OPTIMISTIC COMPUTATIONS IN VIRTUAL ENVIRONMENTS

Maria Hybinette Richard Fujimoto

{ingrid,fujimoto}@cc.gatech.edu

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280, U.S.A.

Keywords: Interactive Parallel Discrete Event Simulation

ABSTRACT

Virtual environments often require computationally intensive image computations that cripple efforts to provide real-time animation. The problem is that physically accurate images require at least seconds and sometimes hours to compute. It is clear such latencies are unacceptable in virtual environments where images must be generated in real time.

We present an optimistic I/O *mechanism implemented in a parallel simulation executive to facilitate the generation of anticipated event sequences. An associated roll-back compensatory scheme enables recovery whenever an unexpected real-time input forces the cancelation of scheduled operations.*

A prototype implemented on a shared memory multiprocessor employing off-the-shelf computationally intensive software, Synthetic Scene Generation Model (SSGM), demonstrates the application of this approach in a virtual environment.

1 INTRODUCTION

The high computational demands of phenomenologically correct image generation limits its use in real-time environments. These types of images often take minutes or sometimes hours to compute. This is a a problem when users or devices interacting with the environment require different images than those that were anticipated. A result is that there may be a significant lag between the interaction and the appearance of a correct image.

By initiating computations in advance, using a faster than real-time simulation system, one can overcome the latency in generating such image sequences. Abort operations may be activated if unexpected interactive inputs arrive to cancel time-consuming operations that are no longer needed.

Bringing virtual environments into interactive and dynamic federated simulations raises two challenges: One is to reduce the time required to generate individual images. The second is to hide the latency of performing these computations. The first issue is not addressed here. Even so, reduced image generation time alone is often insufficient to create realistic animation sequences because it may still not be feasible for real-time generation of frames. However, a faster-than-real-time simulation can execute ahead of time to determine what images will be needed later to hide the latency associated with image generation.

An optimistic parallel simulation system is well suited for this task. Parallel simulation techniques can enable many simulations to execute faster than real time. Further, optimistic simulators provide a roll back capability, so they can deactivate unwanted image generation that have been initiated but is no longer needed. Unexpected interaction can thus be dealt with when a Time Warp system is used in conjunction with the virtual environment. In this report we evaluate the effectiveness of this approach in a practical simulation application. Another motivation for this research is the *re-use* of existing software by independently spawning specified computational modules. The same capabilities may be used in other domains like robotics where motor control commands in a multi-joint robot require significant amount of computations, but where the presence or absence of obstacles may change which computations are necessary.

An Interactive Parallel Discrete Event Simulation System (IPDESS) is implemented to drive the generation of computationally intensive images for real time animations. A missile defense application is used to illustrate the approach. In this application the goal is to test missile defense strategies in realistic combat scenarios involving human-in-the-loop, hardware-in-the-loop, or computation only simulations. For example, an image sequence depicting a missile launch may be activated when a user clicks a mouse key. Subsequent images may include elements such as missile hard-body and missile plume. Another user may want to interact with the active animation in progress by pressing another button to activate an intercept missile. The images may include multiple missiles: the intercept missile and the target missile. Later in the simulation some images may depict either a hit or a miss of the intercepting missile with the target missile.

The paper is organized as follows: The next section describes background and related work concerning interactive parallel simulations. Section 3 presents the optimistic I/O mechanism. This is followed by an example highlighting the benefit of our mechanism. Next, Section 5 discusses the implementation of the mechanism in a Time Warp executive. We then describe the off-the-shelf software used to generate images. The resulting software architecture and its implementation are discussed in Section 7 and 8. We conclude with a summary and a discussion of future directions.

2 BACKGROUND

In an interactive animation environment I/O events direct the display of individual images; user interaction may initiate an unexpected image sequence. To illustrate the problem of integrating an off-the-shelf computer graphics engine into an action-oriented real-time environment, this report considers an animation package that require considerable computation time for individual frames.

To hide image generation times we exploit parallel discrete event simulation (Fujimoto 1990). In a nutshell, distinct components in a parallel discrete event simulation are modeled by logical processes and the simulation progresses as LPs exchange time-stamped event messages that may cause changes in the system state at discrete points in time.

A synchronization mechanism is required to ensure that each LP processes events in time-stamp order. The two prevailing classes of synchronization protocols are called *conservative* and *optimistic* mechanisms. The conservative protocol enforces consistency by avoiding the possibility of an LP ever receiving an event from its past (as measured in simulated time). LPs *wait* to process events until reception of an out of order event is impossible (Chandy and Misra 1979; Bryant 1977). The optimistic protocol, in contrast, uses a detect-and-recover scheme. When an event is received in the past of an LP it recovers by rolling back previously processed events with later time-stamps than the one that was just received. Since the optimistic approach has roll back capability, it can recover from unwanted computations such as image generation requests.

The prevalent optimistic approach utilizes the Time Warp mechanism originally presented in (Jefferson and Sowizral 1982). LPs process incoming messages optimistically without blocking. If a logical process later receives a “straggler” message that has a lower time stamp than its current simulation time it rolls-back all processed events with a later time stamp than the straggler. The straggler and any rolled-back events are then reprocessed in time stamp order. Roll-backs involve sending anti-messages that cancel or “annihilate” the original messages. To enable roll-backs Time Warp systems keep a copy of processed messages in an event queue. Messages are kept in the queue until they are ensured not to be subject of any future roll-back. A message is guaranteed not to be subject to a future roll-back if it is earlier than the smallest time stamp of any unprocessed or partially processed message in the system. This lower bound is referred to as the global virtual time or GVT of the system. GVT is used to commit I/O operations and to reclaim memory for events (called fossil-collection). Any event with a time-stamp lower than GVT is guaranteed to never be rolled back and can be committed, so its I/O operations can be performed and the memory for that event can be reclaimed.

Conventional Time Warp systems provide a mechanism for conservative I/O (Jefferson 1985). We use a Time Warp executive called Georgia Tech Time Warp (GTW). In GTW two types of conservative I/O are available: nonblocking and blocking I/O events (Das et al. 1994). A logical process can execute optimistically beyond a nonblocking I/O event because a nonblocking event does not effect the state of the simulator. Nonblocking events can be used for output events. Blocking I/O events however, prevent the optimistic execution of an LP. Because these events affect the state of the logical process, an LP blocks the execution of other messages with time stamps later than the I/O blocking event. The logical process cannot execute until the I/O event is either processed or canceled. Once GVT progresses beyond a blocking I/O event and the I/O event is processed the logical process becomes unblocked. Blocking I/O events can be used for input.

There is an important difference between committing an I/O event in Time Warp and displaying an image in an animation. Display events are related to real-time, but Time Warp I/O events are related to GVT. Since Time Warp is faster than real time, a new type of event, called an *optimistic* I/O event, is proposed to hide the latency of generating images. The idea is that optimistic I/O events can request an image before it is needed in real-time. The I/O events are referred to as optimistic because there are some cases (due to real time user input) where the scheduled events must be canceled. The simulation aborts time-consuming requests such as image generation that are no longer needed by activating anti-computations called roll-back handlers. The roll-back handlers are user defined. Thus, the actual display request of an image can be undone or changed. Unlike conservative I/O events which cannot be undone, optimistic I/O can be canceled or rescheduled. This will be discussed in detail in section 3.

In contrast to analytic simulations, virtual environments include either human interactions or hardware realizations as an integral part of the simulated system. Virtual worlds require real-time execution as opposed to analytic simulations that often proceed faster than real-time. Virtual environment simulations often exploit limitations of human perception. For example, events happening close together in time can often be processed in any order since a human cannot perceive the actual ordering. An example of the integration of an analytical protocol with virtual environments is the Aggregated Level Simulation Protocol (ALSP) (Wilson and Weatherly 1994). The motivation behind the ALSP was to extend distributed virtual environments to the analysis of war-gaming. ALSP processes events in strict time-stamp order. This is accomplished using the Chandy/Misra/Bryant conservative protocol mentioned earlier.

Interactive parallel discrete event simulation work by (Franks et al. 1997; Ghosh et al. 1993) differers from ours in that they do not use compensatory operations to prevent long-running operations. Further we deliberately avoid the phenomena of I/O thrashing described in the next section. In databases, compensatory transactions are used as an alternative to strictly un-do the effect of long-running operations while avoiding cascading rollback (Garcia-Molina and Salem 1987; Korth et al. 1990). Our approach is inspired by compensatory operations in databases to enable the use of off-the-shelf software in virtual environments. We use optimistic simulation techniques to hide the latency associated with time consuming operations. The next section gives details of our approach.

3 OPTIMISTIC I/O MECHANISM

Figure 1 shows the mechanics of a Time Warp simulation. The right image shows the communication topology of logical processes (LPs) participating in the simulation. The left image shows the progress in simulated time of the LPs. The vertical axis is simulated time, the horizontal axis represents each logical process (except the lower leftmost dark grey rectangle marked "RT" which

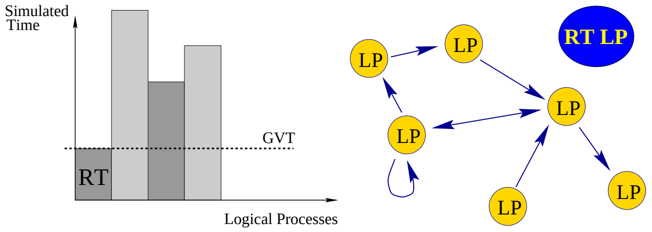

Figure 1: A Simple Solution for Real-Time Time Warp, each bar represents the progress in simulated time of a logical process in the simulation.

show the progress of wall clock time.) This graph relates the simulated time of the Time Warp system to wall clock time. GVT is defined as the lowest time stamp of any unprocessed message or partially processed message in the system. Since a Time Warp simulator progresses faster than real time, wall clock time is behind GVT. If a user is allowed to interact with the simulation, the event that represents the interaction should have a time stamp equal to wall-clock time. This would push GVT to an earlier time. In this situation GVT can not be used for fossil collection since it is possible for a rollback to occur that is earlier than GVT.

A simple solution to realize optimistic computing in a real-time environment is to include a new LP that paces interactive operations according to the real-time clock. Here, the simulator (excluding the new LP) is faster than real-time. As a result the new LP encapsulating interactive input/output includes the real-time clock and suspends GVT accordingly. The effect is that interactive events are released at GVT while other operations can proceed optimistically. This however, does not solve the problem with computations that are computationally intensive. So a second solution is needed.

To provide for computationally intensive operations the better approach provides a compensatory operation. Here, computationally intensive operations are allowed to be scheduled before its time stamp reaches GVT. However, if it is found that the operations are no longer needed a compensatory operation that aborts the request is activated. This is accomplished by piggy-backing a user-defined abort operation on the event that is instantiated upon rolling back the operation. There is a problem with this solution as well. The problem is that there may be a high occurrence of requests that are later aborted resulting in I/O thrashing. The solution can be improved to limit the optimism of these types of transactions.

To avoid I/O thrashing two GVTs are defined. One is called conservative GVT and the other is called optimistic GVT. Figure 2 illustrates the different GVTs. *Conservative* GVT is defined to be the minimum of wall-clock time and the current GVT. An event is fossil col-

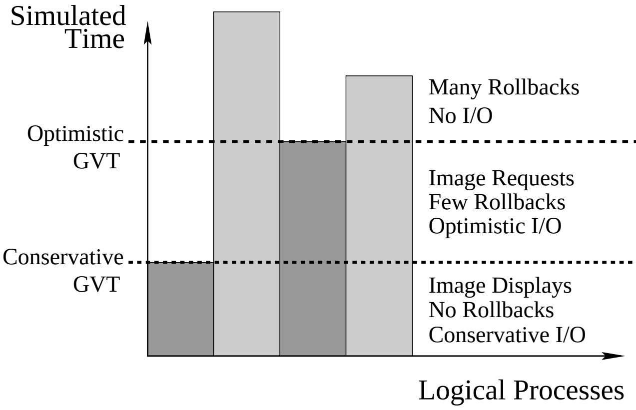

Figure 2: Optimistic and Conservative GVT

lected when its time stamp is lower than the conservative GVT since these event are guaranteed to never rollback. *Optimistic* GVT is the minimum of any unprocessed message or partially processed message. Optimistic GVT is the same as the conventional GVT in a Time Warp system when the LP encapsulating the wall-clock is excluded. Mirroring the GVTs there are conservative and optimistic I/O events. Conservative I/O events are executed as conservative GVT progresses beyond their time stamps. Similarly, optimistic I/O events are executed as the optimistic GVT sweeps their time-stamps.

With respect to a virtual environment, an image request message is an optimistic I/O event, it is buffered until its time stamp is beyond the optimistic GVT. When optimistic GVT catches up to these buffered messages they are processed and the image request is sent to the virtual environment coordinator. When a real-time user-input request is processed, a message is sent to the Time Warp simulator. The simulator responds by rolling back events that have a time stamp later than conservative GVT (includes wall-clock time.) If an optimistic I/O operation is undone, it aborts its corresponding image generator invocations by activating the anti-computation procedure associated with the optimistic event.

4 EXAMPLE TASK

To illustrate how an optimistic simulator can improve the performance of a physical simulation consider the task of intercepting a target missile. The intercept of a target missile has an unpredictable element, namely a hit or a miss. The unpredictable element can be introduced to the interactive animation by providing a push button to launch a target missile. An illustration of the scenario is shown in Figure 3. The idea is that a user pushes a button to activate the launch of a target missile. Next, another user activates an intercept missile aimed for the target missile. The result is either a hit of a miss depending on when the users pushed their buttons. When

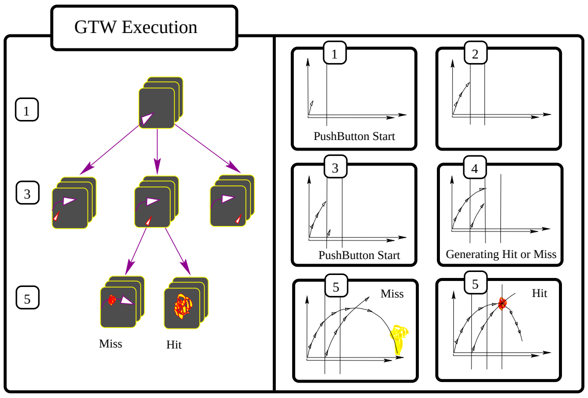

Figure 3: Example Scenario

the user pushes the button to launch the intercept missile, Time Warp will first cancel images in progress that do not contain the intercept missile. Also Time Warp will start requesting images that contain both missiles. As the simulation draws closer to the concluding image (the hit or miss,) Time Warp requests the generation of images which display either a hit or an miss. If a hit occurs, the most likely one, a miss, may already be in progress and thus is canceled. When the actual event occurs and is displayed however, the hit or the miss image will already have been generated and can thus be immediately displayed.

5 OPTIMISTIC I/O IMPLEMENTATION

The optimistic I/O mechanism is implemented using Georgia Tech Time Warp (GTW). While discussing the implementation, it is helpful to introduce a few details of GTW. GTW provides procedures for message passing, for example non-I/O messages utilizes the `TWSend` procedure. In implementing optimistic I/O messages GTW provides a `TW_OIOSend` procedure. `TW_OIOSend` differs from `TWSend` in that a different event handler can be specified for each event. This is similar to the two existing conservative send primitives, but `TW_OIOSend` has two event handlers associated with it, one which is activated when the event is processed and one that is activated if the event is rolled back or canceled. With respect to the example application the `TW_OIOSend` procedure can be defined to "request" the generation of a image with specified parameters. For example the parameters may specify the take-off of a target missile at a certain geographical location.

As previously mentioned I/O messages are scheduled for processing when their time stamps are earlier than GVT. In contrast to conservative GVT optimistic

GVT is not a monotonically increasing function, because an external interaction may set back the optimistic GVT to an earlier time stamp.

In order to delay the processing of optimistic I/O messages until their timestamps are earlier than the optimistic GVT, each processor maintains a priority queue holding unprocessed conservative nonblocking I/O events and unprocessed optimistic I/O events. I/O events are added to this queue as they become the next pending event. A procedure which is called immediately after a new optimistic GVT has been computed, ensures queued optimistic I/O events with time stamps less than or equal to the optimistic GVT are processed. The processed message is then put in the respective LP's processed event queue in case the message must be rolled back. If a processed optimistic I/O event is rolled back a rollback function for that event is activated. GTW refers to the parameters of the original processed event to the rollback function. Hence, resources can be conserved in the interactive environments if the users specifies that image generations are canceled when their corresponding events are rolled back. The parameter to the rollback function in this case would be the corresponding instance that requested the generating the image. The algorithm is listed below:

1. LP sends optimistic I/O message via `TW_OIOSend` primitive.
2. The sending LP of an optimistic I/O message enqueues the message directly into message queue corresponding to the destination LP (GTW provides one message queue per processor.) If the destination and sending LP are mapped to the same processor the message is enqueued directly into a pending queue.
3. If a processor dequeues an optimistic I/O event from its pending queue the event is put in the processor's optimistic I/O holding queue.
4. When optimistic GVT is computed, the events in the I/O holding queue are processed if their timestamps are earlier than the optimistic GVT.
5. If a processed optimistic event is later canceled or rolled back, due to an external event, a specified rollback function is activated.

GTW launches a separate thread called EX to monitor external input events or to output an external event. When a user pushes a button for example, the request goes through the EX thread. In turn the EX thread routes the external event to the appropriate LP.

6 SSGM

In order to demonstrate the capabilities of optimistic I/O the Synthetic Scene Generation Model (SSGM) software package developed by the Naval Research Laboratory (NRL) is used to generate images for a virtual environment. SSGM generates high quality image sequences for Ballistic Missile Defense Organization (BMDO) scenarios. SSGM is important to researchers because it produces images that are physically correct with great accuracy, such accuracy is needed because the same set-up is used to test actual sensors (hardware-in-the-loop). The computed trajectories, plumes and clouds have complex mathematical and time consuming algorithms associated with them. Commercial graphic packages can not generate images near the quality of SSGM images. SSGM produces physically correct optical images for development of Strategic Defense Initiative Hardware (SDI). SSGM is primarily used to predict the performance of optical sensor systems that are designed to track the detection, acquisition and discrimination of targets.

An SSGM image contains three elements, the background, the foreground and the observer element. The elements are specified via parameters in an ASCII file called the Scenario Definition File (SDF). Several phenomenologies exist for each element. For example, a user can select among several missile and plume phenomenologies for the foreground element. Background components may be terrain from various geographical locations or/and specific cloud formations. In contrast to the background and foreground elements the observer element has one single phenomenology, called the Observer. Location and framing mode are parameters among others that must be specified for the Observer. The interface to the SSGM server is thus the Scenario Definition File which parameterizes the use of the phenomenologies.

Even though SSGM provides high-quality images, it cannot be used effectively in a real-time environment because:

- image generation takes to long (high latency).
- image sequences must be specified in advance and cannot be changed once the simulation has started. This inhibits scenarios such as missile intercept to be created dynamically by a user or hardware.

The optimistic I/O mechanism primarily addresses the latter issue.

7 SYSTEM ARCHITECTURE

This section describes the system architecture of the integrated SSGM and GTW system for high-speed simu-

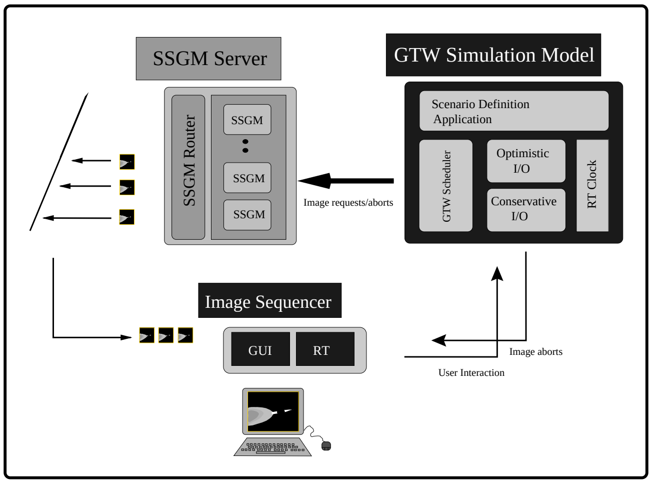

Figure 4: The Interactive Parallel Discrete Event Simulation System

lation. The architecture is composed of three modules: The SSGM Server module, the GTW Simulation module, and the Image Sequencer Server module. The system is depicted in Figure 4.

GTW communicates with the SSGM module by sending image request or image abort messages. Image abort messages cancel images which may be in progress but no longer needed. Image request messages are sent as a result of an optimistic I/O event in Time Warp. The events are "optimistic" because they request images likely for future display. There are situations, however, when the predictions are wrong. In these situations the simulation needs to undo the incorrectly predicted event otherwise computing resources may be wasted. Time Warp preempts the image generation by activating a recuperate event which aborts the respective unwanted SSGM computation. Note that the ability of an optimistic I/O event to recover from unnecessary output results further differentiates optimistic I/O events from conservative I/O events.

The interface to SSGM is an ASCII file called the Scenario Definition File (SDF). The SDF describes the image phenomenologies by parameter. An optimistic message thus includes a scenario definition file generated by the GTW executable. The delay in producing an image within SSGM can be reduced by parallelizing the computation of each phenomenology. This effort has been undertaken by NRL. In our research at Georgia Tech we consider SSGM as a black box, accessible via sending request or abort messages for image generation.

The image sequence module is responsible for displaying the correct image at the correct time. This is done by encoding the frame number into the name of the resulting file.

Figure 5: The Image Sequencer Interface

GTW drives the simulation by sending requests for image generation to SSGM. If a user interacts with the simulation, an external event is sent to GTW. GTW responds by aborting the corresponding optimistic I/O events and by requesting new image generation reflecting the user's interaction.

Image Sequencer Module

The image sequencer module is responsible for showing the correct image at the correct time. It also provides the user with push buttons that allow interaction with the simulation in progress. The current prototype has six possible interactions for the example task. Namely: START simulation, LAUNCH target, LAUNCH interceptor 1 through 3 (one launch for each interceptor), and STOP interaction. Figure 5 shows the user interface. When a user pushes any of the interaction buttons a message is send to GTW through a socket. The message contains the object (intercept, target or simulation), the action desired (start or stop,) and the current time. Time Warp interprets the message and then starts to generate future scenario definition files which accounts for the interaction.

SSGM Simulation Module

Even though we consider the SSGM as a black box, we have extended the SSGM module so that it behaves more like a server. The current model requires that only one scenario at a time can be processed or queued. In other words, SSGM does not queue requests. Further, if an image abort is desired a whole SSGM process has to be aborted even if only a few images are not needed. In order to circumvent the queuing problem we added a submodule that we call the SSGM scheduler. The SSGM scheduler, takes the requests from GTW then forwards the requests to an available machine. If no machine is available it queues the request and waits until a machine can fulfill the request. If an abort is requested however it looks in its queue and removes it if it is in the request queue otherwise it aborts the appropriate SSGM process in progress.

8 IPDESS IMPLEMENTATION

The Interactive Parallel Discrete Event Simulation System (IPDESS) was implemented using the Georgia Tech Time Warp version 2.5 (GTW_v2_5), the Synthetic Simulation Model version 6.0 (SSGM_v6_0), ImageMagick (IM), and a GTW scene parameter application. This section will describe the implementation of I/O operations in the GTW executive, the scene parameter application operating on top of GTW, and the graphical interface that provides for user interaction. GTW is the central manager of the overall system. It drives and controls the animation in progress by communicating with each module of the IPDESS system. Conversely, the modules communicate with GTW to request actions in the system.

The goal of reducing the image latency involves solving several implementation issues in the GTW kernel:

- Provision for optimistic and conservative I/O
- Provision of an optimistic rollback routine for each optimistic I/O event

The goal of the IPDESS is to hide image generation latency of SSGM. In it's current implementation however SSGM may take several hours to generate an image (efforts to reduce this are underway at NRL.) In anticipation of a faster SSGM, the IPDESS can use precomputed images with a delay to simulate processing, or if necessary, trigger actual SSGM computations.

The architecture of IPDESS was introduced in section 7. The connections between the Image Sequencer, the SSGM scheduler and GTW are sockets. For example, when GTW is initiated it launches the SSGM scheduler, and sets up the connection between them. It then waits for a connection to of an animation client (or the image sequencer.)

SDF Application Implementation

The SDF application is written using GTW Kernel primitives. The application uses for example `TWInitAppl`, `TWSend` and `TW_OIOSend` primitives. The application also defines the following event handlers: a target handler, three intercept handlers and a collector handler. Each of these handlers are associated with a Time Warp LP. The collector, target and interceptor each reschedule themselves for each frame period. If a target or an intercept receives a message from EX it changes it state to reflect the user request and then forwards this information to the collector. The collector sums the state of each object in the scenario, and sends optimistic requests to schedule SSGM instances. The collector also sends non blocking conservative I/O messages to ensure that the image sequencer and GTW are synchronized. The optimistic event handler schedules an SSGM event if there has been a change in state by sending a message to the SSGM scheduler, the event requests SSGM should, from now on, generate images reflecting the current specified state. If there is no change in state, the event handler returns. The reason that we trigger the optimistic event even if no request is send to the SSGM scheduler is to give a handle to individual frames in the event of a roll back. The roll back handler can thus can eliminate individual frames and tell the SSGM scheduler which images need to be aborted.

Image Sequencer Implementation

The interface for user interaction with SSGM simulation is provided in part by a public domain animation software package (ImageMagick). The package is enhanced so that it displays the correct image according to a real time clock. Buttons are provided to initiate target and intercept missile launch: as a button is pushed, a message is sent to alert the GTW module. The button pushes trigger or roll back optimistic I/O events in GTW. In order to address the high latency of SSGM, the image sequencer currently uses a library of precomputed images. The images are accessed by the state and the frame number of the scene. By state we mean all objects that could possible be visible. For example the target has not launched and there are currently no interceptor and the user has pushed the START simulation button the state is: SIM START, NO TARGET, NO INTERCEPT1, NO INTERCEPT2, NO INTERCEPT3. The image sequencer thus keeps track of its state and display the image that corresponds to the current state. As SSGM latency becomes reasonable and the images are available as they are generated, the animation tool need only undergo minor changes to show newly generated image instead images from the image library.

9 CONCLUSION AND FUTURE WORK

This report introduced an optimistic I/O mechanism able to hide latency in application software by scheduling processing long before it's output is needed. A rollback computation is coupled with optimistic request to recovery of resources if an unexpected real-time input forces the cancellation of the scheduled operation. The mechanism enables the re-use of existing software by independently spawning specified computational modules.

The mechanism is demonstrated by integrating a computationally intensive image generation application called SSGM into an interactive environment. A software architecture called IPDESS is proposed and implemented and contains three modules: the SSGM module, the GTW module and the image sequencer module.

Improvements to IPDESS include the implementation of a lazy cancellation scheme that reduces computation when rolling back optimistic I/O events. Another area of future research is to investigate parameters that effect the benefit of optimistic computing such as granularity of computations, frequency of roll-backs where performance measurements include average frame per second and available CPU.

REFERENCES

BRYANT, R. E. 1977. Simulation of packet communication architecture computer systems. MIT-LCS-TR-188, Massachusetts Institute of Technology, Cambridge, Massachusetts.

CHANDY, K. M. AND MISRA, J. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering SE-5*, 5 (September), 440–452.

DAS, S., FUJIMOTO, R., PANESAR, K., ALLISON, D., AND HYBINETTE, M. 1994. GTW: A Time Warp system for shared memory multiprocessors. In *1994 Winter Simulation Conference Proceedings* (December 1994), 1332–1339.

FRANKS, S., GOMES, F., UNGER, B., AND CLEARY, J. 1997. State saving for interactive optimistic simulation. In *Proceedings of the* 11^{th} *Workshop on Parallel and Distributed Simulation* (1997), 72–79.

FUJIMOTO, R. M. 1990. Parallel discrete event simulation. *Communications of the ACM 33*, 10 (October), 30–53.

GARCIA-MOLINA, H. AND SALEM, K. 1987. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (1987), 249–259.

GHOSH, K., FUJIMOTO, R. M., AND SCHWAN, K. 1993. Time warp simulation in time constrained systems. In 7^{th} *Workshop on Parallel and Distributed Simulation*, Volume 23 (May 1993), 163–166. SCS Simulation Series.

JEFFERSON, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July), 404–425.

JEFFERSON, D. R. AND SOWIZRAL, H. 1982. Fast concurrent simulation using the Time Warp mechanism, part I: Local control. Technical Report N-1906-AF (December), RAND Corporation.

KORTH, H., LEVY, E., AND SILBERSCHATZ, A. 1990. A formal approach to recovery by compensating transactions. In *Proceedings of the 16th International Conference on Very Large Data Bases* (August 1990), 95–106.

WILSON, A. L. AND WEATHERLY, R. M. 1994. The aggregate level simulation protocol: An evolving system. In *1994 Winter Simulation Conference Proceedings* (December 1994), 781–787.

AUTHOR BIOGRAPHIES

MARIA HYBINETTE is a Ph.D. candidate in the College of Computing at the Georgia Institute of Technology. Her research interests include parallel simulation, parallel algorithms, and real time systems.

RICHARD FUJIMOTO is a Professor in the College of Computing at the Georgia Institute of Technology. He is working on performance issues related executing discrete-event simulation programs on multiprocessor and distributed computing platforms.