A GVT Based Algorithm for Butterfly Barrier in Parallel and Distributed Systems

Syed S. Rizvi, Shalini Potham, and Khaled M. Elleithy
Computer Science Department, University of Bridgeport, Bridgeport, CT 06601 USA
{srizvi, spotham, elleithy}@bridgeport.edu

Abstract-Mattern's GVT algorithm is a time management algorithm that helps achieve the synchronization in parallel and distributed systems. This algorithm uses ring structure to establish cuts C1 and C2 to calculate the GVT. The latency of calculating the GVT is vital in parallel/distributed systems which is extremely high if calculated using this algorithm. However, using synchronous barriers with the Matterns algorithm can help improving the GVT computation process by minimizing the GVT latency. In this paper, we incorporate the butterfly barrier to employ two cuts C1 and C2 and obtain the resultant GVT at an affordable latency. Our analysis shows that the proposed GVT computation algorithm significantly improves the overall performance in terms of memory saving and latency.

Keywords-Time management algorithm, latency, butterfly barrier

I. INTRODUCTION

A parallel and distributed system is an environment where a huge single task is being divided into several sub-tasks and each terminal getting a sub-task to execute. The main problem that is being faced here is the synchronization. All the processes need to be synchronized as the main aim of distributed system is that the final output after execution of entire task should be exactly the same as that of the output attained when the same task is executed sequentially on a single machine.

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 is high. This keeps the algorithm away from its widespread usage. The performance of a parallel/distributed system can be degraded if the latency for computing the GVT is high. The Mattern's GVT algorithm uses several variables which in turn increase the number of memory fetches.

In the proposed algorithm, we implement the similar mechanism structure suggested by the Mattern's [1] 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 barriers such as broadcast and the centralized barriers [7]. When we finish implementing the barrier with the proposed algorithm, the current simulation time is updated. This implies that there is no need to communicate the minimum time or the simulation time reducing the message exchanges. This, therefore, improves the latency at affordable rate.

II. RELATED WORK

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]. Global virtual time (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 prevents the widespread use of this algorithm in discrete event parallel simulation system.

However, if we could improve the latency of the GVT computation, most of the discrete event parallel simulation system would likely to get advantage of this technique in terms of accurate GVT computation. In this paper, we examine the potential use of butterfly barriers with the Mattern's GVT structure using a ring. Simulation results demonstrate that the use of the tree barriers with the Mattern's GVT structure can significantly improve the latency time and thus increase the overall throughput of the parallel simulation system. 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 butterfly barrier structures. 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 barrier structure that may have an impact on the overall latency.

III. PROPOSED ALGORITHM

In this section, we present the proposed algorithm. For the sake of simplicity, we divide the algorithms for both cuts C1 and C2.

A. *The Proposed Algorithm*

```
ALGO  GVT_FLY(V[N][N],Tmin,Tnow,Tred,Ts,n)
  Begin
    n = log2 N
    Loop n times
     Begin
       //Green message sent by LPi to LPj
         V[i][j] = V[i][j]+1
       //Green message received by LPi
         V[i][i] =V[i][i]+1
       //Calculate minimum time stamp
         Tmin = min (Tmin,Ts)

  CUT C1:
    //Messages exchanged at this point are the red messages
    // Red message sent by LPi to LPj
     V[i][j] = V[i][j]+1
    // Red message received by LPi
     V[i][i] = V[i][i]+1
    //Calculate minimum time of red messages
     Tred = min (Tred,Ts)
    Forward token to appropriate LP

  CUT C2:
    //Wait until all messages are received
    Wait until(V[i][i] = Σ_{j=1}^{N} V[j][i] -V[i][i])
    Forward token to appropriate LP
   Tnow= min(Tred,Tmin)
END LOOP
END ALGO
```

B. *A Detailed 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. 5. It can be seen in Fig. 5 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.5, let us understand how the cells are modified with respect to time. The first message is sent by LP_1 to LP_3. 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

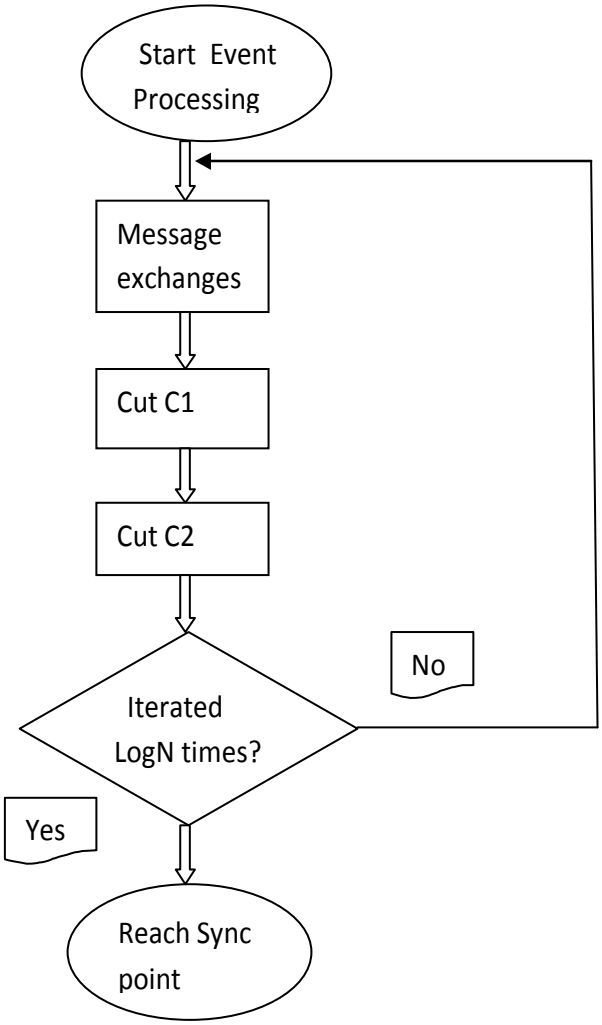


Fig.1. A high level architecture of the proposed algorithm that shows the flow of data with the matrix and butterfly barrier

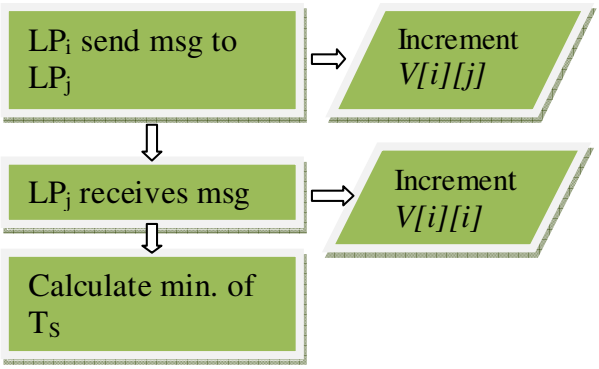


Fig. 2. Handling green messages

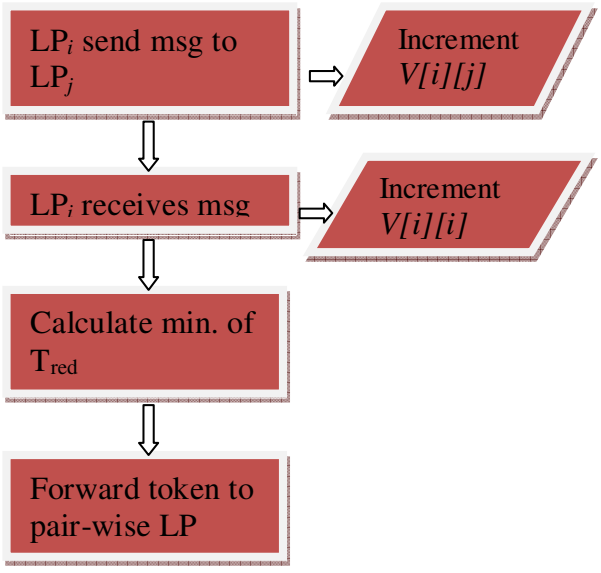


Fig.3 Cut *C1* handling Red messages that represent the transient or struggler messages

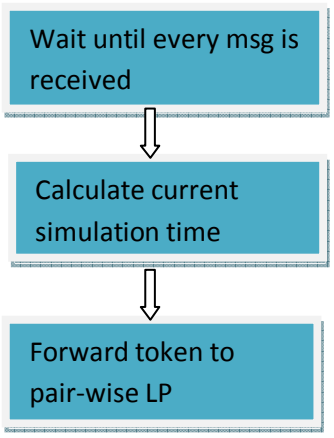


Fig 4: Cut C2 handling green messages for synchronization

the second round, the next message is sent by LP_1 to LP_2. 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. 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 V[i][i]=$(\sum_{j=1}^{N} V[j][i])$ -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.

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

TABLE I: MATRIX OF 4 LPS EXCHANGING GREEN MESSAGES

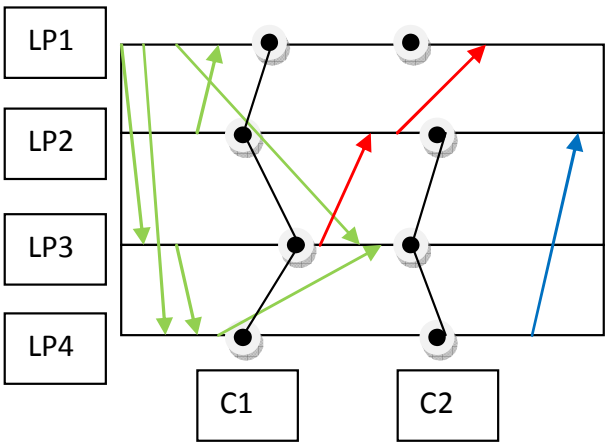
	V1	V2	V3	V4
V1	1	0	2	1
V2	1	0	0	0
V3	0	0	1	1
V4	0	0	1	2

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



● → Represents Cut points

Green-font → green messages (safe events)

Red-font → red messages (transient/struggler messages)

Fig.5. Example of message exchanges between the four LPs. The C1 and C2 represent two cuts for green and red messages.

time which in turns improves the GVT latency. This algorithm proves helpful in upgrading the system performance of the parallel and distributed systems.

IV. 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, 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, Khaked. M. Elleithy, Aasia Riasat, "Minimizing the Null Message Exchange in Conservative Distributed Simulation," *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, CISSE 2006,* Bridgeport CT, pp. 443-448 ,December 4-14 2006,

[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, Khaled M. Elleithy, and Aasia Riasat, "Trees and Butterflies Barriers in Mattern's GVT: A Better Approach to Improve the Latency and the Processor Idle Time for Wide Range Parallel and Distributed Systems", *IEEE International Conference on Information and Emerging Technologies (ICIET-2007),* July 06-07, 2007, Karachi, Pakistan.