback that might later occur, and is operationally defined as the smallest timestamp of any unprocessed or partially processed message or anti-message in the system. GVT defines the commitment horizon of the simulation [10]. Events with timestamp less than GVT are referred to as committed events. Irrevocable operations that were invoked by committed events can be performed, and storage occupied by them or their state can be reclaimed. The latter operation is called fossil collection [11].

Mattern's GVT algorithm helps keep all the processes in synchronization by finding the minimum of time stamps of all the messages at a point. It also makes sure that there are no transient messages in the process of execution as it waits for the processes to receive all the messages that are destined for it. The backlog of this algorithm is that the latency may high for large applications. The Mattern's GVT algorithm uses several variables which in turn increase the number of memory fetches.

Asynchronous GVT algorithms rely on the ability to create a "cut" across the distributed simulation that divides events into two categories: past events and future events [10, 11]. GVT can then be defined by the lower bound on unprocessed events in the "past" of the cut, which is in effect an estimate of the true GVT since events are being processed during its computations. In other words, the computation of

GVT distinguishes between safe and unsafe event messages for participating LPs.

Defining a cut mainly depends on the underlying architecture of the machine(s) being used. For instance, in a distributed environment, Mattern [2] proposed the use of message passing to define a cut in such a way that at most two cuts can exist. The first cut C1 indicates that the GVT computation is initiated. On the other hand, the second cut C2 may be needed to deal with transient messages if discovered during the first cut C1.

It should be noted that the first two cuts need not be consistent. Instead, a consistent cut can be defined where there are no messages left that were scheduled in the future of the sending processor but received in the past of the destination processor [14]. These messages can be ignored because by definition they must be scheduled at a time that is greater than the GVT computed using a consistent cut [4]. The "cuts", while not consistent, effectively divide past from future in a causally consistent manner to solve the simultaneous reporting problem [9]. Additionally, because of the use of message counts, it is able to determine if a second "cut" round is needed and traps any transient messages [4, 10]. Related to both "consistent cuts" and sequentially consistent memory is the issue of causality. Zhou et. al [16], propose a "relaxed" causal receive ordering algorithm called critical causality for distributed virtual environments. D'Souza et. al [17], proposes a statistical approach to estimating GVT. Pancerella [18] propose a hardware based scheme whereby host systems using custom network interface cards are interconnected to form an efficient reduction network to rapidly compute GVT.

In our proposed algorithm, we implement the similar mechanism structure suggested by the Mattern's [1, 2] with the use of a matrix. This utilization of the matrix eliminates two of the variables and an array as used by the original Mattern's GVT algorithm. Consequently, the use of matrix with the Mattern's algorithm provides several advantages such as it reduces the number of memory fetches, saves memory, increases the processor speed, and improves the latency. We incorporated the butterfly barrier as it has great performance when compared to the other broadcast and the centralized barriers [7, 19, 20].

In barrier synchronization, all participating LPs are required to meet at the barrier before proceeding to the next stage of event execution. Unlike the other traditional synchronization schemes such as broadcast and centralized barriers, this scheme does not use a counter for recording the arrival of LPs. The use of butterfly barrier avoids any potential hot spots and critical regions at a cost of relatively large execution delay. Once we finish implementing the barrier with the proposed algorithm, the current simulation time is automatically updated. In next section we present a discussion on the Mattern's GVT computation in optimistic simulation.

2. GVT COMPUTATION IN OPTIMISTIC SIMULATION

The term distributed refers to distributing the execution of a single run of a simulation program across multiple processors [2]. One of the main problems associated with the distributed simulation is the synchronization of a distributed execution. If not properly handled, synchronization problems may degrade the performance of a distributed simulation environment [5]. This situation gets more severe when the synchronization algorithm needs to run to perform a detailed logistics simulation in a distributed environment to simulate a huge amount of data [6].

Event synchronization is an essential part of parallel simulation [2]. In general, synchronization protocols can be categorized into two different families: conservative and optimistic. Time Warp is an optimistic protocol for synchronizing parallel discrete event simulations [3]. GVT is used in the Time Warp synchronization mechanism to reclaim memory, commit output, detect termination, and handle errors. GVT can be considered as a global function which is computed many times during the course of a simulation. The time required to compute the value of GVT may result in performance degradation due to a slower execution rate [4].

On the other hand, a small GVT latency (delay between its occurrence and detection) reduces the processor's idle time and thus improves the overall throughput of distributed simulation system. However, this reduction in the latency is not consistent and linear if it is used in its original form with the existing distributed termination detection algorithm [7].

Mattern's [1] has proposed GVT approximation with distributed termination detection algorithm. This algorithm works fine and gives optimal performance in terms of accurate GVT computation at the expense of slower execution rate. This slower execution rate results a high GVT latency. Due to the high GVT latency, the processors involve in communication remain idle during that period of time. As a result, the overall throughput of a discrete event parallel simulation system degrades significantly. Thus, the high GVT latency may prevent the widespread use of this algorithm in discrete event parallel simulation system.

Preiss [15] introduced a scheme which places the processing elements (PE) into a ring. The first round completes when a token has been passed around the ring and returns to the initiating PE. When the initial token is received, a PE begins accounting for remotely sent messages which may or may not complete prior to the GVT computation. On receiving the second token, a local GVT value is computed as the minimum between the value stored in the token and the PEs local minimum [4].

In this paper, we examine the potential use of matrix with the Mattern's GVT structure using a ring. Our analysis shows that the use of matrix with the Mattern's GVT structure can reduce the memory requirement and improve the time complexity. The performance measure adopted in this paper is the achievable latency for a fixed number of

processors and the number of message transmission during the GVT computation. Thus, the focus of this paper is on the implementation of matrix. In other words, we do not focus on how the GVT is actually computed. Instead, our focus of study is on the parameters (if any) or factors that may improve the latency involved in GVT computation. In addition, we briefly describe that what changes (if any) may introduce due to the implementation of this new structure that may have an impact on the overall memory requirement and latency.

3. IMPLEMENTATION OF MATRIX FOR GVT COMPUTATION

In our proposed algorithm, the GVT calculation will be done by implementing an *N* x *N* matrix and executing the two separate algorithms for each cut C1 and C2. The overall flow of GVT computation is shown in Fig. 1. In our proposed algorithm, an LP can send and receive two types of messages: *green* and *red*. Green messages indicate those messages that are safe to execute by an LP at any given point in simulation time. When one LP sends a green message to one of the neighboring LPs, the cell that belongs to both sending and receiving LPs in a matrix gets updated. The process of receiving a message and updating the cell is shown in Fig.2. On the other hand, the red messages represent those messages that are referred as the straggler or the transient messages (i.e., an event-message which may arrive in the past of an LP). These messages are handled as shown in the Fig. 3. For the sake of simplicity, we divide the algorithms for both cuts C1 and C2.

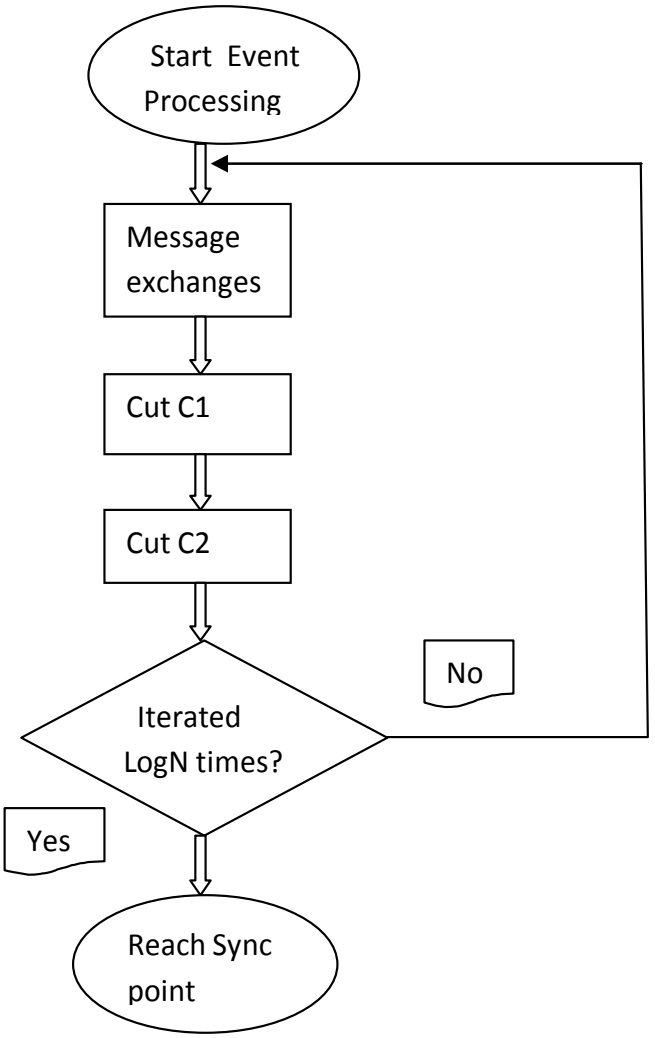


Fig.1. A high level architecture of the proposed algorithm that shows the flow of data with the matrix and butterfly barrier

Algorithm 1: Generation of event-messages and minimum time stamp computation.

ALGO GVT_FLY(V[N][N], $T_{\min}$, T_{now}, T_{red}, $T_{s,n}$)
Begin
$n = \log_2 N$
Loop *n* times
Begin
//Green message sent by LP_i to LP_j
V[i][j] = V[i][j]+1
//Green message received by LP_i
V[i][i] =V[i][i]+1
//Calculate minimum time stamp
$T_{\min} = Min(T_{red}, T_{s,n})$
End

Algorithm 2: Red message generation and calculation of minimum time stamp for CUT C1.

CUT C1:
//Messages exchanged at this point are the red messages
// Red message sent by LP_i to LP_j
V[i][j] = V[i][j]+1
// Red message received by LP_i
V[i][i] = V[i][i]+1
//Calculate minimum time of red messages
$T_{red} = Min(T_{red}, T_s)$
Forward token to appropriate LP

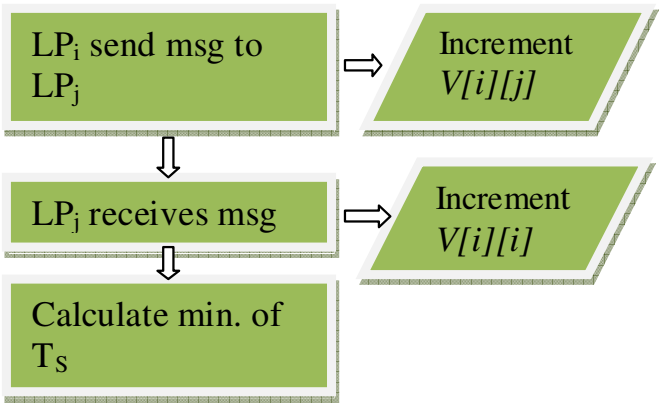


Fig. 2. Handling green messages

Algorithm 3: Forward token and constructing CUT C2.

CUT C2:
//Wait until all messages are received
Wait until $V[i][i] = \left\langle \sum_{j=1}^{N} V[j][i] \right\rangle - V[i][i]$
Forward token to appropriate LP
$T_{now} = Min(T_{red}, T_{\min})$
END LOOP
END ALGO

3.1. Overview of the Proposed Algorithm

The Mattern's algorithm uses *N* vectors of size *N* to maintain a track of the messages being exchanged among the LPs. It also uses an array of size *N* to maintain a log of number of messages a particular LP needs to receive. On the whole, it uses (*N*+1) vectors of size *N*. This increases the number of fetches to memory resulting in more processor idle time.

In our proposed algorithm, we implement an *N* x *N* matrix to calculate the GVT whose flow can be explained as shown in Fig.1. Firstly, the LPs exchange green messages (i.e., green messages represent those messages that are safe to process by LP). Whenever an LP_i sends a green message to LP_j, the cell V[i][j] of the matrix gets updated as shown in Fig.2. On the other hand, if LP_i receives a message, the cell V[i][i] of the matrix is updated. At this point, we also calculate the minimum of all the time stamps of the event messages. After a certain period of time, when the first cut point *C1* is reached, the LPs start exchanging the red messages (i.e., the red messages represent those messages that are referred as the straggler or the transient messages). These messages are handled as shown in the Fig. 3.

When an LP_i sends a red message to LP_j, the cell V[i][j] of the matrix is updated. On the other hand, if LP_i receives a message, the cell V[i][i] of the matrix is updated. At this cut point, we also calculate the minimum timestamp of all the red messages and then the control is passed to the appropriate pair-wise LP. Next, at second cut point *C2*, the LPs have to wait until all the messages destined to them are being received and then calculate the current simulation time as the minimum of the minimum time stamps calculated for red and green messages.

The control token is then forwarded to the appropriate pair-wise LP. Since we are using the butterfly barrier, the entire process is repeated $\log_2 N$ times. In other words, the condition for this algorithm is that the number of processes involved in the system should be a multiple of 2 (i.e., $N = 2^k$).

For the sake of a comprehensive explanation of the proposed algorithm, let us take an example of four LPs communicating with each other as shown in the Fig. 4. It can be seen in Fig. 4 that the four LPs are exchanging messages with respect to the simulation time. Let us see how it modifies the cells of a matrix which are initialized to zero. From the Fig.4, let us understand how the cells are modified with respect to time. The first message is sent by LP_1 to LP_3.

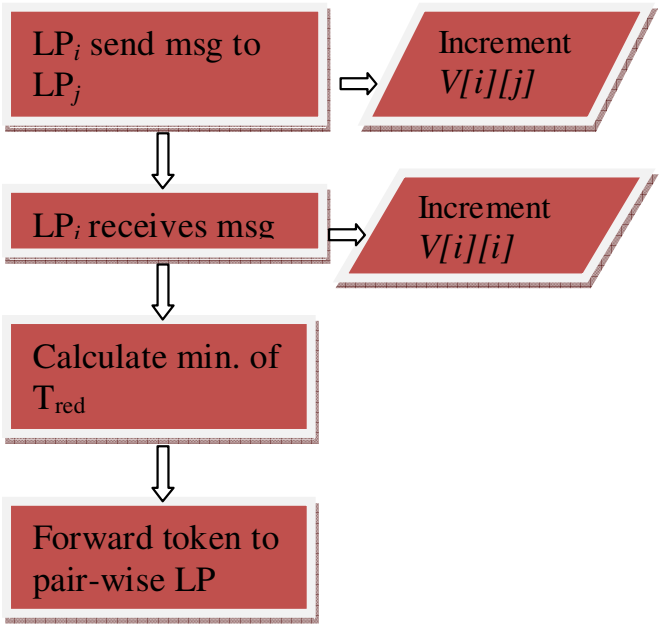
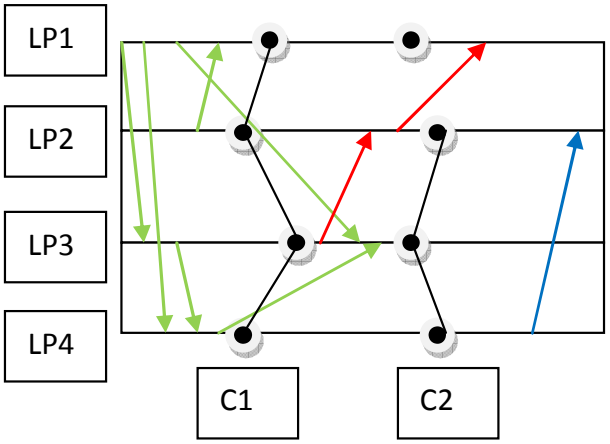


Fig.3 Cut *C1* handling Red messages that represent the transient or struggler messages



→ Represents Cut points
Green-font → green messages (safe events)
Red-font → red messages (transient/struggler messages)

Fig.4. Example of message exchanges between the four LPs. The C1 and C2 represent two cuts for green and red messages.

TABLE I: MATRIX OF 4 LPS EXCHANGING GREEN MESSAGES

	V1	V2	V3	V4
V1	1	0	2	1
V2	1	0	0	0
V3	0	0	1	1
V4	0	0	1	2

TABLE II: MATRIX OF 4 LPS AT CUT C1

	V1	V2	V3	V4
V1	1	0	2	1
V2	2	1	0	0
V3	0	1	3	1
V4	0	0	1	2

TABLE III: MATRIX OF 4 LPS AT CUT C2

	V1	V2	V3	V4
V1	2	0	2	1
V2	2	2	0	0
V3	0	1	3	1
V4	0	1	1	2

As a result, the cell V[1][3] of the matrix is incremented and the message is immediately received by the LP_3 that will increment the cell V[3][3] of the matrix. In the second round, the next message is sent by LP_1 to LP_4. Consequently, the cell V[1][4] of the matrix is incremented and since the message is immediately received by the LP_4, the cell V[4][4] of the matrix is incremented. The next message is sent by LP_3 to LP_4 that will increment the cell V[3][4] of the matrix. However, before this message could be received by LP_4, the next message is sent by LP_1 to LP_3.

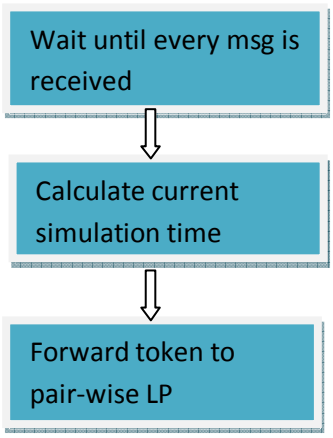


Fig 5: Cut C2 handling green messages for synchronization

The result of this transmission would be an increment in the cell V[1][3] of the matrix. As time progresses, the LP_4 receives the message that results an increment in the cell V[4][4] of the matrix and so on. Table I shows the message exchanges till point *C1*. Table II shows the message exchanges after *C1* and before *C2* and Table III shows the message exchanges after *C2*.

At point *C2*, the LP has to wait until it receives all the messages that are destined to be received by it. This can be done by using the condition that the LP_i has to wait until the value of the cell V[i][i] of the matrix is equal to the sum of all the other cells of the column 'i'. In other words, LP_i has to wait until the following expression is satisfied:

$$V[i][i] = \left\langle \sum_{j=1}^{N} V[j][i] \right\rangle - V[i][i]$$

As an example, if we take V1 from Table II, then at cut point *C2*, it has to wait until $V[1][1] = V[2][1] + V[3][1] + V[4][1]$. According to Table II, the value of V[1][1] is '1' and the sum of other cells of first column is '2'. This implies that the LP_1 has to wait until it receives the message which is destined to reach it. Once it receives the message, it increments V[1][1] and again verifies weather if it has to wait. If not, it then passes the control token to the next appropriate pair-wise LP. Fig. 5

illustrates that how green messages can be handled by Cut *C2*.

Every time the process forwards the control token, it also updates the current simulation time and as a result, we do not require additional instructions as well as time to calculate the GVT. This eliminates the need of communicating the GVT time among the different LPs exchanging messages. This saves time which in turns improves the GVT latency. This algorithm proves helpful in upgrading the system performance of the parallel and distributed systems.

4. CONCLUSION

In this paper, we present an algorithm that helps us to optimize the memory and processor utilization by using matrices instead of using *N* different vectors of size *N* in order to reduce the overall GVT latency. The improved GVT latency can play a vital role in upgrading the parallel/distributed system's performance. In the future, it will be interesting to develop an algorithm to calculate GVT using the tree barriers.

REFERENCES

[1] Mattern, F., Mehl, H., Schoone, A., Tel, G. *Global Virtual Time Approximation with Distributed Termination Detection Algorithms*. Tech. Rep. RUU-CS-91-32, Department of Computer Science, University of Utrecht, The Netherlands, 1991.

[2] Friedemann Mattern, "Efficient Algorithms for Distributed Snapshots and Global virtual Time Approximation," *Journal of Parallel and Distributed Computing,* Vol.18, No.4, pp. 423-434, 1993.

[3] Ranjit Noronha and Abu-Ghazaleh, "Using Programmable NICs for Time-Warp Optimization," *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, PP 6-13, 2002.

[4] D. Bauer, G. Yaun, C. Carothers, S. Kalyanaraman, "Seven-O' Clock: A new Distributed GVT Algorithm using Network Atomic Operations," *19th Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*, PP 39-48.

[5] Syed S. Rizvi, Khaled M. Elleithy, and Aasia Riasat, "Reducing Null Message Traffic in Large Parallel and Distributed Systems," *13th IEEE Symposium on Computers and Communications (ISCC'08)*, pp. 1115 - 1121, July 6 - 9, 2008, Marrakech, Morocco.

[6] Lee A. Belfore, Saurav Mazumdar, and Syed S. Rizvi et al., "Integrating the joint operation feasibility tool with JFAST," *Proceedings of the Fall 2006 Simulation Interoperability Workshop*, Orlando Fl, September 10-15 2006.

[7] Syed S. Rizvi, Aasia Riasat, and Khaled M. Elleithy. An Efficient Optimistic Time Management Algorithm for Discrete-Event Simulation System, *International Journal of Simulation Modeling,* Vol. 9. No. 3, pp. 117 - 130, 2010.

[8] Samir R. Das, Richard M. Fujimoto, Adaptive memory management and optimism control in time warp, ACM Transactions on Modeling and Computer Simulation (TOMACS), v.7 n.2, p.239-271, April 1997.

[9] R. M. Fujimoto, "Time warp on a shared memory multiprocessor," in *Proceedings of the 1989 International Conference on Parallel Processing*, vol. 3, pp. 242–249, August 1989.

[10] R. M. Fujimoto and M. Hybinette, "Computing global virtual time in shared-memory multiprocessors," *ACM Trans. Model. Comput. Simul.*, vol. 7, no. 4, pp. 425–446, 1997.

[11] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*. "John Wiley and Sons, Inc.", 2000.

[12] D. Jefferson, "Virtual Time", *ACM Trans. Prog. Lang. and Systems*, vol. 7, no. 3, July, pp. 404–425, 1985.

[13] Y-B. Lin and E. D. Lazowska, "Determining the Global Virtual Time in a Distributed Simulation", In *Proceedings of the 1990 International Conference on Parallel Processing*, vol. 3, August, pp. 201–209, 1990.

[14] F. Mattern. "Efficient Distributed Snapshots and Global virtual Time Algorithms for Non-FIFO Systems", *Journal of Parallel and Distributed Computing (JPDC)*, vol. 18, no. 4, August, pages 423–434, 1993.

[15] B. R. Preiss, "The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environments", In *Proceedings of the SCS Multiconference on Distributed Simulation*, vol. 21, March, pp. 139–144,1989.

[16] S. Zhou, W. Cai, S. J. Turner and F. Lee, "Critical Causality in Distributed Virtual Environments", In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS '02)*, pp. 53–59, 2002.

[17] L. M. D'Souza, X. Fan, and P. A. Wilsey, "pGVT: An Algorithm for Accurate GVT Estimation", In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pp. 102–109, 1994.

[18] C. M. Pancerella, and P. F. Reynolds, "Disseminating Critical Target-Specific Synchronization Information in Parallel Discrete-Event Simulation", In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, pp. 52–59, 1993.

[19] Abdelrehman Elleithy, Syed Rizvi, and Khaled Elleithy, "Investigating the Effects of Trees and Butterfly Barriers on the Performance of Optimistic GVT Algorithm," *Advanced Techniques in Computing Sciences and Software Engineering,* pp. 449 – 453, Springer, 2010, ISBN:978-90-481-3659-9.

[20] Syed Rizvi, Dipali Shah, and Aasia Riasat, "Implementation of Tree and Butterfly Barriers with Optimistic Time Management Algorithms for Discrete Event Simulation," *Advanced Techniques in Computing Sciences and Software Engineering,* pp. 455 – 460, Springer, 2010, ISBN:978-90-481-3659-9.