Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

EVENT SCHEDULING SCHEMES FOR TIME WARP ON DISTRIBUTED SYSTEMS

Eunmi Choi

Department of Computer Science
Michigan State University
East Lansing, MI 48824-1027, U.S.A.

Dugki Min

Department of Computer Engineering
KonKuk University
MoJinDong KwangJinGu Seoul, KOREA

ABSTRACT

To design an efficient event scheduling strategy for Time Warp on distributed systems, we study two major factors. The first factor is the amount of computation to be processed in a processor between two consecutive communication points. The computation granularity must be small enough to prevent Time Warp from progressing events over-optimistically, while large enough to minimize the degrading effects of performance due to the communication overhead. The second factor is the balancing progress of logical processes in a processor. Within the range of the given computation granularity, logical processes assigned to a processor progress at different speeds and this may cause more rollback situations. Considering these two factors, we propose two event scheduling schemes: the *Balancing Progress by Execution Chance (BPEC)* scheme and the *Balancing Progress by Virtual Time Window (BPVTW)* scheme. The BPEC scheme controls the progress of a process by limiting the number of chances for the process to schedule events, while the BPVTW scheme controls the progress by using a virtual time window, which allows events to be executed only if their timestamps are within the interval of the virtual time window. We have performed experiments on a cluster of DEC Alpha workstations to demonstrate the performances of both schemes.

1 INTRODUCTION

In simulating large VLSI circuits on distributed memory systems, one of the most important considerations for the Time Warp (Jefferson 1985) is to schedule events. Although the number of processors in distributed-memory systems is much smaller than that of the logical processes in VLSI circuits, the local memory size of each processor is so large that many logical processes can be allocated to a processor. The ratio of the number of logical processes to the number of physical processors is called the *LP ratio*. It represents the number of assigned logical processes per processor. For the simulation of VLSI circuits in which hundreds of thousands of gates are integrated in a chip, the LP ratio on distributed-memory systems of hundreds of processors is in the range of a couple of thousands. In addition to the large LP ratio, because of the optimistic characteristic of Time Warp, many processes allocated in a processor are likely to be active with many unprocessed events. Among the unprocessed events of active processes, the processor decides which event is executed first. The order of event execution determines the progress of the processes, and therefore it can significantly affect the performance of Time Warp on distributed-memory systems.

In this paper, we investigate two major factors in order to design an efficient event scheduling strategy for Time Warp on distributed-memory systems. The first factor is the amount of computation to be processed in a processor between two consecutive interprocessor communication points. This quantity is called the *computation granularity*. By adjusting the computation granularity to be larger than a certain degree, we can prevent the communication overhead from dominating the overall simulation performance. A large granularity, however, could increase the number of rollbacks, wasting the simulation time for re-computations. That is, while processors execute a large number of events without communication, processes could propagate many immature events and therefore the protocol should recover the earlier system state from the wrongly-advanced state when an event arrives whose timestamp is smaller than the current LVT (Local Virtual Time). A considerable amount of overhead can occur in the rollback situations because the past state of each process should be saved and recovered by re-computations. As a consequence, the appropriate computation granularity might be small enough to prevent Time Warp from progressing events over-optimistically, while keeping

large enough to minimize the communication overhead. The second factor is the balancing progress of logical processes in a processor. Some logical processes may advance too far ahead of others, possibly leading to inefficient usages of memory and excessive rollbacks, especially for large application problems. By adjusting the granularity, we can control the degree of optimism appropriately as described in the first factor. However, within the range of the given computation granularity, the logical processes assigned to the same processor would progress at different speeds and this may cause more rollback situations. If we control the speed of each process by giving more chances of executing events to slower processes, we may reduce the frequency of rollback situations. We study two schemes of balancing the progress of processes; one of which is to restrict a process to execute events no more than the given number of times, and the other is not to allow an event to be executed if its timestamp is far ahead of other events in different processes. The details of those schemes are studied in Section 3.

The following section describes the simulation model and the terminologies used in this paper. In Sections 3, the event scheduling schemes are presented with the issues of computation granularity and balancing progress of processes. Section 4 shows the experimental results on a distributed system. The concluding remarks are presented in Section 5.

2 THE SIMULATION MODEL

Time Warp has been studied by several researchers recently on distributed-memory systems. Carothers et al. (1994) studied the effect of communication delay on a distributed system when the computation granularity is fixed to the event granularity. Preiss and Loucks (1995) studied memory management techniques on a distributed system. This paper focuses on event scheduling issues of the optimistic protocol on distributed-memory systems. Our implementation of Time Warp on distributed-memory systems is called *Distributed Optimistic Protocol (DOP)*. The DOP executes events in an event-driven and optimistic fashion.

2.1 Rollback Handling

To handle the rollback situation, we use a cancellation mechanism called the *immediate cancellation* (Chung and Chung 1991). In this cancellation mechanism, the process that receives a straggler re-starts to process from the point of the straggler's timestamp, by ignoring all messages sent so far and all the already-processed events. The cancellation scheme applies to the input event queue where a straggler arrives. In the immediate cancellation, there is no antimessage to correct the already-sent wrong events unlike the aggressive cancellation mechanism (Jefferson 1985) and the lazy cancellation mechanism (Gafni 1988). At this point, it is the same as the direct cancellation mechanism (Fujimoto 1989) on shared memory systems.

2.2 Data Structure for Event Manipulation

In parallel discrete event simulation, the events that are generated but not yet executed are pending in the input queues. Each processor schedules the pending events according to the event scheduling strategy. The implementation of data structures of event queues and the event scheduling strategy have crucial influences on the simulation performance.

In our implementation, each logical process has its own event queue where all unprocessed events are pending and the already processed events are stored together. This type of implementation with an individual event queue per process is better than one central event queue shared by all processes in a processor, especially when the LP ratio is large. If all processes share one event queue together in a processor, it takes times to search and manipulate all pending events and rollbacked events. The data structure of the individual event queue per process is also easily adaptive to the situation of migrating a process to another processor by a dynamic load management mechanism. In each process's event queue of our implementation, all the events are sorted in the increasing order of timestamps. Instead of using separate output event queues, using a single event queue which can be used as both input and output event queues can reduce the queue manipulation time when a straggler arrives, by eliminating the corresponding event moving and queue handling from output event queues.

Our event scheduling schemes described in the next section basically follow the 'smallest-timestamp-first' selection scheme. Since searching all assigned processes linearly for the smallest timestamped event takes a lot of time in a large-scale application, we use a central priority queue structure for scheduling events in a processor. Several data structures of the priority queue have been proposed such as the calendar queue (Brown 1988) and the skew heap (Jones 1989). In this paper, we use the heap structure for the priority queue. Each item in the priority queue has a timestamp as the key to be sorted and a process index number. After each processor evaluates processes and determines whether they are active or not, unprocessed events are selected from the active

processes according to the event scheduling scheme, and enqueued with the timestamps into the priority queue by the processor scheduler. From the scheduling heap, each processor dequeues the smallest one from the top and executes the event on the corresponding process.

2.3 GVT Computation

The GVT (Global Virtual Time) computation is necessary in optimistic protocol for fossil collection and for checking the termination condition. The GVT on distributed-memory systems is obtained by computing the minimum value of LVTs from all processes and the timestamps of events in the in-transit messages, which are sent by a sender processor and not yet received by the receiver processor, where the LVT of a process is determined by the smallest unprocessed event in a process. Since each processor maintains many processes, PVT (Processor Virtual Time) is computed as the minimum value of LVTs among the processes in a processor with considering the in-transit messages, and it is used to compute the GVT. Thus, the GVT can be computed as the minimum value of PVTs from all processors. We use a token passing mechanism to collect PVT values asynchronously from processors and compute the GVT value.

3 EVENT SCHEDULING

As one of the most popular simulation applications, VLSI circuits involve a large number of objects performing a small amount of computation for each communicating event. A VLSI circuit usually has several tens to more than hundreds of thousands of gates (or logical processes). The manipulation of an event requires around a hundred machine instructions; its event granularity is very small. In contrast, distributed memory systems consist of a much smaller number of big processors and the interprocessor communication latency is large. Thus, each processor should schedule the unprocessed events of a large number of assigned processes efficiently in order to achieve good performance. As we mentioned in Section 1, there are two important considerations we should make as we design an event scheduling strategy: the computation granularity and the progress balancing.

3.1 Computation Granularity

Computation granularity is the amount of computation to be processed by a processor between two consecutive interprocessor communication points (Choi and Chung 1995). This quantity is also called the *grain size*. If there is little overhead of interprocessor communication, the computation granularity is not a matter of concern. In this case, the ideal strategy is to perform an interprocessor communication for each event execution in order to exchange information as soon as possible. Frequent communications, however, may cause excessive rollbacks due to the tendency of overly optimistic execution in Time Warp. Therefore, a mechanism for restraining the overly optimistic execution is required in this case. The implementations of Time Warp on shared memory systems use this 'communication-per-event' style for the communicating pattern. As an exception, Das et al. (1994) discuss about the possibility of improving performance by processing a batch of k events, where k is the size of the batch. They claim that the batch processing approach reduces queue management overheads somewhat.

In distributed-memory systems, the communication latency is long, and therefore the computation granularity must be an important issue to be studied. An experimental study (Carothers et al. 1994) has investigated the effects of communication overhead on the performance of Time Warp, considering two types of simulation applications; one with a large event granularity and the other with a small event granularity. In the study, the computation granularity is fixed to the event granularity without allowing several events to be executed all at once. According to their performance results, the communication latency in distributed computing environments can significantly degrade the efficiency of Time Warp in the application problems which contain a large number of logical processes with a small event granularity. In contrast, for applications having large grained events, the communication latency appears to have little effect on the performance of Time Warp.

It is clear that a substantial amount of computation should be performed between interprocessor communications in order to yield appropriate performance results on distributed systems. In VLSI circuit simulations which have a small event granularity, a large computation granularity could be achieved by handling a batch of events together between communication points. However, the batch processing of a number of events with infrequent communications may unbalance the progress of logical processes in a processor from those of other processors. In this case, there is a high possibility that the incoming events from other processors become stragglers. The computation granularity, thus, should be tuned properly, depending on the characteristics of simulation application problems and the used computer systems. The optimal granularity should be large enough that the simulation performance cannot be degraded by

the communication overhead, and small enought that rollback situations cannot be excessive.

3.2 Balancing Progress of Processes

The major characteristic of Time Warp is that each process can asynchronously execute its unprocessed events without considering causality effects. Due to this characteristic, some logical processes may advance too far ahead of others, possibly leading to the inefficient use of memory and excessive rollbacks. This situation might happen particularly for a large-scale simulation application with a small event granularity, such as VLSI circuits. By using the scheduling scheme of balancing the progress of logical processes, we can prevent the simulation from propagating incorrect computations too far ahead and reduce seriously excessive rollback situations. Two balancing schemes are presented in the following subsections.

3.2.1 Balancing Progress by Execution Chance (BPEC)

The BPEC is a scheme that balances the progress of logical processes. It limits the number of chances (opportunities) of executing events per logical process. In this scheme, we have two controllable parameters: one parameter for the computation granularity and the other parameter for balancing progress.

As the first parameter to control the computation granularity, this scheme sets the maximum number of events that can be executed by a processor between communication points. This quantity is called the *maximum batch size (MBS)*. Within the limit of the MBS, the processor scheduler selects events from logical processes through the central scheduling priority queue. That is, each processor selects and executes events up to the limit of the MBS. The scheduler counts the number of events executed in the processor since the last communication point while selecting events from the priority queue. Once the number of executed events reaches MBS, the processor stops scheduling events and performs the interprocessor communication. The second parameter is the maximum number of chances needed to execute events per logical process. When the scheduler selects events from the logical processes through the priority queue, it also counts the number of executed events per logical process. The purpose of counting events per process is to balance the progress of processes. By giving the appropriate execution chances to processes, we can achieve the benefit of using parallel optimistic protocol. As well as boosting only processes far behind, the overall progress of processes can advance by optimistically executing all processes. The maximum number of chances that a logical process can execute events between interprocessor communications is called the *Balancing Progress Chances (BPC)*. If a logical process no longer has unprocessed events or it has already executed as many events as the BPC, then the logical process is not allowed to enqueue its events into the priority queue.

By controlling the maximum allowable chances of executing unprocessed events per logical process, the BPEC scheme not only balances the relative speed of the processes' progress, but also controls the computation granularity. In the case that the MBS is set to be larger than the optimal value of grain size, the computation granularity can be adjusted by using the proper BPC value. When all logical processes are individually controlled by the BPC, the total number of executed events in a processor cannot increase too much, and the granularity between two communication points will be adjusted properly. The BPC is also able to control the event schedule by giving an equal number of execution chances to the logical processes. Instead of only executing events on a specific logical process, every process has the same chance to execute its events. Thus, the BPEC scheme contributes to progress balancing of logical processes and reducing the consequent rollback situations. Due to the characteristic of confining the strict upper bound of the number of events that can be executed by a processor between interprocessor communications, this scheme is a static method of controlling computation granularity.

3.2.2 Balancing Progress by Virtual Time Window (BPVTW)

Unlike the BPEC scheme, the BPVTW does not have the strict upper bound of execution chances for each logical process before interprocessor communication. Rather, the relative progress of logical processes is controlled with regard to the timestamp of events. That is, the scheduler prevents a process from executing events if the process is far ahead of the other processes in virtual time. For this purpose, we employ a virtual time window whose base is the GVT such that logical processes can execute only events having timestamps within the interval of the virtual time window. This virtual time window is called the *Balancing Progress Window* and the size of the window is denoted by BPW. Thus, the Balancing Progress Window has the interval $[GVT,\ GVT + BPW]$.

In this scheme, the overall progress is controlled by each logical process not by each processor. As in the BPEC scheme, the processor scheduler selects the smallest timestamped event from the central scheduling queue. The scheduler, however, does not count

the number of executed events. When a logical process has an unprocessed event and its timestamp is in the interval $[GVT,\ GVT + BPW]$, the event is enqueued into the central scheduling queue. If there is no unprocessed event whose timestamp is in the interval $[GVT,\ GVT + BPW]$, the process does not have any chance to execute events until the next cycle after communication. In other words, the scheduler handles events until there are no available unprocessed events in the scheduling queue. Once the scheduling queue is empty, the processor communicates with other processors.

The computation granularity in the BPVTW scheme is indirectly controlled by the BPW value. Since there is no static limitation of the maximum number of events to be executed by a processor per communication, we do not employ the MBS. Instead, the BPVTW scheme concerns the real progress of processes with their timestamps in order to balance the progress of logical processes. The computation granularity is consequently adjusted when the scheduler restricts scheduling events according to the BPW value. Because there is not a limited number of executed events, the actual computation granularity may vary depending on the progress of the simulation. In this sense, the BTVTW scheme is considered a dynamic method of controlling the computation granularity.

A similar concept has been proposed as *Moving Time Window* (MTW) (Sokol et al. 1989). Although the MTW mechanism didn't provide considerable improvement on some cases (Fujimoto 1990), as it will be shown from the experimental results in Section 4, the BPVTW scheme achieves a good performance improvement for large-scale and small-event-granularity applications on parallel and distributed systems due to its ability to control the computation granularity. In addition, for those applications with a large LP ratio, the BPVTW scheme provides the balancing effects as well as control over the degree of optimism.

4 EXPERIMENTAL RESULTS

This section presents the experimental results on a cluster of six DEC Alpha workstations interconnected by a DEC GIGASwitch through FDDI. The DEC Alpha 3000 workstation has a 133 MHz clock rate and a 32 MB memory. The GIGAswitch supports a peak data transfer rate of 200Mbits per second. As benchmark circuits, we use several circuits from the ISCAS89 benchmark suite. In the circuits, we have the D f/f clocking interval set to be the same as the input clocking interval. The logical processes are randomly partitioned into processors. For parallel processing and the interprocessor communication on the distributed system, we use the PVM (Parallel Virtual Machine) 3.3 (Oak Ridge National Laboratory 1994). Time Warp is implemented on the distributed system as a master-slave model.

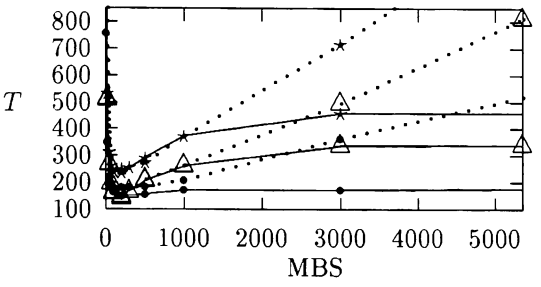

Figure 1: Total Execution Time (T) with BPEC Scheme for S38584 (Curves with Stars), S35932 (Curves with Triangles), S38417 Circuits (Curves with Dots)

4.1 Performance Measurement with the BPEC Scheme

This subsection explains the experimental results when the BPEC scheme is applied as the event scheduling technique to Time Warp on the distributed system.

4.1.1 Effects of the MBS

To study the effect of varying the MBS, we have performed experiments with three different circuits. Figure 1 shows the total execution time for the S38584 (curves with stars), S35932 (curves with triangles), and S38417 (curves with dots) circuits as a function of the MBS. The solid curves represent the case of $BPC = 1$ and the dotted curves show the case of $BPC = \infty$. The number of input vectors is 50, and the BPC is set to 1 and the unlimited value. The three circuits have the following LP ratios; 6733, 5357, and 6074, respectively.

As we mentioned in the previous section of the BPEC scheme, the computation granularity is controlled by values of the MBS and the BPC. To focus on only the effects of the MBS, in this subsection we consider only the dotted lines where the BPC has the unlimited value in Figure 1. As shown by the dotted lines, the total execution time can be minimized when the MBS is tuned properly. When the MBS is near to 1, the total execution time increases very sharply as the MBS decreases. In this situation, the computation granularity determined by the MBS is so small that the simulation spends most of the time in the frequent communications. It implies that

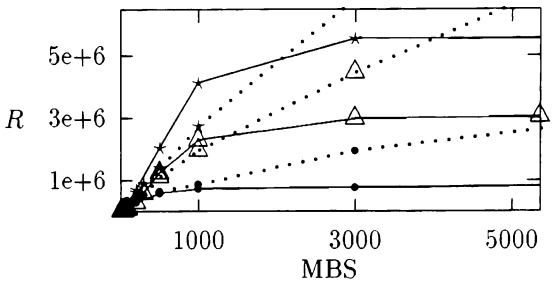

Figure 2: Number of Rollbacks (R) with BPEC Scheme for S38584 (Curves with Stars), S35932 (Curves with Triangles), S38417 Circuits (Curves with Dots)

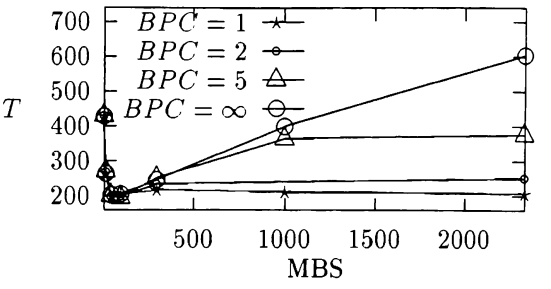

Figure 3: Total Execution Time (T) with BPEC Scheme for S13207 Circuit

the MBS should be larger than such a small value and that a substantial amount of computation could be performed between interprocessor communication points. The appropriate MBS will minimize the degrading effects of the communication overhead. In the extreme case of MBS = 1, the computation granularity is the same as the event granularity.

On the contrary, the total execution time increases linearly as the MBS increases after 200, approximately. It is because a large computation granularity may increase the number of rollbacks. To observe this relationship, Figure 2 shows the number of rollbacks varying the value of the MBS. Again, the dotted lines are for the case when the BPC is set to the unlimited value and the solid lines are when the BPC is set to 1. As mentioned before, we focus on the dotted lines in this subsection to focus on the effect of the MBS. In the dotted lines, the number of rollbacks is very small when the MBS is near to 1 because the overall simulation advances in a conservative fashion. As the MBS increases, the number of rollbacks increases almost linearly. This result implies that the larger the computation granularity is, the higher the probability is that immature events are propagated to other processors.

As a consequence, the appropriate value of the MBS should be as small as possible, but larger than a certain value so that the communication overhead can be reduced as much as possible. As for the cases in Figure 1, the optimal performance occurs when the MBS is around 200.

4.1.2 Effects of the BPC

In this subsection, we consider all the curves in Figure 1 to study the effects of the BPC. When the MBS is much larger than the optimal value, the simulation performance depends on the BPC value. Figure 3 shows the simulation results with the S13207 circuit. In the simulation, 100 input event vectors are used. Like the circuits in Figure 1, the curves in Figure 3 also reach steady performance when the BPC has small values, such as 1, 2, and 5. At this steady state, the BPC value controls the computation granularity because of the limited number of executed events per process. With a small BPC, processes have limited chances to execute events although they have more unprocessed events. Thus, a fast process is prevented from executing too many events so that the process cannot go too far ahead of the others, and we are then able to obtain the balancing effect with a small BPC value. In the interval between the optimal and the steady states, the computation granularity is controlled not only by the BPC, but also by the MBS. The depth of the valley of each curve at the optimal performance is deeper as the value of the BPC becomes larger. As shown in Figure 3 for the S13207 circuit, the simulation results with the BPC = 1 produce better performances than when the BPC is larger.

4.1.3 Effects of the LP Ratio

In order to study the effects of the LP ratio, we stage a set of simulations with three variations of the S35932 circuit. The experimental results are given in Figure 4. The D35932 circuit and the T35932 circuit are the circuits that are constructed by connecting the original S35932 circuit double and triple times, respectively. Therefore, the T35932 circuit has around a hundred thousand logic gates. Since the two circuits have logical gates multiple times of the S35932 circuit, they have multiple times higher LP ratios than the S35932 circuit. For all three curves, the BPC is fixed at 1. As before, each curve shows the optimal and the steady performances as the MBS varies. However, unlike the S13207 circuit which has the steady performance with the optimal performance on all the ranges

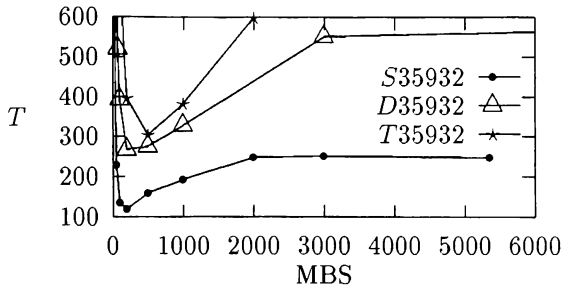

Figure 4: Effect of Granularity on Total Execution Time (T) by Varying LP ratio

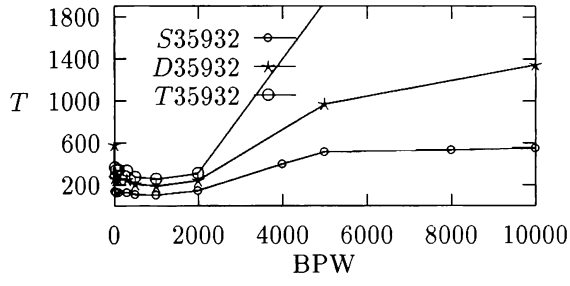

Figure 5: Total Execution Time (T) with BPVTW Scheme for S35932

of MBS when the BPC is 1, the steady performances of the three circuits in Figure 4 are much higher than their optimal performances. The larger the LP ratio is, the larger the difference is. Interestingly, even if tens of thousands of logic gates are assigned in a processor, the simulation shows reasonably good performance on distributed systems if the MBS is tuned appropriately. These results imply that the effects of computation granularity on the performance become more significant as the LP ratio becomes large enough.

4.2 Performance Measurement with the BPVTW Scheme

Additional sets of experiments have been performed with the BPVTW scheme on the distributed system. Figure 5 presents the experimental results in terms of the total execution time, varying the BPW. The three circuits, S35932, D35932 and T35932, are used, which are described in Section 4.1.3.

In the figure, all three circuits show good performances up to the BPW value of 2000. Unlike the BPEC scheme which shows sharp increases in total execution time for small values of BPC, the BPVTW scheme shows good performance even with very small values of BPW. It is because each process adds the unit delay of virtual time in executing events, and many events exist in the narrow virtual time interval. The interval of BPW which contains the optimal performances is quite wide in comparison to the sharp optimal range in the BPEC scheme. Thus, the progress balancing between logical processes with the virtual time window is effective and works well in the wide interval of BPW. This is because the BPVTW scheme concerns the actual progress of processes with their timestamps.

On the contrary, when the BPW is larger than 2000, the curves of total execution time increase with different rates as the BPW increases. In this larger window interval, the virtual time window does not take an important role of balancing progress of logical processes because there are many executed events in this interval. Also, the computation granularity is not restrained by the BPW. Thus, too many unprocessed events are executed, propagating lots of immature events. The larger the LP ratio is, the more unprocessed events are generated. In the figure, the T35932 circuit shows the worst performance when the BPW is larger than 2000. The D35932 circuit is worse than the S35932 circuit. Therefore, when the BPVTW scheme is used as the event scheduling method, the BPW must be tuned properly. Otherwise, the simulation performance will be as bad as if there were no limit of the MBS and the BPC in the BPEC scheme.

As discussed in Section 3, the BPVTW scheme does not control the computation granularity in a direct way as in the BPEC scheme. Rather, by using the virtual time window per logical process, the computation granularity is indirectly controlled while the progress of logical processes are balanced. In order to observe the effects of the BPW on the computation granularity, the average number of executed events between interprocessor communications are also shown in Figure 6. In the figure, where the simulation shows the almost optimal performance, the average number of executed events per communication is around 400 or 700, depending on the circuit. This amount is similar to the optimal grain sizes for the same circuits as shown in Figure 4.

5 CONCLUSION

In this paper, we propose two efficient event scheduling schemes on the optimistic protocol: the BPEC scheme and the BPVTW scheme. The BPEC scheme limits the number of executed events per logical process between interprocessor communications by using the BPC parameter. The BPVTW scheme controls

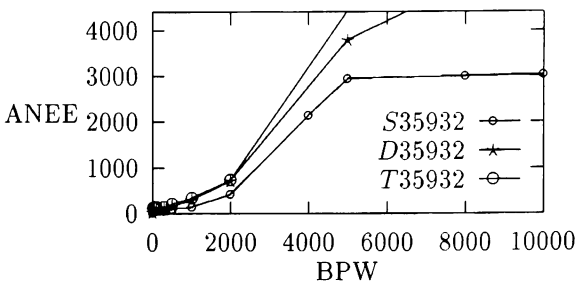

Figure 6: Average Number of Executed Events ($ANEE$) with BPVTW Scheme for S35932

the progress of processes in virtual time by using the BPW parameter, and prevents processes from executing events far ahead of other processes.

We experimented with the DOP considering the design issues of the control of the computation granularity and the event selection schemes on a cluster of DEC Alpha workstations. According to our experimental results, both schemes show good performances when the computation granularity is adjusted properly. Otherwise, communication overheads or excessive rollback situations may significantly degrade the simulation performance. The results also showed that the progress balancing among logical processes have significant effects on obtaining the optimal performance. In addition, it is shown that the LP ratio and the characteristic of the simulated circuit affect the simulation performance.

REFERENCES

Brown, R. 1988. Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem. *Communications of ACM*, 1220–1227.

Carothers, C. D., R. M. Fujimoto, and P. England. 1994. Effect of Communication Overheads on Time Warp Performance: An Experimental Study. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, 118–125.

Choi, E., and M. J. Chung. 1995. An Important Factor for Optimistic Protocol on Distributed Systems: Granularity. In *Proceedings of the 1995 Winter Simulation Conference*, 642–649.

Chung, Y., and M. J. Chung. 1991. Time Warp for Efficient Parallel Logic Simulation on a Massively Parallel SIMD Machine. In *Proceedings of the Tenth Annual IEEE International Phoenix Conference on Computers and Communications*, 183–189.

Das, S, R. M. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. 1994. GTW: A Time Warp System For Shared Memory Multiprocessors. In *Proceedings of the 1994 Winter Simulation Conference*, 1332–1339.

Fujimoto, R. M. 1989. Time Warp on a Shared Memory Multiprocessor. In *Proceedings of 1989 International Conference on Parallel Processing* 3:242–249.

Fujimoto, R. M. 1990. Parallel Discrete Event Simulation. *Communications of ACM* 33:30–53.

Gafni, A. 1988. Rollback Mechanisms for Optimistic Distributed Simulation Systems. In *Proceedings of the SCS Multiconference on Distributed Simulation* 19:61–67.

Jefferson, D. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7:404–425.

Jones, D. W. 1989. Concurrent Operations on Priority Queues. *Communications of ACM*, 132–137.

Preiss, B. R., and W. M. Loucks. 1995. Memory Management Techniques for Time Warp on a Distributed Memory Machine. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 30–39.

Sokol, L. M., B. K. Stucky, and V. S. Hwang. 1989. MTW: A Control Mechanism for Parallel Discrete Simulation. In *Proceedings of the 1989 International Conference on Parallel Processing* 3:250–254.

The Oak Ridge National Laboratory. 1994. *PVM 3 User's Guide and Reference Manual.*

AUTHOR BIOGRAPHIES

EUNMI CHOI is a Ph.D. candidate in computer science at Michigan State University. Her current research interests include parallel asynchronous protocols, parallel logic simulation on parallel and distributed systems, and parallel and distributed algorithms. She received an M.S. in computer science from MSU in 1991, and a B.S. from Korea University in 1988. She is a member of ACM and the IEEE Computer Society.

DUGKI MIN received a B.S. degree in industrial engineering from Korea University in 1986, an M.S. degree in 1991 and a Ph.D. degree in 1995, both in computer science from Michigan State University. He is currently an Assistant Professor in the Department of Computer Engineering at Konkuk University. His research interests include parallel and distributed computing, distributed multimedia systems, and computer simulation.