NASA-CR-191796

Doc. No. 18-210062 REV -

VME Rollback Hardware for Time Warp Multiprocessor Systems.

Michael J. Robb and Calvin A. Buzzell
Integrated Parallel Technology, Inc.
5994 W. Las Positas Blvd. Ste 209
Pleasanton, CA 94588

25 September 1992

(NASA-CR-191796) VME ROLLBACK HARDWARE FOR TIME WARP MULTIPROCESSOR SYSTEMS Final Report (Integrated Parallel Technology) 51 p

N94-32483

Unclas

G3/62 0010554

Prepared for

NASA
National Aeronautics and Space Administration
Washington, D.C. 20546

JPL
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

RCCD
10/16/92

Table of Contents

1.0 Introduction 3

1.1 Time Warp and the RBC Module 3
1.2 Time Warp with and without RBC 4

2.0 Phase II Project Objectives 6

3.0 RBC Hardware Module Functional Overview 6

4.0 RBC Functional Requirements 8

4.1 Installed-vs- State Memory 8
4.2 Max. State Memory per Node 8
4.3 Max. Amount of State per Process 8
4.4 Max. Number of Processes and Memory Allocation 8
4.5 Minimum State Size 9
4.6 Dynamic Process Creation 10
4.7 Memory Fragmentation 10
4.8 Dynamic Process Destruction 10
4.9 Dynamic State Size Growth/Shrinkage 10
4.10 Number of Previous States Retained 10
4.11 State Memory Expansion Increments 11

5.0 Phase II Research Conducted 11

5.1 Systems Configuration 11

5.1.1 Sun 3 Configuration 12
5.1.2 Sun 4 Configuration 13

5.2 Vendor Survey 14
5.3 Time Warp Modifications 16

5.3.1 Xrbc 16
5.3.2 Rbc_test 16
5.3.3 Application Bank 16
5.3.4 Application Spot 17
5.3.5 Application Pucks 17

5.4 Functional Description and RBC Block Diagram 18

5.4.1 RBC Block Diagram Description 18
5.4.2 RBC Functions 21

6.0 Results Obtained 24

6.1 Sun 3 Single Node Testing 24

6.1.1 Effect of Cutoff on Elapsed Time 24
6.1.2 Effect of State Size on Elapsed Time 25
6.1.3 Copy Time vs Elapsed Time and RBC Savings 27

6.2 Sun 3 Muti-Node Testing With Bank 29

6.2.1 Effect of Cutoff on Elapsed Time 29
6.2.2 Effect of State Size on Elapsed Time 30
6.2.3 Copy Time vs Elapsed Time and RBC Savings 32
6.2.4 RBC vs Sun 3 Object Times 33
6.2.5 RBC vs Sun 3 Event Statistics 34

6.3 Sun 3 Muti-Node Testing With Pucks 35

6.4 Sun 4 Single Node Testing 38

6.4.1 Effect of State Size on Elapsed Time 38
6.4.2 Copy Time vs Elapsed Time and RBC Savings 41
6.4.3 Single Node Testing With Pucks 42

6.5 Bank Consolidated Performance 44

7.0 RBC Improvements 45

8.0 Conclusions and Estimate of Technical Feasibility 46

9.0 References 47

PROJECT SUMMARY

Purpose of the Research

The purpose of the NASA Phase II Project described in this report was to develop and demonstrate innovative hardware, to implement specific rollback and timing functions required for efficient queue management and precision timekeeping, in multiprocessor discrete event simulations. Powerful, general purpose, multiple processor computers offer the promise of significant speed ups for many large simulation problems. However, expensive clock synchronization algorithms are required for asynchronous simulation. These algorithms introduce substantial overheads that often completely negate the benefits of parallel execution. The research effort was to specifically evaluate the technical feasibility of building hardware modules which eliminate the state saving overhead of the Time Warp paradigm, used in distributed simulations on multiprocessor systems.

Description of the Research

Integrated Parallel Technology designed, built and tested four (4) Rollback Chip (RBC) Modules configured with up to sixteen megabytes (16 Mbytes) of state memory, and performed modifications to NASA-JPL's Time Warp Operating System (TWOS), to allow performance testing with existing simulations. The RBC modules were configured as standard VME modules and integrated with both a Sun 3/260 VME backplane, and a SPARC IPX Sbus via a commercial Sbus to VMEbus translator.

The RBC module was designed to allow over 256 processes, and up to 1 megabyte (Mbyte) of state per multiprocessor node, providing Time Warp applications the opportunity to significantly increase the amount of state able to be saved per process compared to current hardware limitations.

A series of timing tests were conducted to document the performance of the custom hardware and for use in the analysis of projected effects on Time Warp multiprocessor systems. Performance testing was accomplished on single and multi-node Sun 3/260s, running the discrete event simulations Bank, Spot, and Pucks.

Research Findings

The RBC hardware developed in this effort has demonstrated that modular hardware is able to virtually eliminate the state saving overhead of Time Warp, with overall speed up dependant on the state size and granularity of the simulation. Further, the hardware is able to provide users with applications requiring large amounts of state to incrementally add memory to the hardware to match their requirements.

Potential Applications of the Research

The RBC hardware provides significant speedup of large scale discrete event simulation problems and allow multiprocessors using Time Warp to dramatically increase performance. Further, Time Warp applications with RBC will be practical even with large numbers of processes and/or large amounts of state per process. For example, terrain objects in large simulations which include the environment require megabytes of state.

1.0 INTRODUCTION

The purpose of the NASA Phase II Project[1] described in this report was to develop and demonstrate innovative hardware, to implement specific rollback and timing functions required for efficient queue management and precision timekeeping, in multiprocessor discrete event simulation. A hardware mechanism to support state savings and rollback operations for Time Warp has been proposed and studied by Dr. Richard Fujimoto at the University of Utah [Fuj87, Fuj88].

The research effort described in this report was conducted by Integrated Parallel Technology, Inc. (IPT) in collaboration with Dr. R. Fujimoto, Georgia Institute of Technology, and Dr. S. Bellenot, Florida State University. The approach of the effort was to develop and demonstrate state saving and specific Time Warp operations on a custom hardware module, integrated in a VME architecture. IPT designed and built four (4) Rollback Chip Modules configured with up to sixteen megabytes (16 Mbytes) of state memory, and performed modifications to NASA-JPL's Time Warp Operating System (TWOS) to allow performance testing with existing simulations. Performance testing was accomplished on single and multi-node Sun 3/260's and a single node Sun SPARC IPX, running the discrete event simulations Bank, Spot, and Pucks.

1.1 Time Warp and the Rollback Chip (RBC) Module

The Time Warp mechanism speeds up object-oriented discrete event simulation by providing a synchronization protocol for multiprocessor computer systems [Jef82, Jef85]. Time Warp divides a simulation into multiple objects running on separate nodes which schedule events for each other. Events are scheduled for a particular virtual time and these events must be executed in the correct order. Time Warp is optimistic, and always runs the event with the earliest virtual time on each of its parallel nodes. If an event (message) arrives in the past, Time Warp will roll back the target object to the appropriate time to correctly execute the new event. In order to allow an object to roll back to a previous virtual time, old copies of the object's state, its collection of variables, need to be saved. In TWOS, NASA-JPL's version of Time Warp, a copy of each object's state is saved after every event. This state copying overhead is not required if the simulation was executing sequentially, in virtual time order, on one node.

The Time Warp protocol allows for significant speedup in execution of applications involving small states. However, the algorithms associated with state management introduce substantial overheads that often completely negate the benefits of parallel execution [Fuj88, RMM88]. Jefferson acknowledges that this is a major stumbling block and estimates that approximately one-third of the system overhead in JPL's implementation of Time Warp is state saving. This fraction can be expected to rise as other overheads in the implementation (e.g., interprocessor communication) are reduced. Also, the tested applications contain only a modest amount of state (at most only four thousand bytes), and copy the entire state vector of each process after each event. Attempts to minimize the effect of state saving overhead on performance have resulted in applications specifically written with small state sizes and/or large granularity, and in proposed modifications to the state saving algorithm, such as state skipping and periodic state saving, at the cost of increased run times and complexity [Fuj89, Bel92].

The RBC module was designed to off load the overhead associated with Time Warp state saving in a manner which was transparent to the applications programmer and with minimal overhead cost to the simulation, while retaining the speed up benefits of Time Warp.

[1] Work performed under NASA contract NAS7-1102, April 1990 to August 1992.

Conceptually, the Rollback Chip (RBC) module is a specialized memory board which is added to systems running Time Warp, or other applications which require previous events and states to be retained for later recovery in simulation time. The RBC looks like regular state memory to a simulation object or process, with the exception that it keeps multiple copies (up to 64) of each location (variable) in state memory. These previous copies of the state are not visible to the simulation and the RBC controls which version of a location is provided to the node, based on information provided by Time Warp. Each RBC module provides the necessary memory and control hardware to off load the overhead associated with state saving, rollback and recovery, for a single node of a multiprocessor. Because the node's state memory now resides on the RBC rather than as part of the node memory space, this effectively frees all memory previously in the processors local memory for tasks other than state saving. Further, the RBC memory is also expandable which facilitates incremental growth for simulations with large states.

For a more detailed discussion of the Time Warp paradigm and its performance see Section 8, References [Jef82, Jef85,and Jef87]. The concept of hardware support for Time Warp and development of a "Rollback Chip" is discussed in references [Fuj87,Fuj88].

1.2 Time Warp With and Without RBC

Performance gains realized through the use of the RBC module over standard Time Warp implementations can be summarized in three broad categories:

(1) State Saving Overhead

In standard Time Warp implementations state saving is time consuming due to the requirement to copy the current state to a new state area on every "mark" or state save operation. Consider a process which has just completed an event and must save its state. Prior to starting the new event, memory is allocated for its state vector (the previous memory area cannot be overwritten since it is now "saved" and cannot be modified). However, data which is in the just saved state is needed by the new event and it is possible that this new event will need a piece of data which was last updated many states previously. Thus on each allocation of a new state block, the entire previous state must be copied to the new state area prior to execution of the new event.

As an example, for a single MC68020 processor configured with 32 Kbytes of state and a memory access time of 500 ns (for 4 bytes), 8 milliseconds are required to copy the state block on each Mark operation. For a typical simulation, such as Bank, executing 20,000 events (equating to a cutoff of 5000 in section 6) a total of 160 seconds are required for state saving alone. Depending on the granularity of the simulation (i.e. amount of time spent in each event between saves), and the amount of non-state saving overhead (e.g. communication overhead), state saving overhead can consume over 90% of the processor time.

Addition of the RBC module effectively removes all of the state saving overhead since no copying is performed. Assuming a perfect implementation of the RBC module (i.e. no overhead imposed by the RBC), the savings for a simulation would exactly equal the state saving time of the node. Section 6, Performance Results, provides graphical presentation of the efficiency of the RBC in capturing the state saving time.

(2) Memory Management Overhead

In NASA-JPL's version of Time Warp, run time memory is allocated from a single heap consisting of several megabytes of memory. For the Sun 3 implementation the standard heap size is 2 megabytes (heap size is user selectable). Allocation of memory during run time is generally contiguous in memory at start up but becomes increasingly fragmented as the simulation progresses, due to the "sharing" of the physical memory space with other processes, and local variables, etc. This results in additional operating system overhead in allocation and de-allocation of memory blocks to find areas where the saved state can reside, and can have a profound effect on performance. In Section 6 we present performance curves showing a non-linear behavior associated with the memory management routines with substantial impacts on run time.

Addition of the RBC module completely off loads this function from the node. All accesses to state memory occur in the same memory range and aggregate memory usage on the node is substantially reduced, thus relieving the fragmentation effect. State memory is contiguous in the RBC hardware for ease of bookkeeping but this is not a requirement and is transparent to the processor. Additionally, since the maintenance of previous states is now handled by hardware an additional bookkeeping task is removed from the operating system.

(3) State Size

State blocks can consume large areas of memory for certain kinds of discrete event simulations. Estimates at NASA JPL are for near term requirements of 16K bytes of state per process and some applications could require as much as 1 Mbyte of state (e.g., environment modeling). If the local processor memory is fixed in size or if the addition of memory imposes a performance limitation, saving large states can become prohibitive.

As an example, assume an MC68020 processor configured with 4 Mbytes of state memory per node (such as the BBN Butterfly multiprocessor) running an application with 32 Kbytes of state. For 63 previous states saved, and 1 process per node, 2.09 MBytes of memory are required. Thus, approximately 50 % of available node memory is consumed with the saving of current and previous states. As illustrated by this example, even with modest state sizes local memory can become "state bound" as large numbers of previous states or large numbers of processes are generated.

Addition of the RBC module provides up to 64 Mbytes of installed memory per node when fully configured, corresponding to up to 1 Mbyte of state memory and 63 previous states retained. The node provides a single state memory "window" corresponding to the state size, thus freeing up the remainder of its local memory for other uses. Additionally, the the RBC allows for state size growth in increments, up to the 1 Mbyte maximum, depending on the application requirements.

2.0 PHASE II PROJECT OBJECTIVES

The objectives of the Phase II program were to:

(1) *Build and test production Rollback Modules based on the Phase I prototype.*

During the Phase II effort four (4) RBC modules were designed, built and tested. Three of the modules were populated with 4 Mbytes of state memory and one module was populated with 16 Mbytes of memory. All four modules were packaged as 9U X 400 mm VME wire wrap boards, suitable for installation in a 9U VME chassis such as used in the Sun 3/260 series workstations.

(2) *Develop and demonstrate an interface to multiple Rollback modules which is easily adapted to multiple commercial multiprocessor computers.*

As part of the Phase II effort, a vendor survey was conducted to identify a standard interface applicable to the greatest number of commercial multiprocessors. The specific results of the vendor survey are presented in section 5, Research Conducted, of this report. The general conclusion reached was that the VME interface standard was the most widely accepted standard suitable for a memory device such as the RBC.

The Phase II effort demonstrated two interface configurations for the RBC module, a direct interface to the VME backplane of the Sun 3/260 workstation and an indirect interface with a Sun SPARC IPX, S-Bus based, workstation through a commercially available VME translator module.

(3) *Build and demonstrate the performance of multiple rollback modules integrated with a common interface to a commercial multiprocessor.*

The Phase II effort developed a number of test and debug routines for the Sun workstations which provided specific timing and performance data on the RBC module. Performance testing was performed for both the Sun 3/260 and the SPARC IPX implementations.

(4) *Demonstrate system "proof of concept" of the Rollback modules in a system by integrating the Rollback modules in a commercial multiprocessor with Time Warp software running real applications.*

The standard Time Warp application Pucks, and two applications, Bank and Spot, which were written for this effort were used for performance testing of Time Warp with and without the RBC module. A discussion of these applications is provided in section 5, Phase II Research Consucted. Performance results are provided in section 6, Results Obtained.

3.0 ROLLBACK CHIP (RBC) MODULE FUNCTIONAL OVERVIEW

In a system configured with RBC modules each processor communicates with its RBC via the local node memory interface. The RBC module monitors all accesses to and from local memory by the processor node, and any access which is destined to be a state access (either read or write) is routed to the RBC hardware controller, which provides the needed data or performs the required action. The mechanics of state saving are removed from the processor and allocated to the RBC hardware controller. The hardware implementation of state saving does not involve data copying as new state blocks are allocated since this would result in only marginal time savings over the existing methodology used in Time Warp. Rather, the RBC hardware keeps track of the location of the most recent data for all state addresses in the current or previous mark frames. A write to a state variable results in a write to the Current Mark Frame (CMF) which is now located in the state memory on the RBC module. A read from state results in the RBC hardware locating the frame in memory which contains the most recent version of the address being accessed, and returning it to the processor.

The RBC implementation provides for 65 total frames of state memory. The "newest" or current frame (referred to as the Current Mark Frame) represents the frame currently being utilized by the node processor. The "oldest" frame (referred to as the Oldest Mark Frame) represents the frame which is the furthest in simulation time away from the CMF and is also ahead of the GVT (Global Virtual Time). The maximum distance, in frames, between the CMF and OMF is 62 frames (CMF + OMF + 62 frames = 64 frames total being tracked). One additional frame, referred to as the Archive frame, holds for each state address, the last valid piece of data written, for all frames prior to GVT. If during a state read no valid data is located in any of the 64 tracked frames then the "last" valid piece of data for that location is available in the Archive frame.

The tracking of the locations in state memory containing valid data is accomplished through the use of a large memory array referred to as the Written Bit (WB) memory. The Written Bit memory array can be conceptualized as 1 Meg addresses X 64 Bits in size with each of the 1 Meg array "addresses" corresponding to a state byte address and each of the 64 "bits", or columns, of the array corresponding to one of 64 frames which the array tracks. A logic "1" in a location at a particular address and column indicates that valid data is stored in that location in state memory.

The RBC module resides in two (2) separate blocks of node processor memory (i.e. within the node processors address space). The two blocks are referred to as the RBC State Memory area and RBC Command area. The RBC State Memory area consists of the 1 Mbyte of memory which is utilized for state memory accesses. This block is represented as single continuous block aligned on 1 Mbyte boundaries and can be thought of as a 1 Mbyte window into the memory space of the RBC. Address translation of state fragments from the node processor into RBC state memory is performed at allocation time and not on the fly by the RBC hardware. The RBC Command area consists of approximately eight (8) Kbytes of memory and is used to communicate commands and status information between the RBC module and node processor, and to load the RBC configuration tables.

The RBC's physical configuration is that of standard VME module which is designed to interface one-to-one with individual processors in a multiprocessor system. Based on the results of the vendor survey performed during this effort, the VME specification is the most widely accepted processor interface for add-on memory devices in current generation multiprocessors. The intent of the "standard interface" and "one-to-one configuration" approach is to allow the user the flexibility to configure multiple RBC's to meet specific simulation requirements while maintaining the simplicity of a standard interface.

4.0 RBC FUNCTIONAL REQUIREMENTS

The functional requirements established for the RBC module during the Phase II effort are delineated in the RBC specification, IPT Document Number 22-210018. To illustrate the major capabilities of the RBC module excerpts from the specification are provided in sections 4.1-4.11 below.

4.1 Installed Memory -vs- State Memory per Processor or Node

Installed memory represents the total amount of memory on the RBC module exclusive of the archive frame. Because the RBC must track previous versions of each object's state, the installed memory does not represent the size of a particular node's state. The RBC always retains 64 frames of state memory representing up to 63 previous versions of each state variable, all of identical size. This requires that the RBC's installed memory be 64 times the size of the total state memory for a given node. Thus the maximum amount of state memory for a particular RBC is always equal to it's installed memory divided by 64.

4.2 Maximum State Memory per Process or Node

The RBC hardware accommodates up to 1 Mbyte of total current mark frame state per processor or node. The current mark frame state can be either one process with 1 Mbyte of state or N processes with 1 Mbyte/N amount of state. The 1 Mbyte of state, configured either as a single process or multiple processes, represents the memory area which must be managed by the RBC hardware and which previous versions of must be kept.

4.3 Maximum Amount of State Allowed per Process

As indicated in paragraph 4.2 the 1 Mbyte of state may be configured as one process with one Mbyte of state or as N processes with 1 Mbyte /N state per process on a node by node basis. Thus the maximum amount of state memory per process is 1 Mbyte and each node may be configured independently.

4.4 Maximum Number of Processes and Memory Allocation

The allocation of memory within the RBC is defined in terms of memory "segments". A segment is defined as a block or portion of RBC state memory which has been allocated to a specific process. Segment definition is performed via the loading of segment definition tables and multiple segments within the same RBC module need not be the same size. The capability for up to 256 segments is provided and segments may be defined at any time.

In the RBC module, the defined memory segments are not explicitly linked to the processes they support. The RBC tracks the segments rather than the processes. A single segment can consist of only one process, however processes may consist of multiple segments and the segments need not be contiguous. All references to the RBC are made in terms of the defined segments.

As an example refer to Figure 0.5, Example RBC Memory Segmentation:

a. This RBC module (and processor node) contains one (1) Mbyte of state memory and five processes.

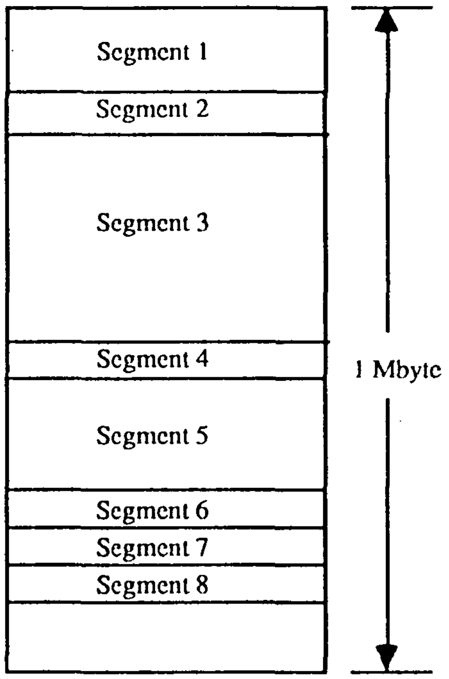

Figure 0.5, Example RBC Memory Segmentation

b. The five processes were initially allocated in segments 1-5 as shown in Figure 0.5, with segment 1 corresponding to process 1, segment 2 to process 2 etc..

c. Segments 6, 7, and 8 were added during run time and all are associated with process number 2.

d. The blank area at the end of the 1 Mbyte range represents uncommitted memory.

All references to the RBC are made in terms of the defined segments. During RBC memory references (i.e. state reads and writes) explicit identification of the segment being accessed is not required since the segment number (when required) can be determined from the address and the segment definition tables. However, for RBC commands the segment number cannot be inferred, and must be explicitly referenced. Thus for the RBC commands (Mark, Rollback, Advance etc.) the command must indicate which segment is to be operated on. As an example, from figure 1, if process number two (2) is to be rolled back four separate commands are required, one for each segment. The rbc_driver and modified TW code maintains the necessary linkage between the segment definitions and their processes. However the 256 max segments is a hard limit which may not be exceeded regardless of the number of processes.

4.5 Minimum State Memory Size per Segment

The minimum state size for any memory segment is 1 Kbyte. The LSB (increment value) for segment definition is also 1 Kbyte. To fully utilize the 1 Mbyte of available state memory at least one segment must be defined as greater than the 1 Kbyte increment value (since only 256 total segments are allowed).

4.6 Dynamic Process Creation

The RBC module provides the capability to dynamically create a new process on a given node during run time. Creation of processes (segments) is limited to those which will fit within available memory segments. The RBC itself does not provide the capability to directly shift or move segments to reclaim memory which has become fragmented as a result of process destruction.

4.7 Memory Fragmentation

Creation of processes (segments) is limited to those which will fit within available memory segments. The RBC itself does not provide the capability to directly shift or move segments to reclaim memory which has become fragmented as a result of process destruction.

The capability to consolidate memory fragments is implicitly provided in that the capability to destroy and re-initialize a process is provided. Thus by successive destruction and creation processes may be moved to allow more efficient memory utilization.

4.8 Dynamic Process Destruction

As previously indicated, the RBC is concerned with memory segments and not processes. Destruction of a process results when the segment definition or definitions utilized for that process is written over for use by another process. Thus a process can be considered to be "destroyed" only when its definition data has been overwritten. However, in practice simply not issuing RBC commands to the segments associated with a process effectively destroys it.

4.9 Dynamic State Size Growth/Shrinkage

Dynamic growth or shrinkage of state memory is supported in the RBC module through the use of the memory segments as outlined in sections 4.4 - 4.6. When utilizing dynamic memory with the RBC the following cautions apply:

a. The "expanded" state vector is treated as a new process or segment by the RBC hardware. New CMF and OMF counters are used, a new archive frame is established, and a new portion of the WB memory is utilized. The expanded process must be initialized just as a new process would be to avoid returning spurious or old data.

b. When issuing RBC commands to the expanded process all segments which make up the process must be explicitly commanded so that they stay in synchronization.

4.10 Number of Previous States Retained

64 versions or "Frames" of the state are tracked and stored at all times in the RBC hardware. The 64 frames consist of 1 frame of CMF state and 63 frames of state history.

4.11 State Memory Expansion Increments

State memory (exclusive of the archive frame) is expandable in increments of power of two from 1 Mbytes to the fully configured 64 Mbytes. Since state memory accommodates both the current mark frame and the 63 state history frames the available CMF state memory per processor is actually state memory/64. Table 1 summarizes the available configurations for state memory and the corresponding CMF state sizes. Also, the WB memory is not expandable and thus will always be configured as a 1 Meg X 64 array.

Table 1. State Memory vs CMF state size

State Memory Installed	CMF frame size
1 Mbyte	16 Kbytes
2 Mbytes	32 Kbytes
4 Mbytes	64 Kbytes
8 Mbytes	128 Kbytes
16 Mbytes	256 Kbytes
32 Mbytes	512 Kbytes
64 Mbytes	1.024 Mbytes

5.0 PHASE II RESEARCH CONDUCTED

This section describes the work carried out during the Phase II effort to create the RBC modules, perform modifications to the Time Warp code and conduct performance testing on the Sun workstations.

5.1 Systems Configuration

Referring to Figure 1, Sun 3 Systems Configuration, and Figure 2, Sun 4 Systems Configuration, a depiction of the two configurations used for test and analysis of the RBC modules is provided.

For maximum performance as an add-on memory board, each RBC module must reside on its individual node's memory bus and ideally would outperform the nodes memory both in State saving and in state memory accesses. Since the RBC is essentially competing with the node's memory interface during performance testing, any induced delays in the path from the node to the RBC will directly impact results.

In Figures 1 and 2, the PC-AT Debug Terminal is used as a test and debug device which provides a path to the RBC via its RISC processor (DSP56001). This feature allows for interrogation of all internal DSP registers, interrogation of memory, load and modification of code, program execution in single step or run mode, and interrogation of RBC board parameters. The link from the PC to the RBC consists of a direct connection to the DSP via a memory mapped, 8 bit parallel port, which is implemented with a flat ribbon cable connector mounted on the front RBC module. The debug port, with modified cabling, can support up to 8 RBC's simultaneously. Use of the debug port requires firmware resident in the DSP as well as software and hardware resident on the PC, and is therefore not accessible by the user.

Also in both figures 1 and 2, the RS-232 link provides the ability to down load code and files from the Sun network to the PC for further down loading to the RBC module (DSP code development is performed on the Sun 3).

5.1.1 Sun 3 Configuration

As shown in Figure 1, the configuration for single and multi-node Sun 3 testing consists of a network of Sun 3/260's which are connected via an ethernet link. Each RBC module resides on the Sun's VME backplane along with other devices (memory board, display board etc.). Message communication for Time Warp is provided by the ethernet link. In this configuration the RBC outperforms the node in state saving but is slightly slower than the node for a standard memory reference. Thus the overhead cost of using the RBC appears partly as a slower memory reference time. The major reason for the slower response are:

a) On-Board Cache - The Sun 3/2xx series of workstations are provided with a 25 mhz, 64 Kbyte on-board cache. When the Sun 3 uses the cache no time penalty for acquiring and accessing the VME bus is incurred in the transaction. Since the RBC resides only on the bus, any access to it must always incur the VME access penalty.

b) RBC Response Time - The access time of the prototype RBCs built for this effort are somewhat slower than a standard memory reference due to the complexity of the module and its initial implementation in wire wrap rather than printed circuit board.

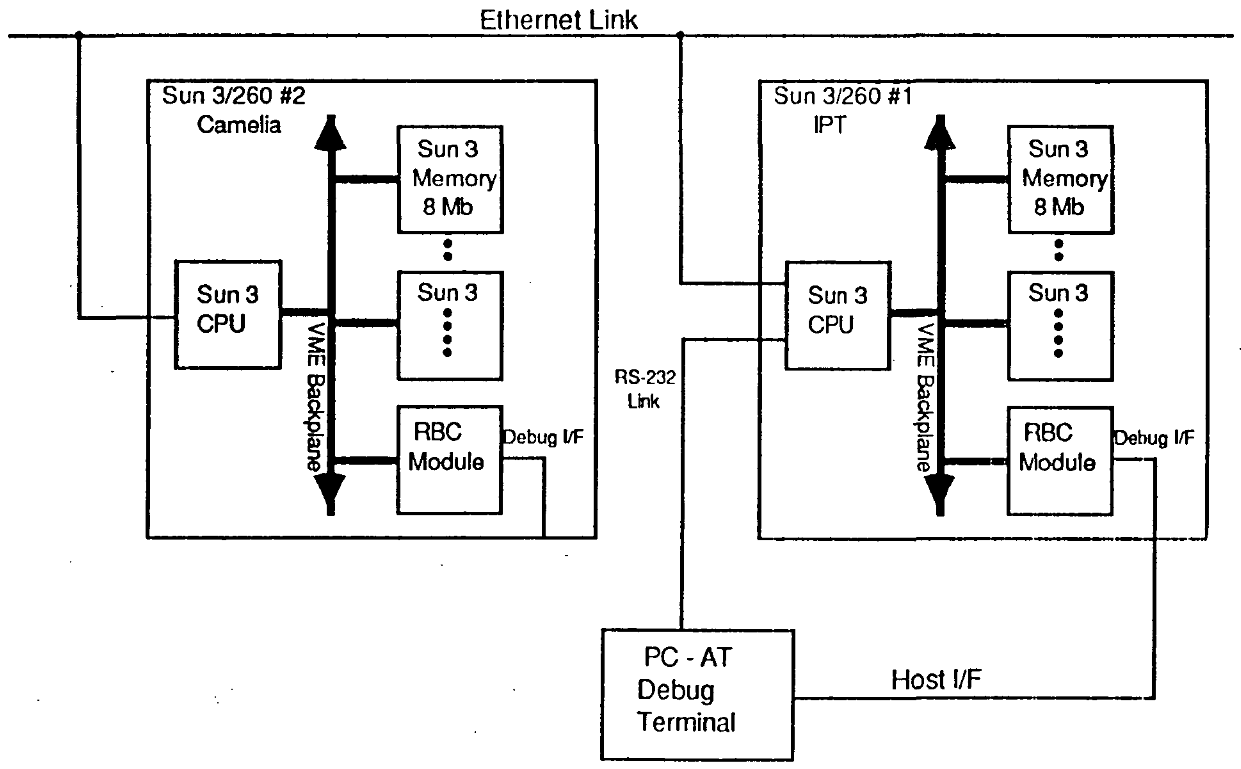

Figure 1, Sun 3 Systems Configuration

Subsequent versions of the board will provide improved access times, as discussed in Section 6, Performance Results.

c) Sun 3 Device Interleaving - The Sun 3 VME controller provides two separate protocol implementations for VME devices. Devices which respond within a preset timing window are considered "fast" while those outside the window are "slow". Because the RBC response time is outside the "fast" window, additional timing delays are introduced by the Sun 3 in implementing the "slow" protocol.

While faster versions of the RBC will reduce the difference between the memory reference times it is clear that an add-on memory device cannot perform at speed with a CPU's main memory, especially when caching is involved. If the RBC were incorporated in the design of a CPU's memory controller we would not expect this difference to occur. Additionally, the results in section 6 show that of the two effects (state saving time and memory reference time), state saving is generally the more dominant.

5.1.2 Sun 4 Configuration

As shown in Figure 2, the configuration for single node testing of the Sun 4 consists of a single Sun SPARC IPX which is connected via an ethernet link to the Sun 3 network. The ethernet link is provided for file and disk support in testing rather than for multi-node Sun 4 performance testing. This is because of incompatibilities between the MC68020 chip used in the Sun 3s and the SPARC chip used in the IPX, resulting in an inability to run Time Warp in a mixed Sun 3/Sun 4 network. In this configuration the RBC module resides in a stand alone VME chassis with an S-Bus to VME translator module[2] providing the connect path to the SPARC's Sbus.

Accesses to the RBC prototype from the the Sun 4 are slower than for standard Sun 4 memory because:

a) Translation Latency - Use of the S-Bus to VME Bus translation module results in the imposition of approximately 1 microsecond (1μs) of translation latency on each state read or write to the RBC module.

b) On-Board Cache - Like the Sun 3 series of workstations, the Sun 4 IPX is provided with 64 Kbyte of on-board cache but at 40 mhz instead of 25, as on the Sun 3. The Sun 4 is rated at 3-5 times the speed of the Sun 3, depending on the application.

c) RBC Response Time - The RBC response time to access attempts by the Sun 4 is identical to its Sun 3 response time.

The intent of the Sun 4 testing is primarily to verify that the RBC can be interfaced to a standard VME device with little or no modification, and to identify any functional errors in the module's performance. The large latency inherent in the use of the general purpose translation module precludes its likely use in a deliverable configuration. As discussed in Section 6, Results Obtained, some anomalies were found in the RBC's behavior when subjected to the higher throughput rates of the Sun 4 as compared to that of the Sun 3. Also, as discussed for the Sun 3 implementation, it is clear that an add-on memory device cannot perform at speed with a CPU's main memory, especially when caching is involved. As with the Sun 3 the results in section 6

[2] PT-SBS915 Sbus-to-VMEbus Adapter, Performance Technologies Incorporated, 315 Science Parkway, Rochester New York, 14620

show that of the two effects (state saving time and memory reference time), state saving is generally the more dominant.

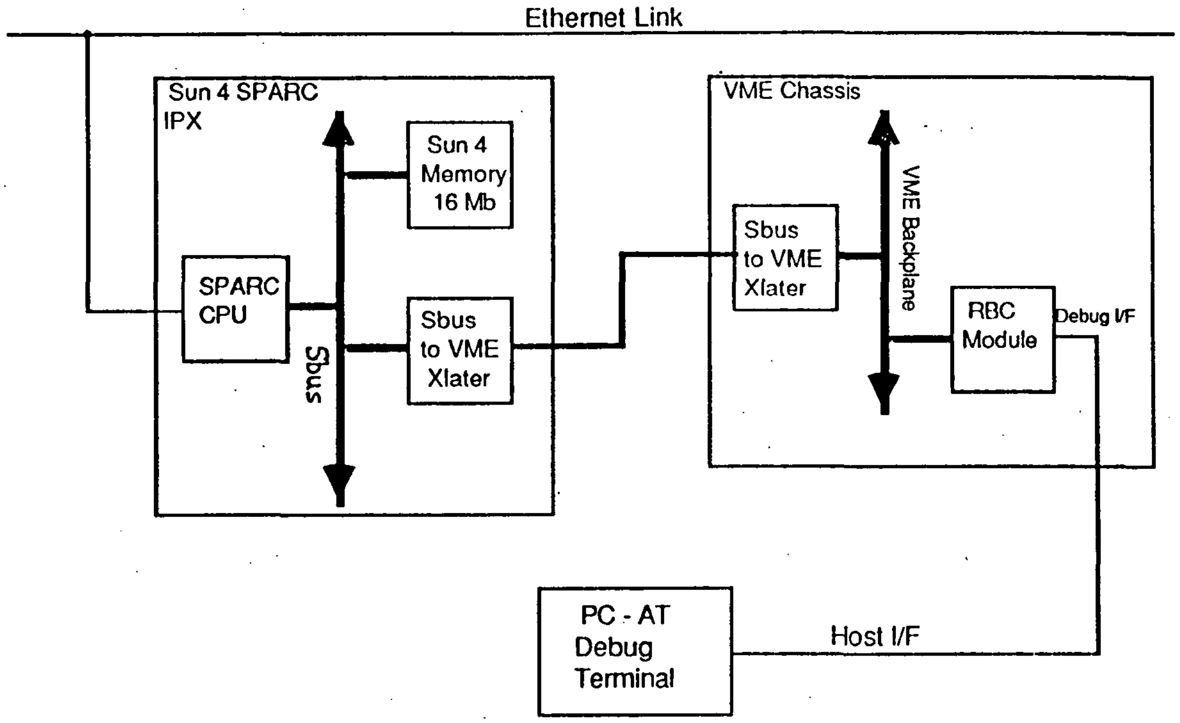

Figure 2, Sun 4 Systems Configuration

5.2 Vendor Survey

During the initial months of the Phase II effort, a survey of major multiprocessor manufacturers was conducted to determine the hardware interface which is most widely supported for add-on boards and modules, at the node level (for interface purposes the RBC module can be conceptualized as a standard memory board which is added to each node of the machine). The intent of the survey was to ensure that the design of the RBC module allowed it to be interfaced with as many machines as is practicable.

Twelve manufacturers were surveyed and the data collected is summarized in Table 2, Multiprocessor Vendor Survey Results. From the data presented in Table 2 and the technical analysis of the various interfaces, the following conclusions can be drawn:

a. The majority of the multiprocessors do not support a recognized standard bus interface at the individual node level. However, a number of machines do provide the ability to interface with the node over a custom or semi-custom interface.

b. Based on vendor discussions, the lack of inclusion of a standard interface at the node level appears in most cases to be an oversight in first generation machines. Vendors have indicated that machines which are in the design stage or those which have just recently been released will address the node level interface.

c. Where a recognized standard interface is currently implemented or planned in the future, the VME bus standard is the accepted interface.

d. In general, the non-transputer based multiprocessor implementations offer the highest potential for easy RBC implementation. This is due to the speed limitations inherent in the serial transmission capability, which is built directly into the transputer processors (chip) and is used for both interprocessor communication and node I/O. While the capability is excellent for interprocessor communication, it is generally unacceptable for a memory interface.

Table 2, Multiprocessor Vendor Survey Results

Vendor	Processor	Interface to RBC Potential		
		Direct VME I/F	VME I/F via Adapter	I/F * Difficult
BBN Advance Computers	68020 (GP1000)		√	
	88100 (TC2000)	√		
Cogent Research	IMS T800 Transputer		√	
Concurrent Computer	Proprietary			√
Encore Computer	NS32332			√
Intel Scientific Computers	80386 (IPSC/2)		√	
	i860 (IPSC/860)			√
Kendall Square Research	(no response to query)			
Meiko Scientific	IMS T800 Transputer			√
Myrias Computer	68020			√
NCUBE	Proprietary			√
Paracom	IMS T800	√		
	IMS T414	√		
Sun Microsystems	68020 (Sun 3)	√		
	SPARC (Sun 4)	√	√	
Topologix	(no response to query)			

* Interfacing the RBC with the current generation of these machines would either be difficult to accomplish because of lack of a node interface capability, or because the node interface would impose performance limitations on the RBC.

Based on the conclusions noted above and Table 2, the multiprocessor interface designed for the RBC module was the VME standard, as originally proposed. For processors which do not support the VME standard an adapter will be designed in the future to translate the node processor's interface to the VME standard.

5.3 Time Warp Modifications and Added Modules

To use the RBC hardware with Time Warp several software changes were required to be made and a number of modules added. Since the RBC is a device to the Unix operating system, an RBC device driver was needed. In Versions 4.1.x of the SunOS Unix, the Unix kernel needed to be modified to include the RBC device driver. To accommodate both the additions required to directly support Time Warp and the requirement for the ability to interrogate the RBC directly during test, the driver functions were divided between the two modules rbc_driver and rbc_debug.

The Time Warp code for TWOS version 2.5.1 was modified to use the RBC. The goal was to make the use of the RBC transparent to the Time Warp user in all ways other than speed of execution. The modified code uses RBC memory for all state variables. The modified TWOS supports dynamic creation of objects, and dynamic allocation of state variables but not dynamic load management of objects. (This last feature isn't available even on the non-RBC Sun version of TWOS 2.5.1.). To use the modified TWOS, the applications programer merely links his/her program with a Time Warp library. The library shares most of its files with TWOS 2.5.1, but is compiled in a separate directory.

The modified code has been tested on Sun 3's and Sun 4's and includes some bug fixes found in later versions of TWOS. One can actually run this code without having a rollback chip board installed. The code will automatically run the rbc_emulator code if there is no RBC board.

To aid in test and debug of the RBC modules, a number of user routines were written including a simulated Time Warp event sequence, and a stand alone debug package. Additionally, two Time Warp applications were written for performance studies and a standard Time Warp application was modified (made smaller) for use with the RBC. These routines are discussed in sections 5.3.1 -5.3.5.

5.3.1 Xrbc

Xrbc is a test and debug program which was used to test the ability of the RBC hardware module to perform all its required functions. Xrbc allows for all commands available on the RBC memory map to be executed, one at a time, while varying the segment ID, frame number, and starting and ending memory location. This code was the primary vehicle for RBC module test prior to integration.

5.3.2 Rbc_test

Rbc_test is a test and debug program which is used to test the ability of the emulator version of Time Warp and also the RBC hardware to perform multiple marks rollbacks and advances. Rbc_test can be conceptualized as an emulation of a standard Time Warp run with the exception that the sequencing of events is fixed and the operations (Mark, Rollback, Advance) occur in groups rather than interleaved. This routine also tests the ability of the hardware and software to detect and recover from Mark Frame Counter (MFC) overflow and underflow.

Rbc_test is provided as a deliverable routine with the RBC modules and is run as a confidence test after installation.

5.3.3 Application Bank

Bank is small simulation which was written for this effort and was used to obtain performance numbers for the rollback chip board (RBC). Bank is loosely modeled on a computer subsystem with multiple memory banks that have multiple memory accesses. The analogy is tenuous, but it does explain the name. There is only one object type in bank. This object type is also called `bank'. In a bank initialization section (its init code) sends an intialization message to be sent to itself at time zero.This initialization message causes the bank object to send MSG_NUM messages to a random bank object with a random arrival time (exponential with mean 10.0). Each message is either a read or write message. Write messages contain a value and a location to write the value in the objects memory bank. Read messages ask the bank object to read a given location. Either message generates another message for a random bank with an arrival time of now plus a random time (exponential with mean 10.0), messages which would arrive after CUTOFF (1000.0) are not sent.

Bank contains many compile time constants which provides a convenient method for performance testing of the RBC under various loads. These are:

NUM_BANKS - Number of banks is the number of bank objects in the simulation. Since there are `really' only 16 different random streams in the random code in rand.c, more that 16 bank objects will no longer have the same degree of randomness.

BANK_SIZE -Bank size is the number of integers in the bank objects `memory bank'. The actual size in bytes of bank object state is 4 times BANK_SIZE plus 17.

NUM_MSGS - Number of messages relates to the amount of parallalism available in the simulation. At each simulation time, there are NUM_MSGS times NUM_BANKS messages in the system.

Another way to look at bank is an n-server system (n=number of bank objects) with nm clients (m= number of messages). The strange behavior is that each server can service a client in 0 service time (hence no waiting), but its arrival at its next random server (topologically, the servers are fully connected) is a random time in the future.

5.3.4 Application Spot

Spot is an application which was also written for this effort and is related to bank. Spot uses the TWOS feature of dynamically allocating state memory. Spot was used to test the rollback chips implementation of this feature. Each spot object is a modified bank object. Spot_0 generates 10 dynamically allocated pieces of state. At random times, a spot object will `pass' a dynamically allocated chunk (if it has any) to the spot with the next higher number. Thus, if the simulation lasts long enough all of the pieces will migrate to the highest numbered spot object, spot_7.

5.3.5 Application Pucks

Pucks is an application which simulates the interaction of colliding pucks and has the capability to run on a single node processor. The version of Pucks in use for the RBC consisted of a standard configuration which was modified to allow it to run on the small number of nodes in use for this effort. Standard Pucks uses 128 sectors, 48 cushions and 128 pucks. Each sector requires 7 Kbytes of state memory, each cushion 5 Kbytes and each puck 5 Kbytes for a total of 1776 Kbytes. In a typical Time Warp run with large numbers of nodes, the memory utilization per node for Pucks is small since the allocation is distributed over all the nodes. However, in this effort Pucks memory allocation must be distributed across the four nodes available, theoretically resulting in over 400 Kbytes per board. Therefore Pucks was modified to reduce the number of Sectors, Cushions and Pucks to allow testing on as little as a single node. The single node version uses 1 sector, 4 cushions and 1 puck resulting in a total state memory requirement of 32 Kbytes (which consumes one half the memory available on a single node with 4 mb of memory). For 2-node Sun3 and single node Sun 4 performance testing, additional versions of Pucks were built to provide as close a match as possible between the application and the RBC's installed memory.

5.4 Functional Description and RBC Block Diagram

This section describes each of the major functional blocks of the RBC module and each of the six basic RBC functions as they relate to Time Warp operations. Refer to Figure 3, RBC Block Diagram, a simplified block diagram for the following discussions. A complete RBC block diagram is provided, as deliverable documentation, as IPT drawing Number 20-210019.

5.4.1 RBC Block Diagram Description

Segment ID (SID) Latch and Buffer, and Segment Encoder (Seg Enc) Table

All RBC operations are based on the ID of the segment (or process) being acted on. As shown in Figure 3, the output of the SID latch and Seg_Enc Table both directly drive the SID bus (labeled Segment_no.) which in turn drives a number of the memory elements on the RBC. These memory elements contain the current run time configuration for all processes which are active on the node. There are two methods for communicating the segment number to the RBC:

(1) Node Commands - For commands from the node (Mark, Rollback, Advance), where the Segment ID must be explicitly referenced by the node, the SID latch is used to set the correct address on the RBC memory elements and to communicate the SID to the DSP processor (connection not shown in figure 3).

(2) State Read/Write - For state reads and writes, the SID must be inferred from the memory address requested by the node to correctly set the segment address on the RBC memory elements. This is accomplished through the Segment Encoder table, and is performed on the fly for each read or write.

CMF and OMF Counter Memories

In order to properly maintain the written bit memory, and thus the state memory, the RBC must maintain pointers to the CMF (Current Mark Frame) which represents the frame that the node is currently working in, and the OMF (Oldest Mark Frame) which represents the last frame tracked prior to GVT, for each segment in the simulation. Maintenance of the CMF and OMF pointers is crucial to correct RBC operation because they represent the "span" of the simulation. Because of background processing on the RBC it is possible for written bits outside the current span of the simulation to be set. Additionally, because 64 frames are maintained and typical scenarios have in excess of 30,000 events, the memories are circular in nature (i.e. modulo) and will "roll over" many times during a run. Because of this, the simulation span occurs for values of CMF both greater and less than the OMF.

Written Bit Array

The written bit array is a large memory area which is used to track the location of valid pieces of data in state memory. The array can be conceptualized as 1 Meg addresses X 64 Bits in size with each of the 1 Meg array "addresses" corresponding to a state byte address and each of the 64 "bits", or columns, of the array corresponding to one of 64 frames which the array tracks. A logic "1" in a location at a particular address and column indicates that valid data is stored in that location in state memory. Although visualized as 1 meg X 64, the WB memory is actually much smaller in size to conserve board space and reduce cost. The physical implementation of the WB memory is 128K addresses X 64 bits, with each of the 128K addresses corresponding to a Double Long Word (64 data bits wide), and is comprised of eight (8) 128 K X 8 high speed static RAMS.

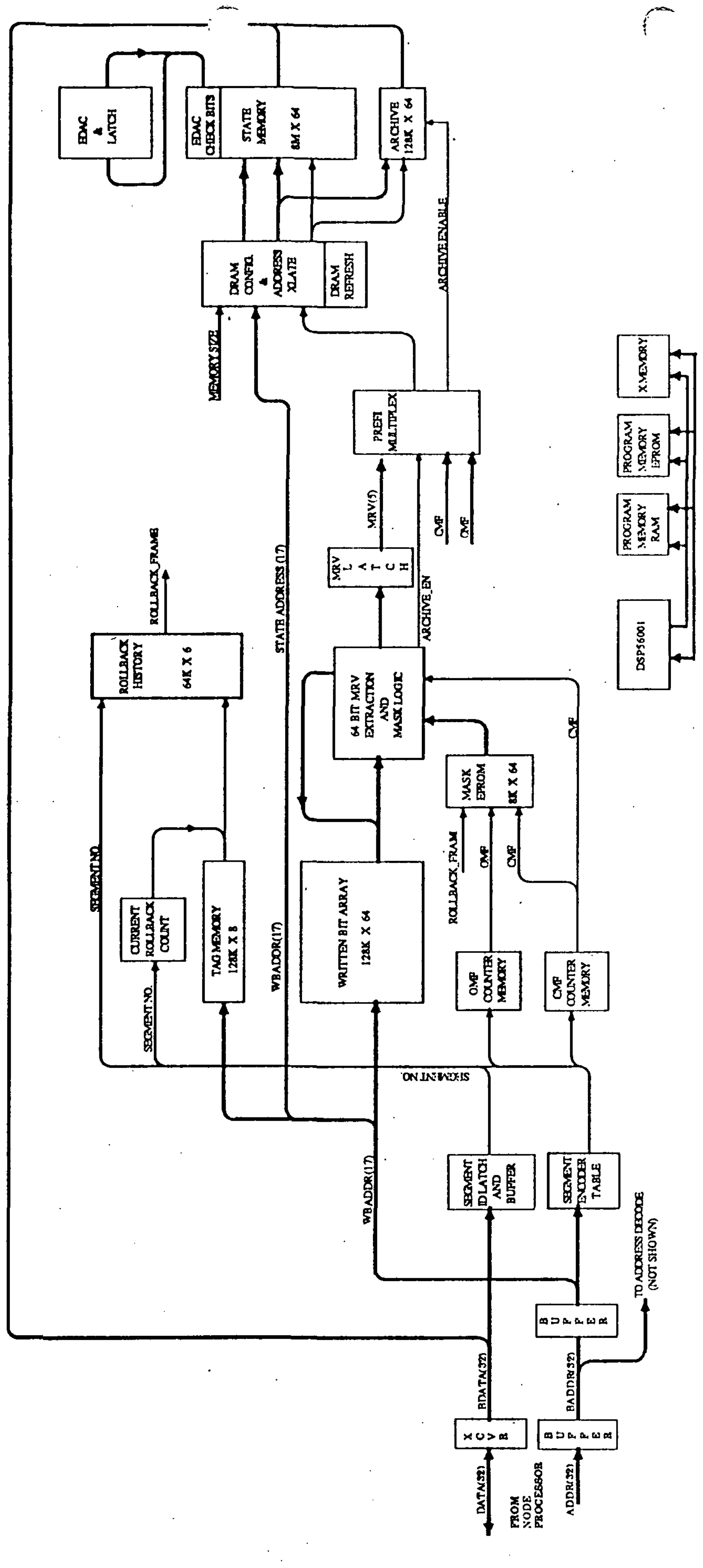

Figure 3, RBC Block Diagram

As show in figure 3, the address input to the array consists of 17 bits of the node address bus, which is also fed forward to the state memory array. The output of the written bit array is fed into the MRV extraction and mask logic block which acts on the array and is tasked with maintaining it.

MRV Extraction and Mask Logic

The MRV extraction and mask logic essentially performs two functions; first it maintains the written bit array by selectively 'ORing' or 'ANDing' the written bits with a specified mask pattern, and second, it performs MRV extractions to locate the last valid piece of data for a particular address. Patterns for the varied mask operations are stored in the mask EPROM and inputs to the EPROM, consisting of the OMF, CMF and rollback frame destination, are used to vary the mask output. Additionally, the inputs to the mask EPROM may be directly set by the DSP during background tasks.

Prefix Multiplexor

The prefix multiplexor is controlled by the DSP 56001 and RBC control logic and is used to select the frame which is to be accessed for a particular read or write. For State writes the CMF is used. For Sate Reads the MRV is used. During advances the prefix input for the MRV is loaded with the OMF frame number closest to the New_OMF, containing valid data.

Note that during the advance, data is read from the OMF and latched into the latch labelled LATC. The DSP then initiates a Write to Archive. The effect is that old versions of valid state data are written into the archive frame as the GVT progresses.

Tag Memory, Current Rollback Count, and Rollback History

The tag memory, current rollback count memory and rollback history memory, form a subsystem within the RBC module. Conceptually, when a rollback is commanded by the node processor for a particular segment, the RBC should cycle through the written bit memory clearing all written bits from the Old_CMF (where we were) to the New_CMF (where we have rolled back to). However the time to clear the written bits can be long for large states since each address must be accessed to perform the clearing. For a scenario with a large number of rollbacks the potential for significant performance degradation exists. To remove the potential problem, at the cost of a little complexity, the 'lazy rollback' approach was implemented.

In lazy rollbacks a rollback history is kept of all rollbacks for each segment. Each time a rollback is commanded, the destination frame for this rollback is added to the history and the rollback count is incremented. Additionally, previous destinations in the rollback history are compared to the current destination and if the current destination is "deeper" (further back in time) than the entry in the history, it is updated to the "new" destination. An entry in the history can be thought of as containing a record of the furthest rollback destination that has been commanded since the time that this rollback entry was commanded. The wrinkle here is that the entries in the history are indexed by the rollback count rather than by time. The tag memory is a shadow of the written bit memory and contains a rollback count entry for each written bit address. The tag memory entry signifies the "time", in rollback counts, at which this written bit memory location was last updated. The rollback history, indexed by the tag memory's output, will provide the destination frame, for each address in the written bit memory, of the deepest rollback that has ocurred since the last time this location was accessed. With this subsystem, rollback updates can now be performed on state read and writes, a single address at a time rather than for all addresses as before.

DRAM Configuration, State Memory, and EDAC

The DRAM configuration and address translation function is used to match the prefix and state address requests from the RBC with the installed memory configuration for the board. This module would not be required if only a single installed memory configuration were available (i.e. address and prefix lines would be hard coded). The address translation essentially slides the prefix across the upper address lines depending on the installed memory.

State memory consists of up to 64 Mbytes of Dynamic RAM configured as 8M X 64 bits and using of 256K X 4 bit chips. The design of the RBC can handle either 256K X 4's or 1 M X 4's depending on the required memory. However, because the prototype boards are wire wrap modules they only support the 256K X 4's. Archive memory is fixed at 128K X 64 bits which equates to the full 64 Mbyte implementation. Archive memory is not expandable and is implemented with 256K X 4's.

Dynamic RAMs are susceptible to two types of errors generally referred to as 'hard' and 'soft' errors. Hard errors are indicative of device failure in either single or multiple bits and when detected require replacement of the IC. Soft errors are radiation induced errors which are a problem for Dynamic RAMs because of the capacitive technique used to store the logic level of each cell in the memory array. Soft errors result in apparent random errors in data stored in the memory. Soft error rates increase with DRAM density per chip and are additive across the memory array. Because of the very large size of the state memory array and the chip densities utilized, a prediction of the Mean Time Between Failure for soft, single bit errors in the array was computed at between 105 and 325 days. While this is not anticipated to be a problem for the prototype modules it would be unacceptable for production units. Therefore an EDAC (Error Detection and Correction) circuit which provides detection and correction of single bit errors, and detection but not correction of multiple bit errors was implemented. Based on the addition of the EDAC circuitry the MTBF for soft, single bit errors was recomputed to be 109 years. However this does not account for hard failures. With the addition of the hard error probability the estimated MTBF is 30 years. Note that in the prototype modules the EDAC correction circuitry is disabled due to a design problem which was not able to be corrected prior to the completion of the Phase II effort.

5.4.2 RBC Functions

In general all RBC manipulations during run time are centered around the Written Bit memory and MRV (Most Recent Version) extraction hardware. There are six basic commands or functions (disregarding housekeeping and initialization commands) which are issued by the node processor during run time. These are discussed in the following sections.

5.4.2.1 Transparent Read/Write

A transparent read/write is defined as an access to local node memory which does not fall within the state or command memory ranges as defined for the RBC module. RBC state and command area memory definitions are selectable by dip switch settings on the RBC, which feed the address decode logic to provides the necessary comparison. The RBC hardware ignores an attempted access by the node processor to an area which is not either contained in a state memory block or the RBC command area.

5.4.2.2 State Read/Write

An attempt to access state memory causes the RBC hardware to respond with the requested action (i.e. read or write).

Conceptually, for a state read the RBC merely performs an MRV extraction to locate the frame number where the last valid data exists for this address (defined as the frame number closest to the CMF containing a logic 1), sets the prefix on the state memory, and performs the requested read. However, because of the addition of the rollback history subsystem the read's complexity increases. Thus, before the RBC can perform the MRV extraction and read, it must first perform a rollback update on the written bits, to insure that bits which have been rolled back since the last access of this location are cleared to prevent accessing "future" data. After the rollback update, the RBC must also update the tag memory for this state address to reflect the fact that the written bits have been correctly set.

Similarly, for a state write, the RBC conceptually must write the data provide by the node into memory in the Current Mark Frame (CMF), and store a logic '1' in the written bit memory location corresponding to that address in the CMF. However, because of the addition of the rollback history subsystem and the requirement to support byte, word and longword transfers the write's complexity increases. Thus for a write to state, the RBC must first perform a rollback update on the written bits, to insure that bits which have been rolled back since the last access of this location are cleared to prevent accessing "future" data. After the rollback update, the RBC must also update the tag memory for this state address to reflect the fact that the written bits have been correctly set. Additionally, since the data being written could potentially be byte wide data and the RBC is organized as double long words, care must be exercised to insure that data which is stored in the remaining byte fields of the double long word are also correctly updated. The approach to maintaining this data integrity is to copy forward the data stored in the remaining byte fields along with the new data, and then write the new double longword into the CMF frame. However, to perform a copy forward requires an MRV extraction be performed to locate the last valid whole double long word.

5.4.2.3 Mark Command

A mark command from the node processor indicates to the RBC hardware that the node wishes to save the current state and allocate a new state memory block for use in the next frame. Since the RBC does not use state memory copying the mark command is implemented by simply incrementing the CMF frame counter by 1.

5.4.2.4 Rollback Command

A rollback command is issued by the Node processor when the optimistic strategy employed by Time Warp results in the Node receiving a message or event which is supposed to occur at a time that has already passed. When issuing the Rollback the processor will also indicate how many frames are to be "rolled back" (i.e., removed from the state history). For the RBC, conceptually, a Rollback is quite simple. The frames which are in the affected range as specified by the rollback distance need simply be "popped" or decremented from the CMF counter. No change in the State memory itself is needed since valid data is tracked in the written bit memory. However the WB memory must be purged of bits in "rolled back" frames which are no longer valid. This is because upon the first Mark following a rollback the CMF counter will advance to a frame which was previously "Rolled Back". It is unacceptable to have set bits in this frame erroneously indicating valid data where none exists. As previously discussed, implementation of the rollback history mechanism has removed the requirement to perform the written bit update during the rollback command and has moved it to the state read/write. During the rollback command the rollback history update is performed.

5.4.2.5 Advance Command

An advance command is issued by the node processor when the simulation has advanced far enough to guaranty that states prior to some time are no longer needed (the data in the state memory may be needed but the state itself as a point in time is not). The node processor calculates the number of frames which can be advanced and writes this information to the RBC hardware.

Execution of the advance function is performed as a background task by the DSP and does not materially impact the simulation, unless the advance is large enough to cause the RBC to temporarily run out of frames while waiting for it to complete (see section 6, for further discussion). The Advance task is the lowest priority task and is pre-empted by all other tasks.

In an advance the RBC interrogates the WB array for valid bits between the "old" OMF and "new" OMF, based on the advance distance, for the process specified. Data represented by the most recent valid bit for each WB address is physically copied from the indicated frame to the archive frame. The OMF counter is advanced to a new OMF only after completion of the advance operation.

6.0 RESULTS OBTAINED

Results of the performance tests for Sun 3 single and multi-node testing and Sun 4 single node testing are provided graphically in the following sections.

6.1 Sun 3 Single Node Testing

Single node Sun 3 test results for Bank are provided in Figures 4 through 8.

6.1.1 Effect of Cutoff on Elapsed Time

Referring to Figure 4, Effect of Cutoff on Elapsed Time, the elapsed run time, in seconds, is plotted as a function of State size per object, for various cutoffs. This is a plot of Sun 3 elapsed times only, using a 3 megabyte heap. For this application (Bank) on one node and with 8 objects, the total state memory for 128 Kbytes/object (the last plotted point) is 1 Megabyte. The intent of the plot is to show the effect of the simulation's cutoff time on the elapsed time and to determine the appropriate cutoff (if one exists) for use in subsequent test runs. For each cutoff above 1000 two plots are provided. One is the elapsed time as measured, and the second is the calculated elapsed time based on the cutoff of 1000.

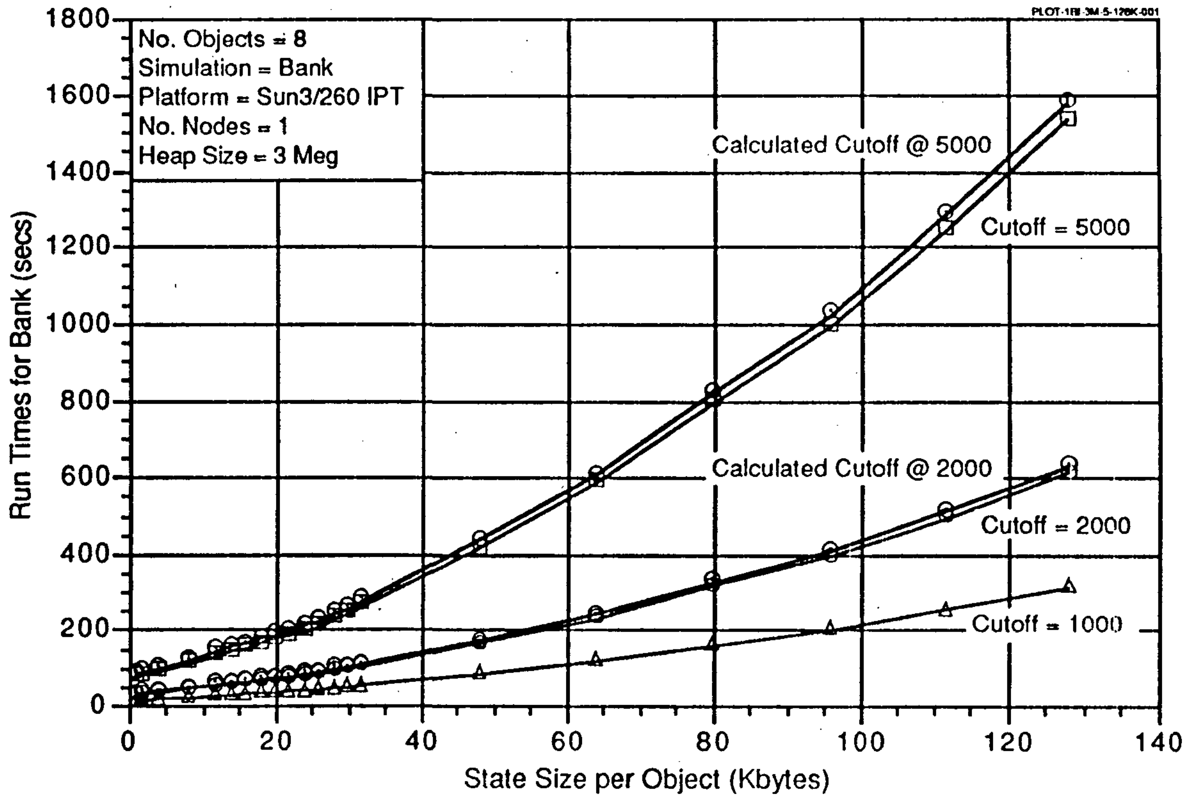

Figure 4, Effect of Cutoff on Run Time

From the information provided a number of observations can be drawn.

- All three curves appear non-linear in shape. For a simple simulation such as bank we would expect that as state size is increased and the number of events (cutoff) holds constant that we would see a linear increase in elapsed time. Clearly, an additional effect is present. As we shall see in later plots this effect is present to some degree in all the Time Warp without RBC runs.

- While non-linear all three curves track, as evidenced by the correlation between the calculated and measured traces. The slight increase in elapsed time between the calculated and measured runs appears to be a function of timing resolution in the lower cutoff values.

- Based on figure 4, the effect of cutoff on run time appears negligible and therefore a cutoff of 5000 will be used for subsequent Sun 3 single node testing.

6.1.2 Effect of State Size on Elapsed Time

In Figure 5, Effect of State Size on Elapsed Time Through 32 Kbytes/object, and Figure 6, Effect of State Size on Elapsed Time Through 128 Kbytes/object, the elapsed run time, in seconds, is plotted as a function of State size per object, for the Sun 3 with and without the RBC. Two variants of the Sun 3 runs are provided based on the heap size specified for Time Warp.

For the prototype modules built in the Phase II effort, two configurations were implemented. Three of the RBC modules contained 4 megabytes of installed memory and one contained 16 megabytes. For the 16 Mbyte variant the maximum state size per object, for Bank with 8 objects, is 32 Kbytes. Thus, Figure 5 represents the measured times for the maximum RBC configuration built under this effort. Figure 6 provides additional test results for the Sun 3 and an extrapolation of the RBC times up to the maximum of 1 megabyte of state.

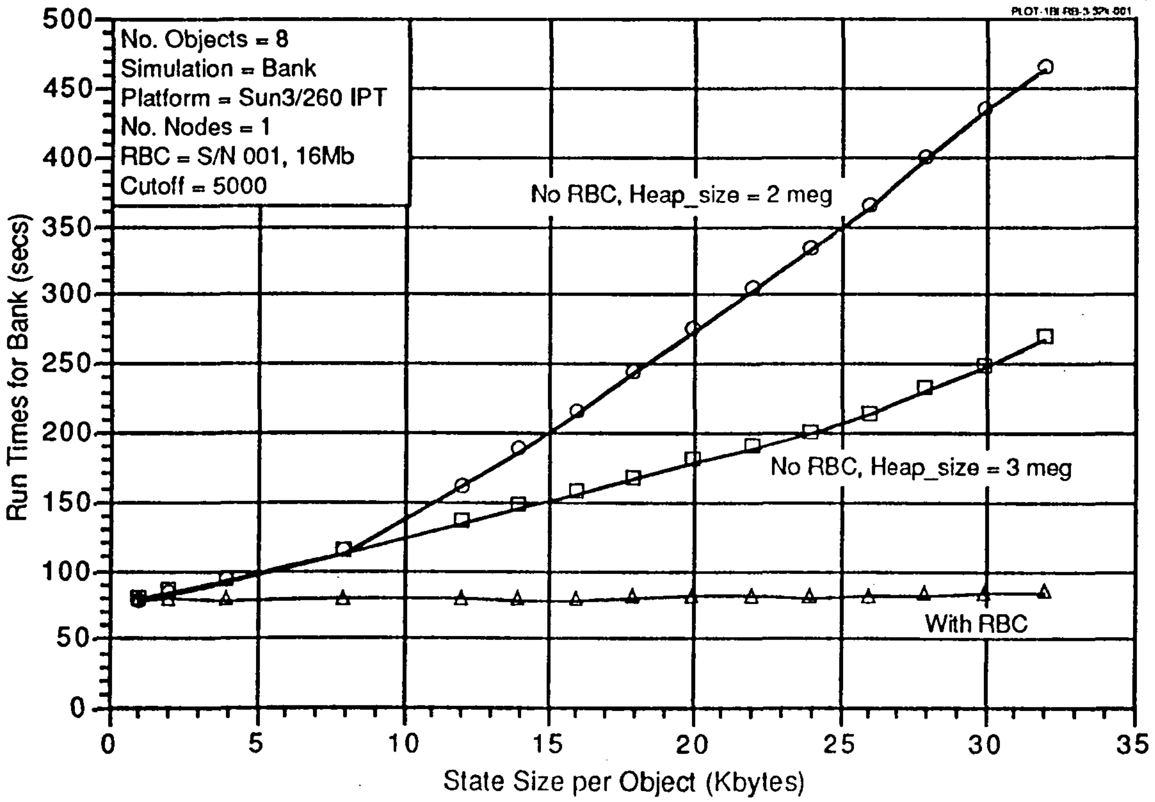

Figure 5, Effect of State Size on Elapsed Time Through 32 Kbytes/object

Of interest with respect to the two graphs are the following observations:

- From Figure 5, RBC speed up for state sizes through 32 Kbytes/object is:

  For Heap Size of 2 Meg => ranges from .993 to 5.52

  For Heap Size of 3 Meg => ranges from 1.02 to 3.20

- From Figure 6, estimated speed up through 128 Kbytes/object, for heap size of 3 meg ranges from 1.02 to 15.3
- From Figure 6, estimated speed up through 96 Kbytes/object for heap size of 2 meg ranges from .993 to 22.15
- The effect of Sun 3 heap size on elapsed time is clearly demonstrated in both graphs with the 2 meg heap size failing to run above a state size of 96 Kbytes/object (the simulation appears to run but does not progress).
- As expected, the curve for RBC elapsed time vs state size is essentially flat in Figure 5. As seen in Figure 6, there is some slope associated with the extrapolated curve, which appears to be due to increases in the background advance time as the state sizes become large.

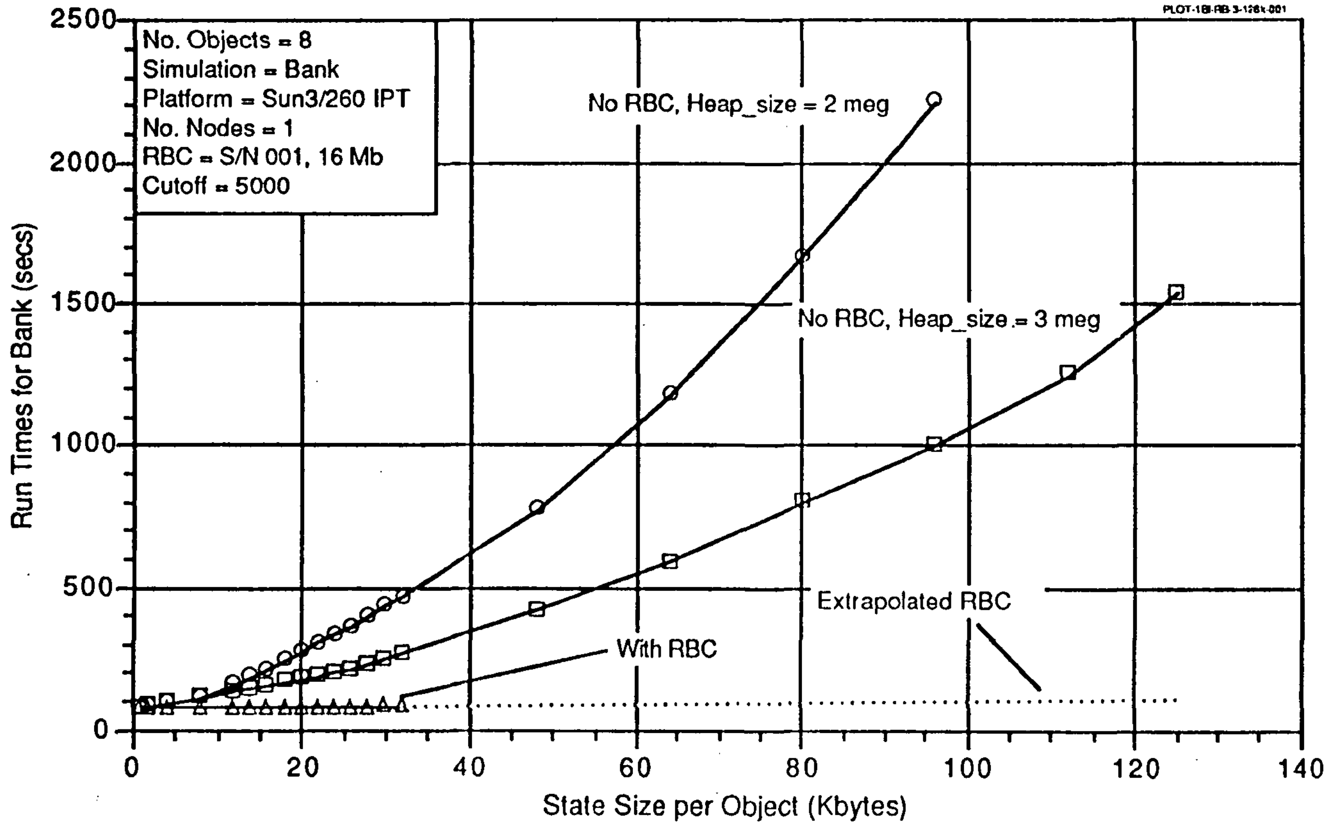

Figure 6, Effect of State Size on Elapsed Time Through 128 Kbytes/object

6.1.3 Copy Time vs Elapsed Time and RBC Savings

In Figure 7, Effect of State Size on Copy Time Through 128 Kbytes/object, the elapsed run time and time spent copying state, in seconds, are plotted as a function of State size per object, for the Sun 3 without the RBC. Two variants of the Sun 3 runs are provided based on the heap size specified for Time Warp.

State copy times for the Sun 3 were obtained by instrumenting the Time Warp state copy routine. Copy time reflects only the time actually spent copying the state vector and does not include memory management processing.

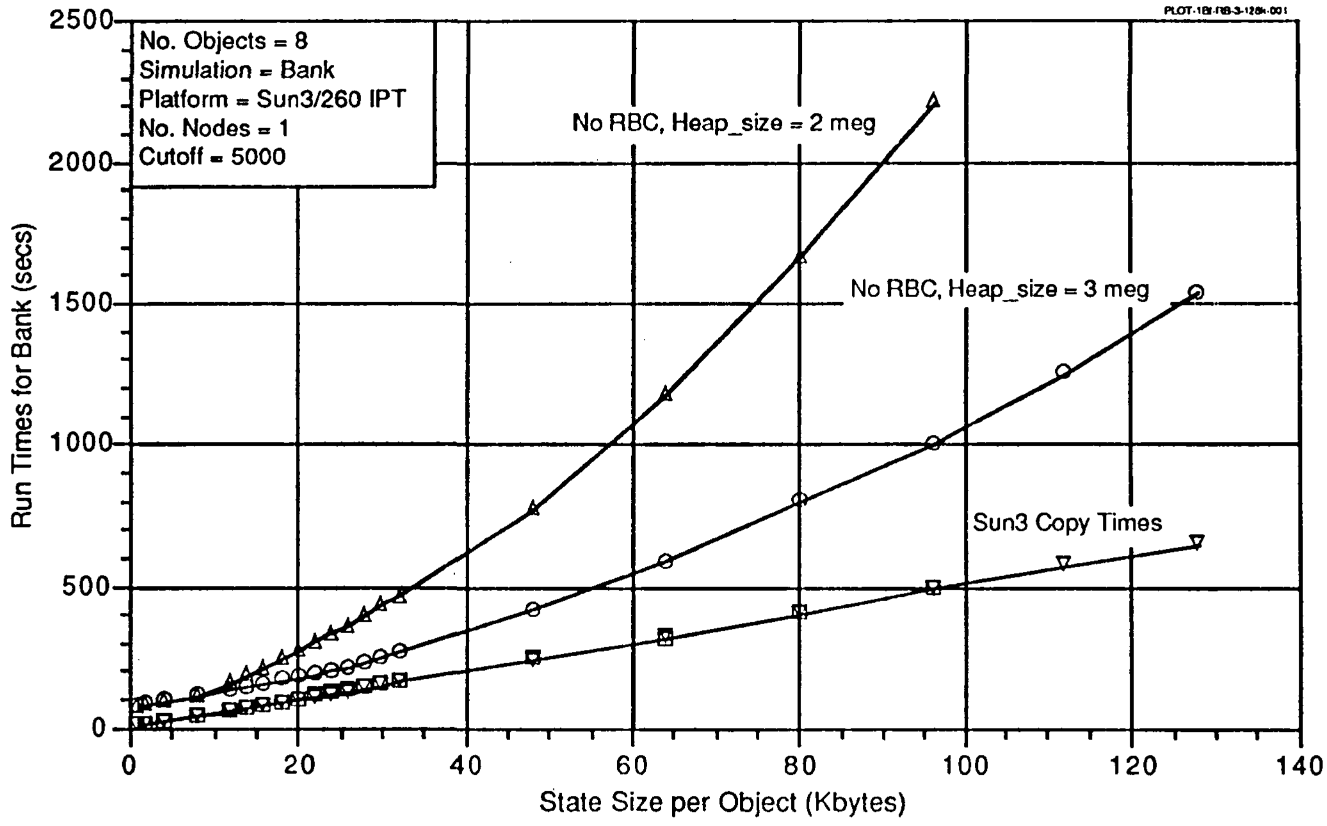

Figure 7, Effect of State Size on Copy Time Through 128 Kbytes/object

The purpose of Figure 7 is to relate state copy time to elapsed time, and to highlight the non-linear behavior associated with the Sun 3 runs, as state size is increased. Clearly both Sun 3 implementations start out linear for small state sizes. Also, the 2 meg heap run deviates from linear much earlier than the 3 meg heap run. We believe that the deviation from linear as state size increases is an artifact of the fragmentation of heap memory, that occurs during the Time Warp runs, and that the increased elapsed time is caused by Time Warp's memory manager allocating and de-allocating memory chunks, to free blocks large enough for the state to reside in. Thus while the increased time is related to state size it is not strictly state copy time. Based on Figure 7, one can envision that for nodes with larger amounts of internal memory the linear region would persist, and that conversely for nodes with smaller amounts of memory the non-linear effect would be accentuated.

In Figure 8, RBC Savings vs Copy Times, the calculated savings for the RBC module, and time spent copying state for Time Warp, in seconds, are plotted as a function of State size per object. Two variants of the Sun 3 runs are provided based on the heap size specified for Time Warp.

The intent of this graph is provide a measure of the effectiveness of the RBC in extracting all of the potential savings which are available in a Time Warp run. The theory behind the RBC module is that it off-loads state copy time and therefore the achievable speed up is limited to extraction of that state copy time alone. Based on the state copy times vs elapsed time presented in Figure 7, we would expect that in addition to the state copy time originally postulated, that the RBC would also save the excess memory management time, due to fragmentation, since with the state saving off-loaded from the node, memory fragmentation should not occur.

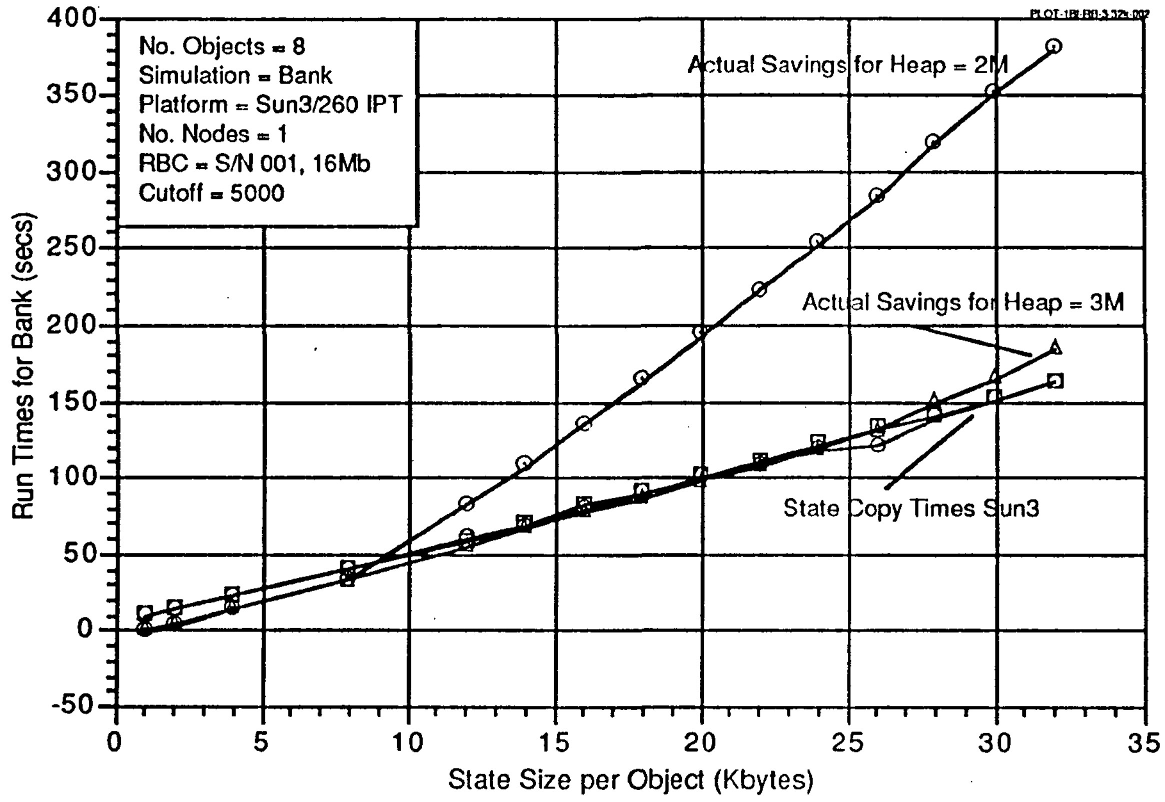

Figure 8, RBC Savings vs Copy Times

The graph presented Figure 8 shows that the RBC does in fact extract essentially all of the state copy time and in addition appears to save the memory management time as well.

6.2 Sun 3 Multi-Node Testing With Bank

Multi-node Sun 3 test results for Bank are provided in Figures 9 through 15. Multi-node testing for Bank was accomplished on a 2-node network rather than 4-node, as originally planned, due to delays in integration in the final months of the effort.

6.2.1 2-Node Sun 3, Effect of Cutoff on Elapsed Time

Referring to Figure 9, 2-Node Effect of Cutoff on Elapsed Time, the elapsed run time, in seconds, is plotted as a function of State size per object, for various cutoffs. This is essentially the same plot as was used for the single node case to show that cutoff has negligible effect on run time, and thus allow a single cutoff value to be used for subsequent runs. This is a plot of Sun 3 elapsed times only, using a 3 megabyte heap. For this application (Bank) on two nodes and with 8 objects (4 objects per node), the total state memory for 128 Kbytes/object (the last plotted point) is 1/2 Megabyte per node. For each cutoff above 1000 two plots are provided. One is the elapsed time as measured, and the second is the calculated elapsed time based on the cutoff of 1000.

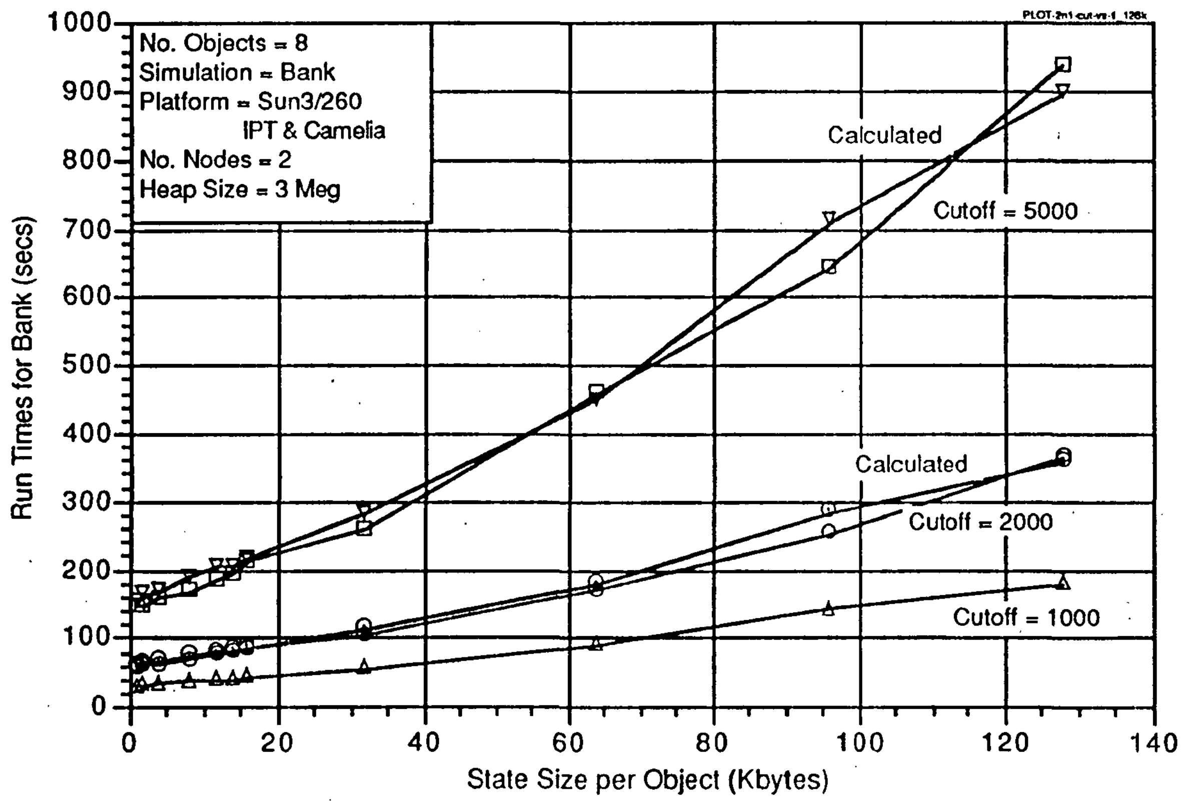

Figure 9, 2-Node Effect of Cutoff on Elapsed Time

Once again all three curves are non-linear in shape, and a high correlation appears between the measured and calculated curves above the cutoff of 1000. Comparison of the correlation with the 1-node measurements shows higher deviation for the 2-node case. This appears to be a result of the higher run-to-run variance in the 2-node measurements which is likely due to the asynchronous nature of the ethernet communications.

6.2.2 2-Node Sun 3, Effect of State Size on Elapsed Time

In Figure 10, 2-Node Effect, of State Size on Elapsed Time, Through 32 Kbytes/object, and Figure 11, 2-Node, Effect of State Size on Elapsed Time, Through 128 Kbytes/object, the elapsed run time, in seconds, is plotted as a function of State size per object, for the Sun 3 2-Node, with and without the RBC.

The RBC modules used in 2-node testing are implemented with 4 megabytes of installed memory. For the 4 Mbyte RBC variant the maximum state size per object, for Bank with 8 objects (4 objects per node), is 16 Kbytes (16 Kbytes * 4 Objects * 64 states = 4 Mbytes). Figure 10 provides the measured times for the 2-node RBC configuration. Figure 11 provides additional test results for the Sun 3 and an extrapolation of the RBC times up to 1/2 megabyte of state.

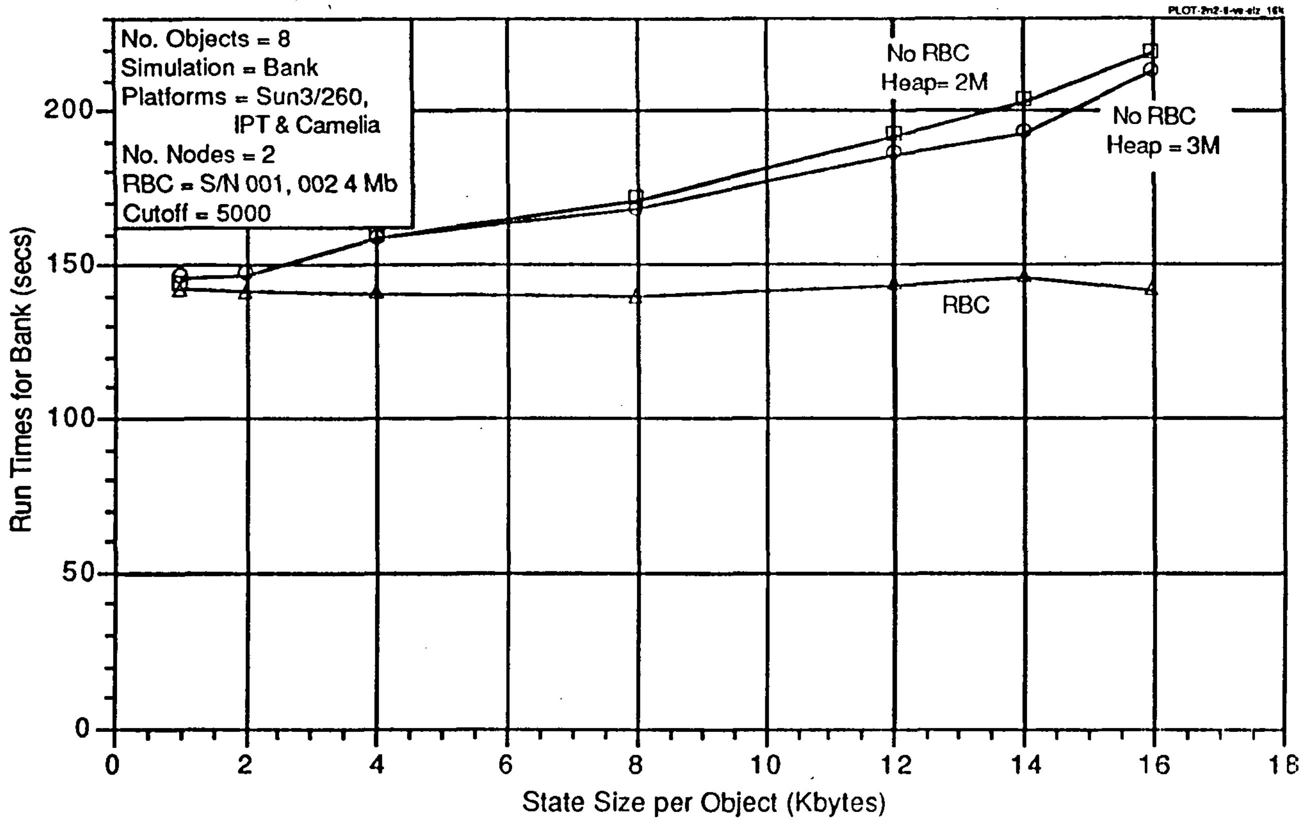

Figure 10 2-Node, Effect of State Size on Elapsed Time Through 32 Kbytes/object

Of interest with respect to the two graphs are the following observations:

- From Figure 10, RBC speed up for state sizes through 16 K/object is:

  For Heap Size of 2 Meg => ranges from 1.01 to 1.54

  For Heap Size of 3 Meg => ranges from 1.02 to 1.50

- From Figure 11, estimated speed up through 128 Kbytes/object, for heap size of 3 meg ranges from 1.02 to 6.6

- From Figure 11, estimated speed up through 96 Kbytes/object for heap size of 2 meg ranges from 1.01 to 8.18

- Absolute speed up for the 2-node case when compared to the 1-node results appear to show that the RBC performs less than half as well with the 2-node case. As will be shown in Figures 12 and 13, in the following section, this is a direct result of the percentage of state saving time going down as a result of the increase in communication time. In the best case the RBC can only reduce the elapsed run time by an amount equal to the state copying time and potentially the memory management time due to fragmentation.

  Thus if a simulation consumes 90% of its time in state saving the RBC has a profound effect on speedup. If however that simulation spends 10% of its time in state saving the RBC will have negligible effect on speedup.

- The effect of Sun 3 heap size on elapsed time is clearly demonstrated in Figure 11 but is not apparent in Figure 10. This is because the aggregate state size on each node is reduced by half from that of the single node test when the objects are split between the 2 nodes.

- As expected, the curve for RBC elapsed time vs state size is essentially flat in Figure 10, although as was shown with Figure 9 the deviation of the curves is more marked. Also, as was the case in the single node test, there is some slope associated with the extrapolated curve in Figure 11, which again appears to be due to increases in the background advance time as the state sizes become large.

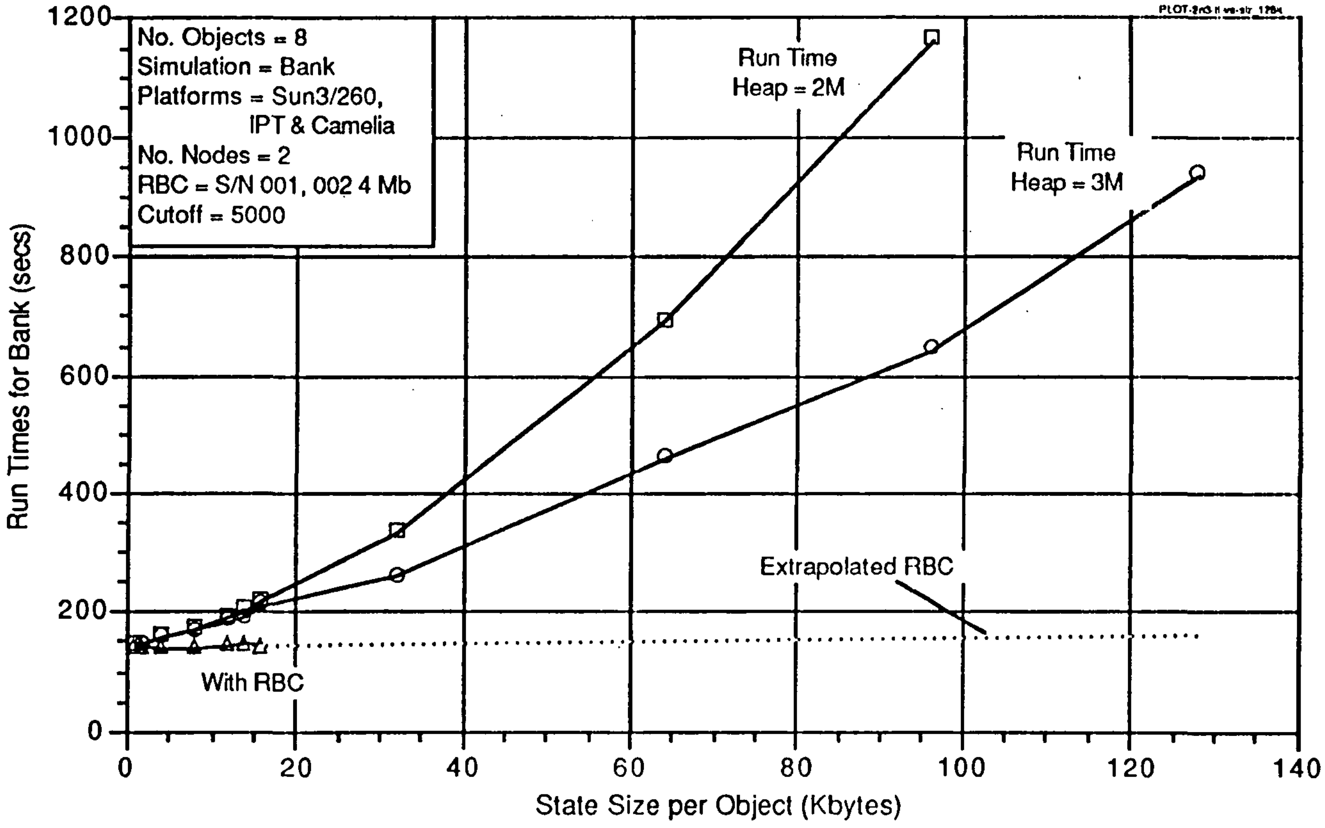

Figure 11, 2-Node, Effect of State Size on Elapsed Time, Through 128 Kbytes/object

6.2.3 2-Node Sun 3, Copy Time vs Elapsed Time and RBC Savings

In Figure 12, 2-Node, Effect of State Size on Copy Time Through 128 Kbytes/object, the elapsed run time and time spent copying state, in seconds, are plotted as a function of State size per object, for the Sun 3 without the RBC. Two variants of the Sun 3 runs are provided based on the heap size specified for Time Warp.

As with Figure 7 for the single node case, the purpose of Figure 12 is to relate state copy time to elapsed time and to highlight the non-linear behavior associated with the Sun 3 runs as state size is increased. Also as with all of the 2-node runs a higher random deviation is shown than for the single node runs. Clearly both Sun 3 implementations start out linear for small state sizes. Additionally, the 2 meg heap run deviates from linear much earlier than the 3 meg heap run, and because of the reduction in objects (and thus state memory) on each node, the deviation from linear occurs after the 16 K/object cutoff point for the RBC modules.

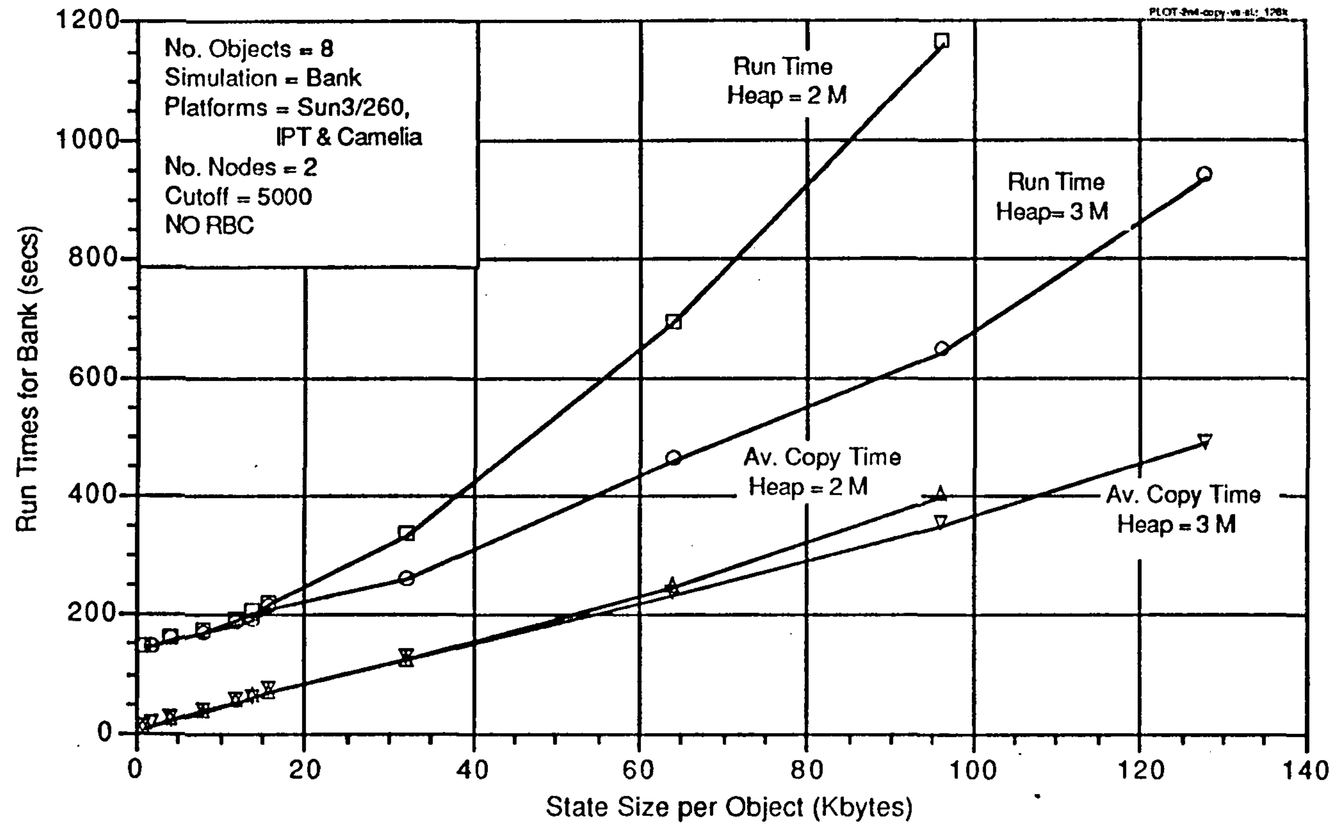

Figure 12, 2-Node, Effect of State Size on Copy Time Through 128 Kbytes/object

In Figure 13, 2-Node, RBC Savings vs Copy Times, the calculated savings for the RBC module and time spent copying state for Time Warp, in seconds, are plotted as a function of State size per object.

Of interest with respect to the graph are the following observations:

- As with the single node case, the RBC appears to extract the majority of the copy time in savings. However there is a larger difference then for the single node case. There are two reasons for this:

- When running Time Warp on a single node, advances and marks occur but rollbacks do not. Thus the overhead associated with rollbacks both in the RBC and the rbc_driver are not included in the elapsed time.

- Because the 2-node implementation splits the 8 objects between the nodes, resulting in 1/2 the state memory per node, the Sun 3 run times are essentially linear in the region shown in the graph. Thus off-loading the state memory to the RBC does not result in a reduction in memory management overhead, as was shown in the one node case.

- As is readily seen in Figure 13, the copy time accounts for less than 1/3 of the overall elapsed time thus limiting the absolute speed up that the RBC can achieve.

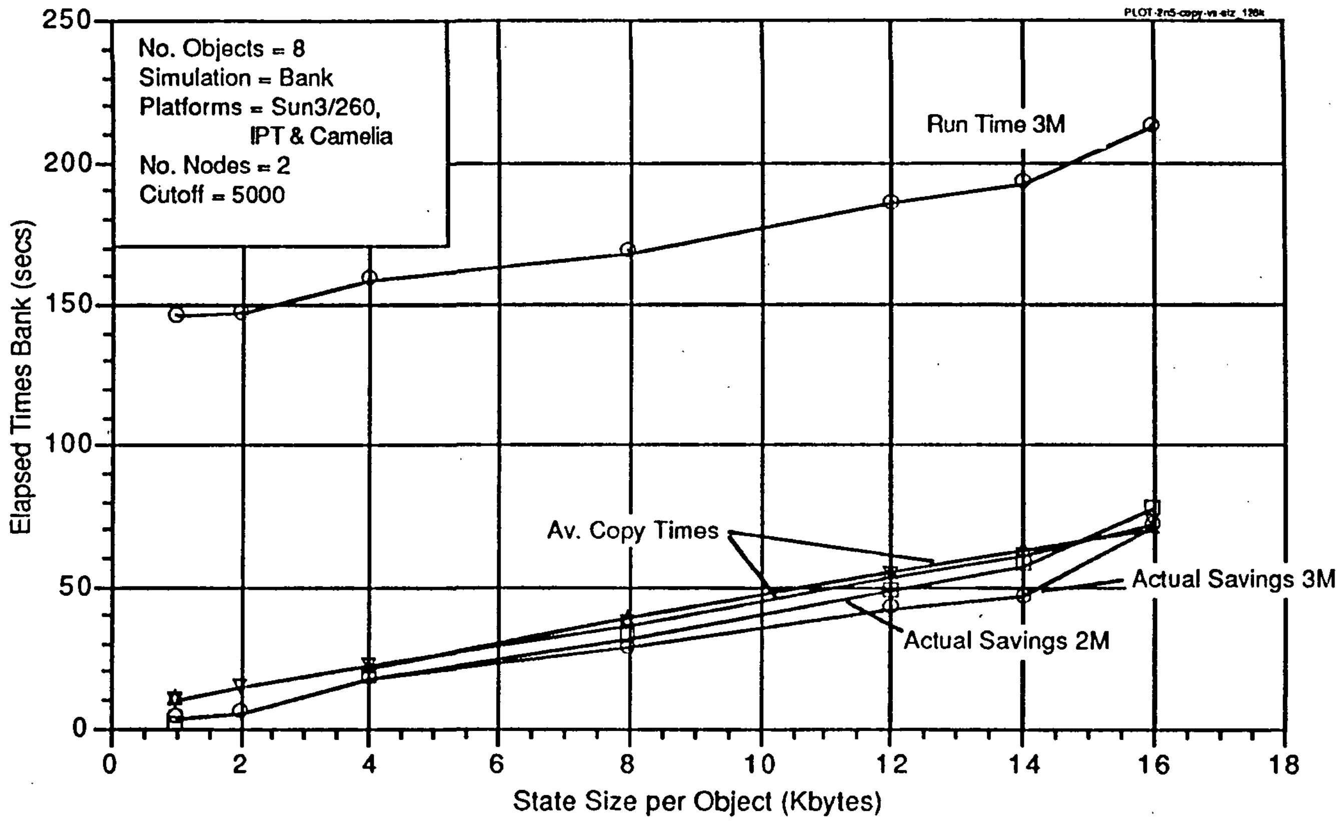

Figure 13, 2-Node, RBC Savings vs Copy Times

6.2.4 2-Node Sun 3, RBC vs Sun 3 Object Times

Referring to Figure 14, 2-Node RBC vs Sun 3 Object Times, a graphical depiction of the increased state read and write times for the RBC over the Sun 3 is provided. In Figure 14, RBC object times are shown with black symbols. Both the committed and uncommitted object times are generally greater for the RBC. The only difference computationally between an event's elapsed time with the RBC versus the Sun 3 is the state memory access time and the overhead associated with RBC command communication.

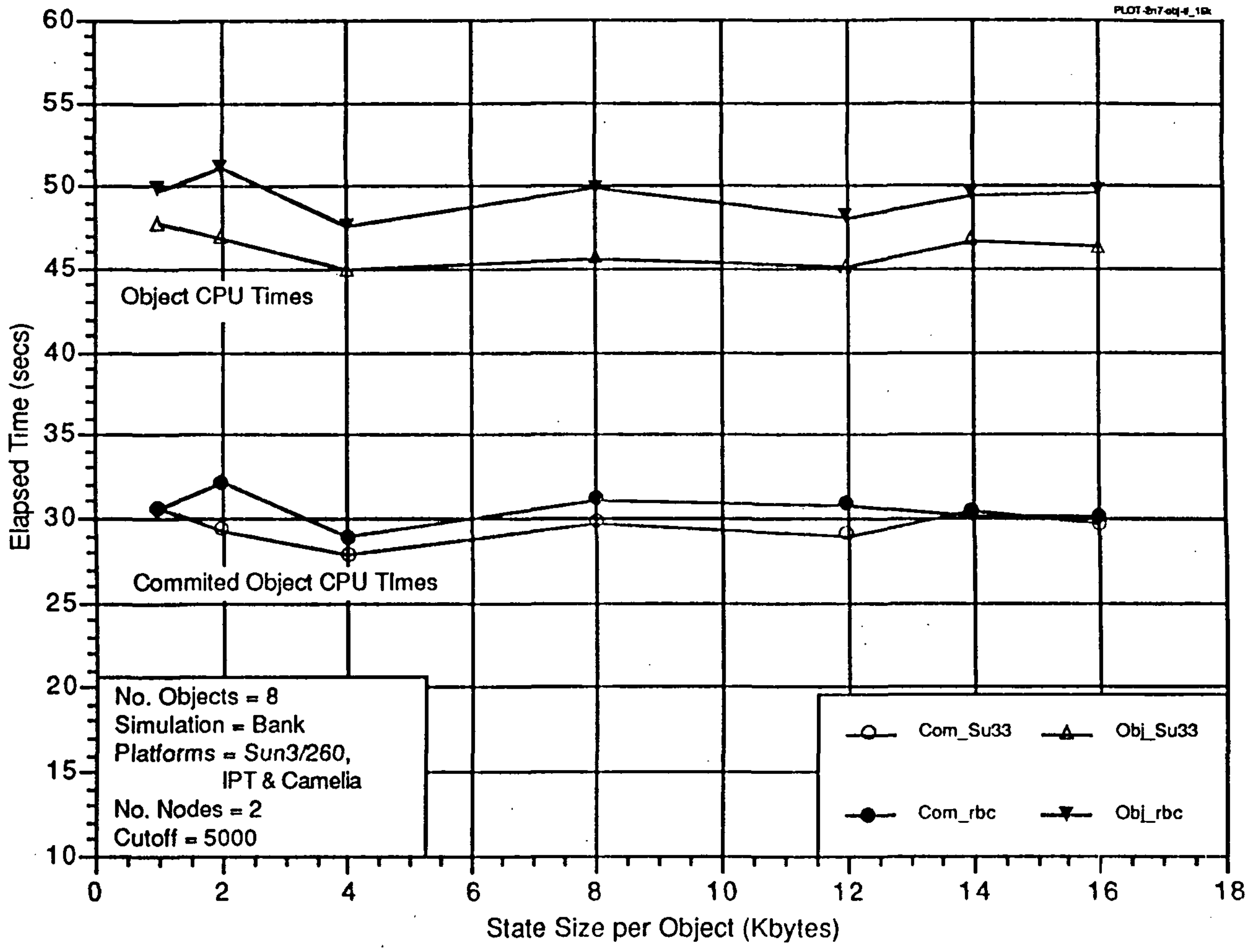

Figure 14, 2-Node RBC vs Sun 3 Object Times

6.2.5 2-Node Sun 3, RBC vs Sun 3 Event Statistics

Referring to Figure 15, 2-Node RBC vs Sun 3 Event Statistics, a graphical depiction of the number of events started, completed, rolled back, and committed for the RBC and Sun 3 are provided. Once again the RBC statistics are shown with black symbols. Of interest with respect to Figure 15 is that the RBC has generally increased event activity over that of the Sun 3 for all state sizes above 2 K/object. Additionally, the trend appears to show a widening gap between the RBC and Sun 3 as the state size increases. This phenomenon is not clearly understood but appears to be explained by the fact that since less time is spent in state saving for a fixed communication topology, the objects spend more time processing and therefore tend to run further ahead in simulation time. Since the objects are generally moving further ahead more events are started, completed and of course rolled back.

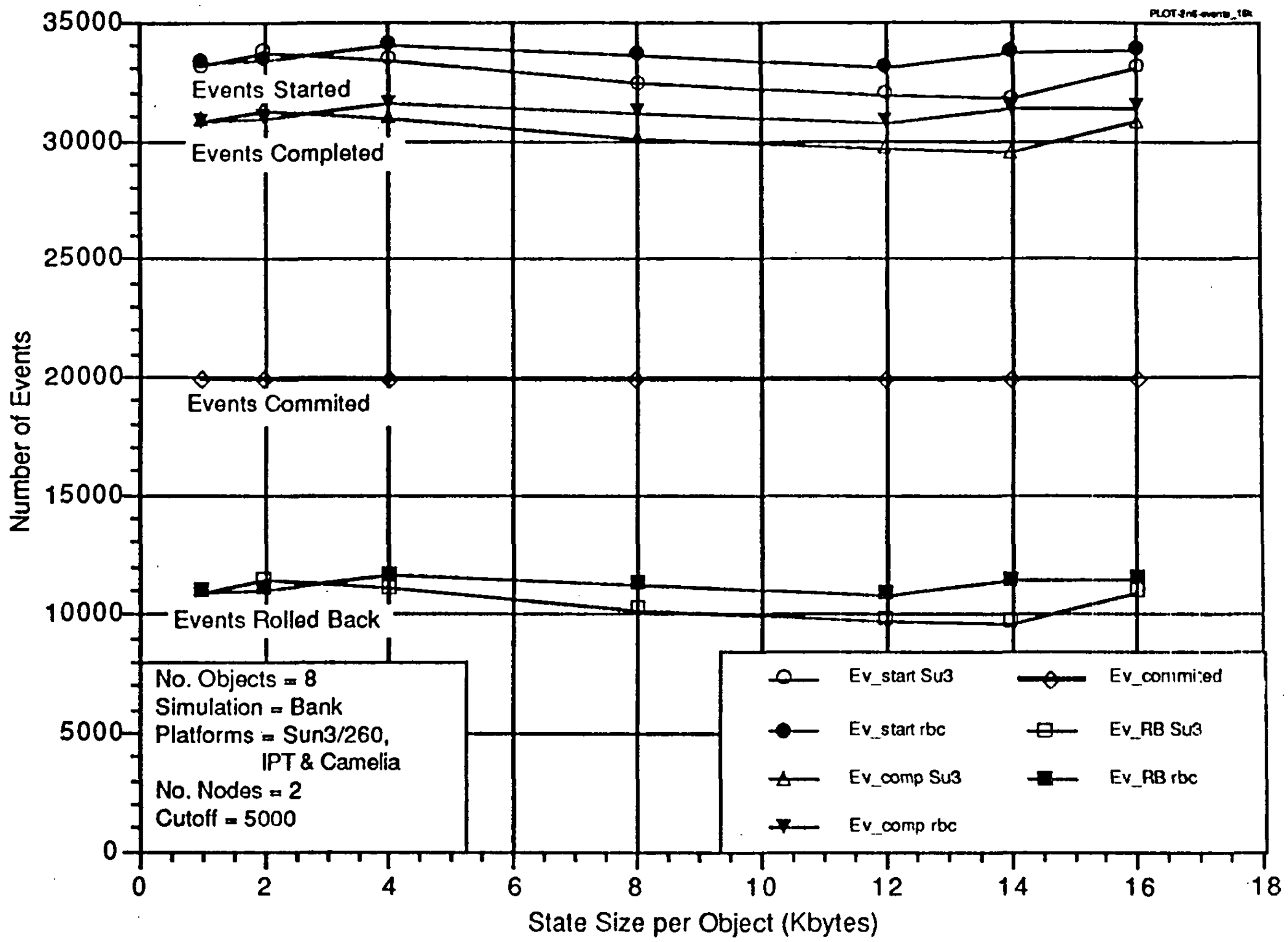

Figure 15, 2-Node RBC vs Sun 3 Event Statistics

6.3 Sun 3 Multi-Node Testing With Pucks

Multi-node Sun 3 test results for Pucks are provided in Figures 16 and 17. Multi-node testing for Pucks was accomplished on a 2-node network rather than 4-node, as originally planned, due to delays in integration in the final months of the effort.

In Figure 16, 2-Node Pucks, RBC vs Sun 3 Elapsed Time, the elapsed run time, in seconds, is plotted as a function of the test run number, for the Sun 3 with and without the RBC. Additionally the average elapsed time is also plotted for each set of runs. Plots of elapsed time versus state size are not provided because Pucks does not have the capability to vary its state size per object as Bank does.

In Figure 17, 2-Node Pucks, Savings vs Copy Time, the average state copy time, RBC savings, and object times, in seconds, are plotted as a function of test run number, for the Sun 3 with and without the RBC.

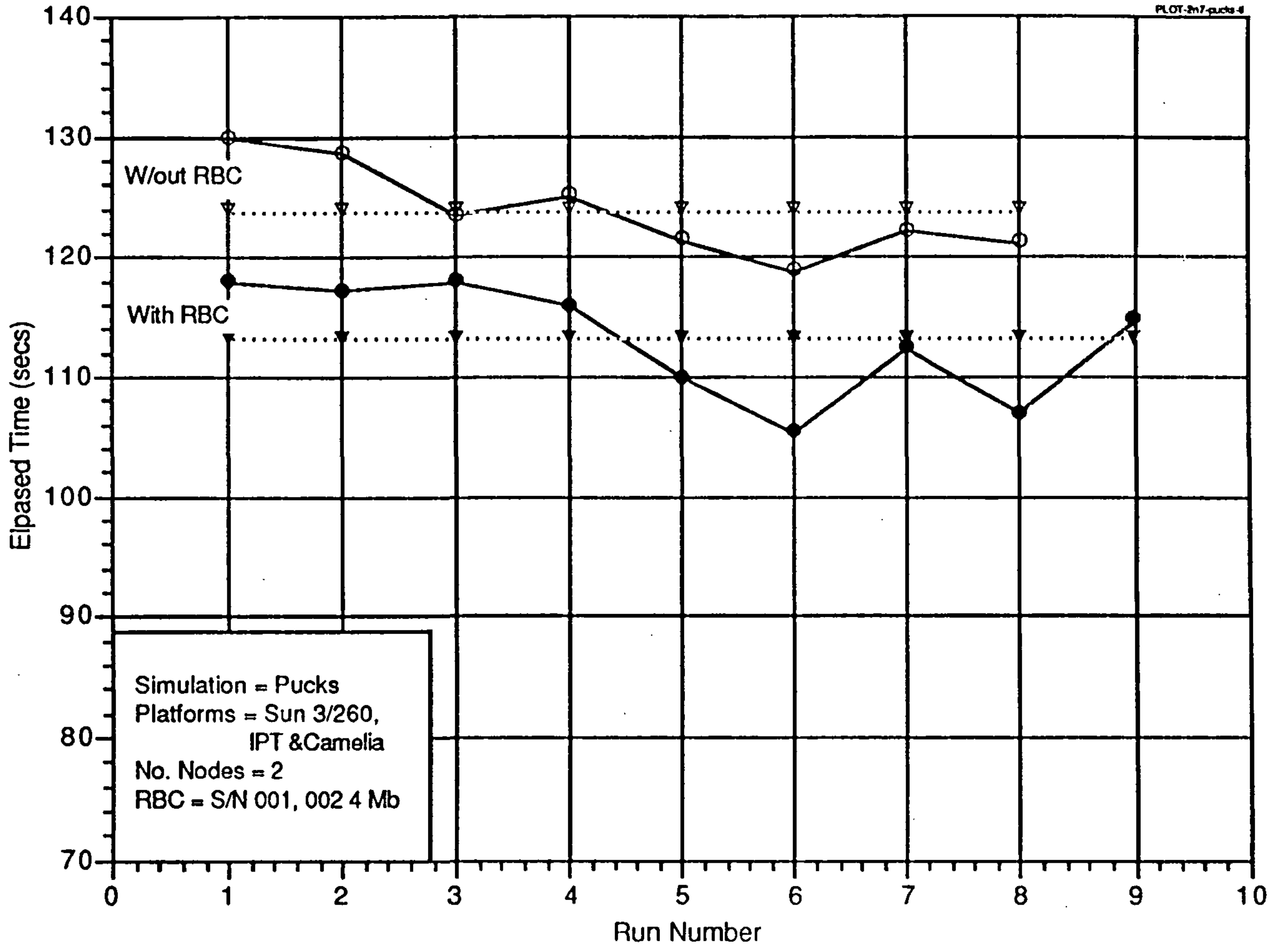

Figure 16, 2-Node Pucks, RBC vs Sun 3 Elapsed Time,

Of interest with respect to the two graphs are the following observations:

- In Figure 16, the average speedup ratio for the RBC implementation over the RBC is only 1.09.

- As is readily seen in Figure 17, the average copy time for pucks accounts for only 15 seconds out of the 123 second run time. Thus state copy time accounts for less than 10 % of the overall elapsed time thus limiting the absolute speed up that the RBC can achieve

- Also in Figure 17, the RBC appears to be "saving" only 11 seconds of the 15 available as a result of the state copying. This difference is not in line with what we expect based on the results for Bank previously examined. The apparent discrepancy is resolved by comparing Bank and Pucks state read/write behavior. Pucks performs many more state reads and writes than Bank does during a simulation run. Our estimate is that Pucks essentially reads the entire state on each event. As discussed in section 5.1.1 the RBC prototypes have a slower access time when performing state reads and writes then standard Sun 3 memory. This slower access time coupled with the large number of state accesses results in the reduced copy time savings.

The increased aggregate state access times for the RBC is also reflected in Figure 17 in the 4 -5 second differential between object times.

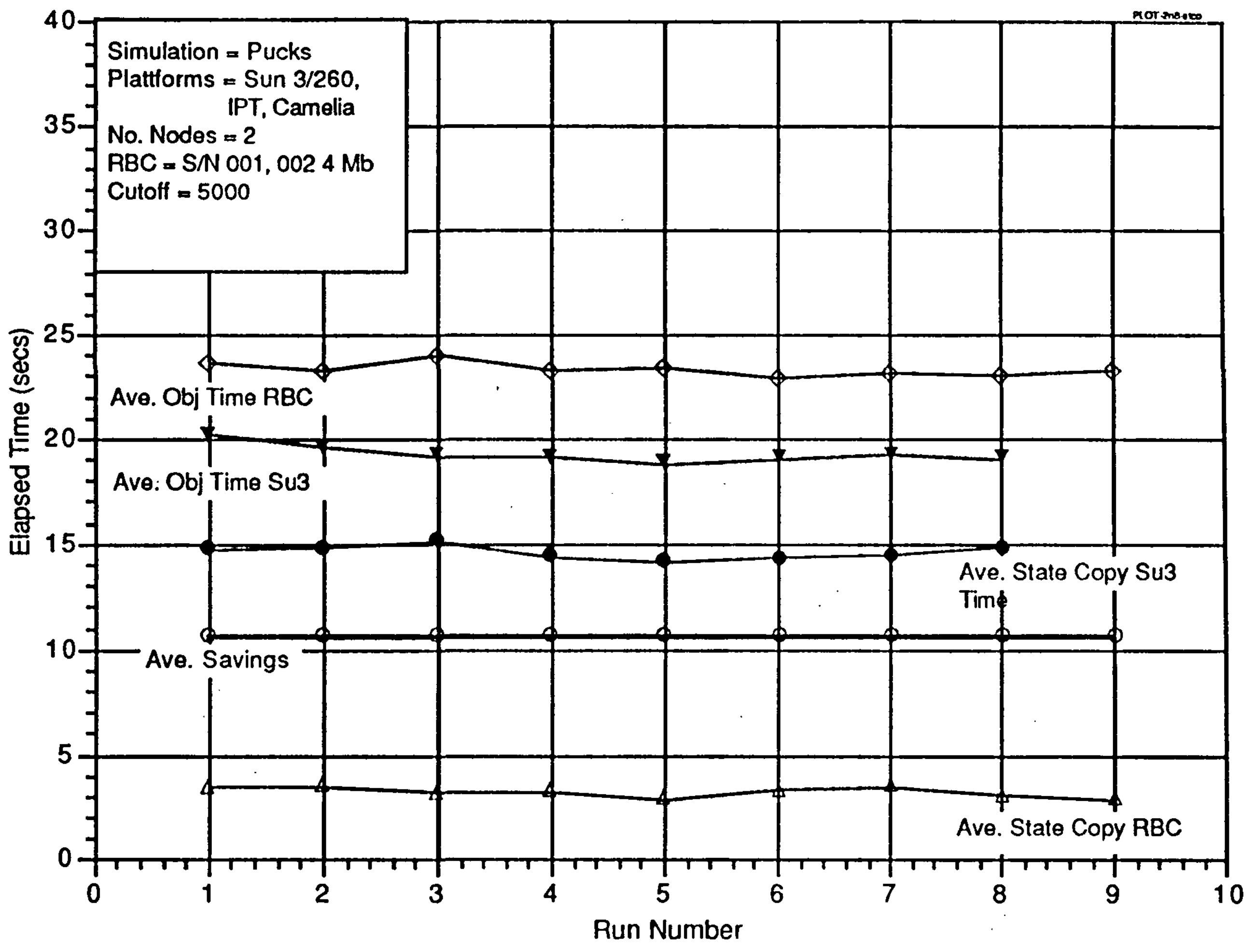

Figure 17, 2-Node Pucks, Savings vs Copy Time

6.4 Sun 4 Single Node Testing

Single node, Sun 4 test results for Bank are provided in Figures 18 through 23. Note that the intent of the Sun 4 testing is primarily to verify that the RBC can be interfaced to a standard VME device with little or no modification, and to identify any functional errors in the module's performance. The testing performed in the Phase II effort has shown that the RBC can be interfaced to different VME interfaces with minimal modifications. The one area of potential concern is the variation in VME bus time-out times, which is discussed in more detail in Section 7, Conclusions and Estimate of Technical Feasibility. Additionally, some anomalies in RBC timing have been noted as a result of the Sun 4 testing, as noted in the following discussions, particularly in the advance background task.

6.4.1 Sun 4, Effect of State Size on Elapsed Time

In Figure 18, Sun 4, Effect of State Size on Elapsed Time Through 32 Kbytes/object, and Figure 19, Sun 4, Effect of State Size on Elapsed Time Through 128 Kbytes/object, the elapsed run time for Bank, in seconds, is plotted as a function of State size per object, for the Sun 4 with and without the RBC.

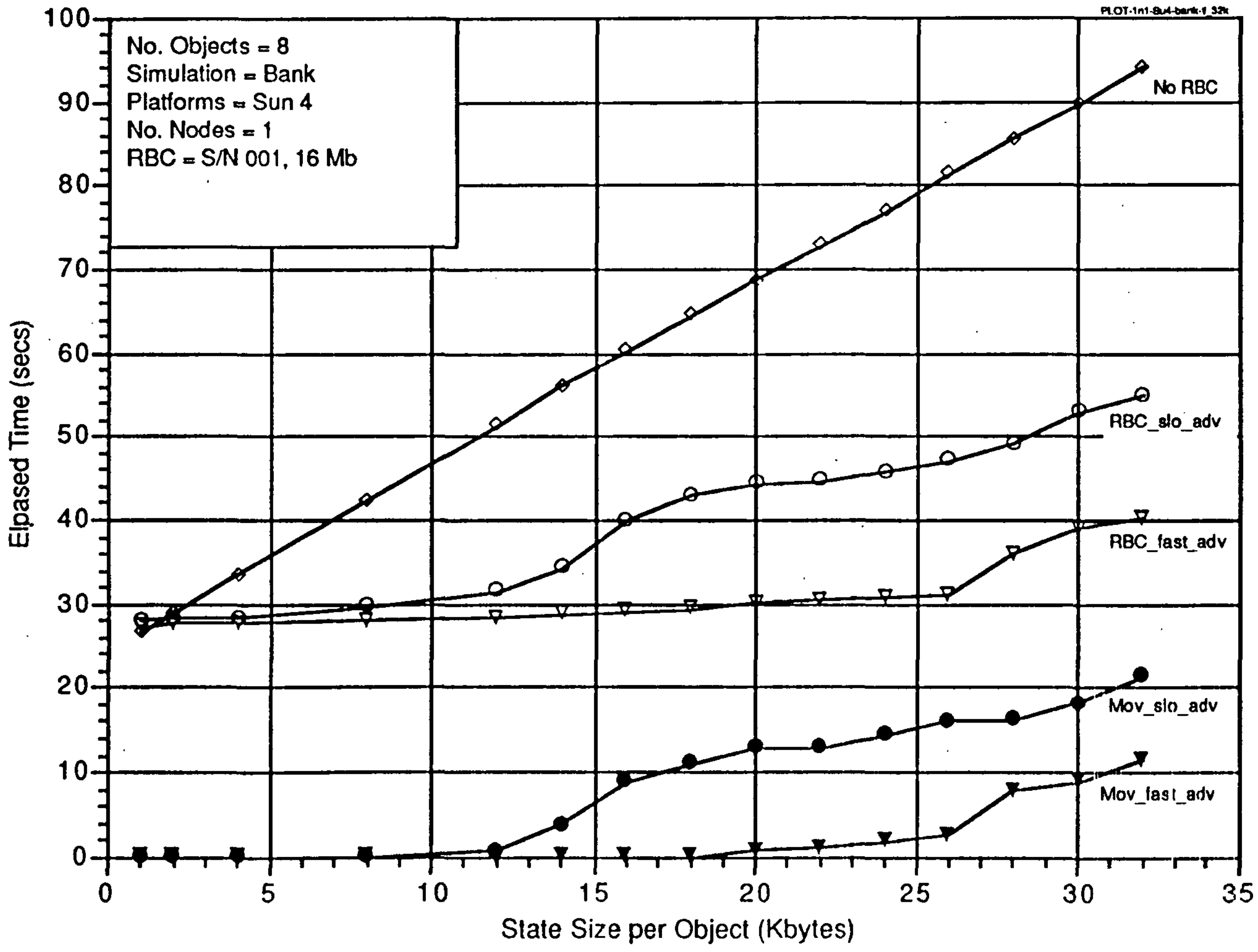

Figure 18, Sun 4, Effect of State Size on Elapsed Time Through 32 Kbytes/object, In Figure 18 and 19, two separate RBC elapsed time curves are presented, designated as RBC_slo_adv and RBC_fast_adv. These two curves represent the elapsed time for the RBC when running with two different versions of the DSP firmware. RBC_slo_adv is the DSP code as coded in C. RBC_fast_adv is the DSP code with the update portion of the advance background loop coded in DSP assembly. Additionally, Figure 18 provides two normalized curves designated as Mov_slo_adv and Mov_fast_adv. These two curves provide the number of "mark rollovers" (movers) for each of the DSP code versions shown. Mark rollovers are an absolute number and not an elapsed time value, and therefore have been normalized to a maximum of approximately 20 for display purposes only.

Mark rollovers occur when the node requests that the RBC save or "mark" a frame but the RBC fails to comply because no frames are available. The term "mark overflow" is derived from the error message "frame overflow on mark" which is returned by the DSP. Mark overflows occur when the DSP, in attempting to perform a background advance, fails to keep up with node's requests for new frames. As an example:

- At an arbitrary time in the simulation, the CMF is 56, the OMF is 5, and no rollbacks or advances are in progress.

- At GVT update the Node issues a request to advance 10 frames. The RBC accepts the advance command and initiates a background advance. The New_OMF for the segment will be 15 at the completion of the advance.

- While the DSP performs the background advance, the simulation progresses and the node issues mark commands, of distance 1, to the RBC. The first 12 marks are processed with no problem and the resultant CMF is now 4 (CMF has rolled over the mark frame stack). On the next mark attempt, if the background advance has still not completed, the RBC cannot complete the mark command because the CMF will over run the OMF. If this were allowed the node would then write over state memory data in the OMF frame.

- RBC to node protocol provides that on a mark overflow, the node will retry the command at a later time and is essentially waiting for the RBC to complete its background advance.

Of interest with respect to the two graphs are the following observations:

- From Figure 18, RBC speed up for state sizes through 32 Kbytes/object ranges from .95 to 1.71 for the RBC_slo_adv trace and from .975 to 2.33 for the RBC_fast_adv trace.

- From Figure 19, estimated speed up through 128 Kbytes/object ranges from .95 to 2.31 for the RBC_slo_adv trace and from .975 to 7.17 for the RBC_fast_adv trace.

- From Figure 18, Sun 4 elapsed times are essentially linear for increasing state size through 32 Kbytes/object. From Figure 19, non-linear behavior of the elapsed time trace does not become apparent until approximately 80 Kbytes/object.

- From Figure 18, The curve for RBC_slo_adv deviates considerably from the flat characteristic exhibited in the Sun 3 plots discussed previously. The RBC_slo_adv appears to show a very high correlation with the curve for mark rollovers, indicating that the node is waiting for the RBC to free frames so that the simulation can continue.

- From Figure 18, The curve for RBC_fast_adv also deviates from the expected flat characteristic, however it is clear that the deviation occurs only for state sizes above 25 Kbytes/object, well above the 12 K/object deviation point for the RBC_slo_adv code. As with the RBC_slo_adv curve, high correlation exists between the deviation from the flat characteristic and the occurrence of mark rollovers.

Thus, we conclude that insufficient frames, due to the slow execution of the advance background loop, is the likely cause of the deviation in overall speed up, and that the incidence of mark rollovers increases with increasing state size. It also appears that the potential for this condition is present in both the Sun 3 and Sun 4 implementations. A partial solution has been implemented and results shown in Figure 18. Based on the modifications to the advance background loop we believe further improvements in performance are possible with more extensive changes from C to Assembly.

- From Figure 18, extrapolated results for both the RBC_slo and RBC_fast are provided. The RBC_slo extrapolation assumes a straight line fit through all points plotted. The extrapolation for RBC_fast assumes a straight line without inclusion of the last 3 plotted points (i.e., assumes that straight line behavior holds).

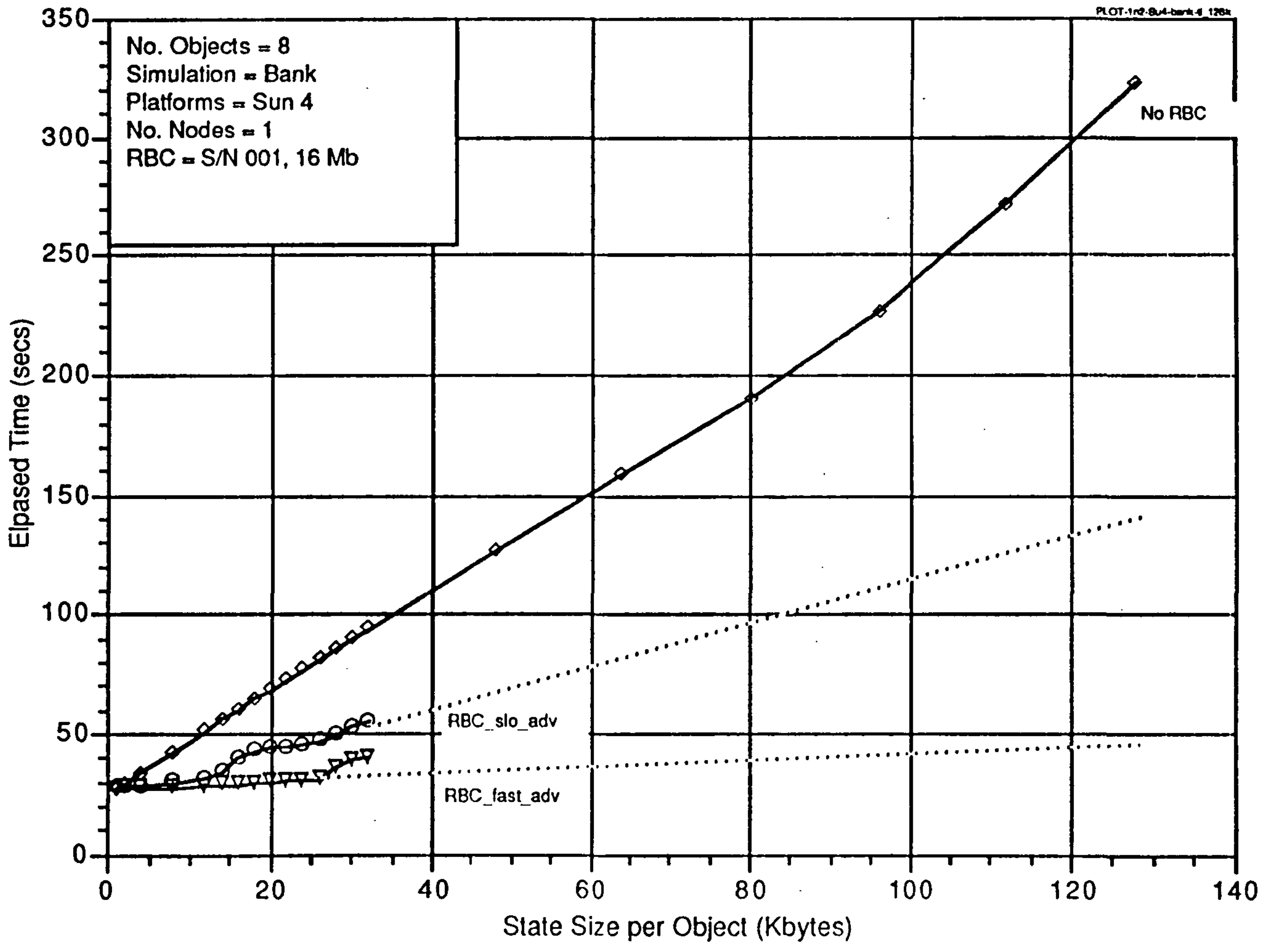

Figure 19, Sun 4, Effect of State Size on Elapsed Time Through 128 Kbytes/object

6.4.2 Sun 4, Copy Time vs Elapsed Time and RBC Savings

In Figure 20, Sun 4, RBC Savings vs Copy Times, the calculated savings provided by the RBC module, the times spent copying state and elapsed run time for the Sun 4, in seconds, are plotted as a function of State size per object.

As with the single node Sun 3 graph of Figure 7, the intent of this graph is provide a measure of the effectiveness of the RBC in extracting all of the potential savings which are available in a Time Warp run. In Figure 20, the curve labeled RBC run Time is from the RBC_fast_adv of Figure 18, and the curve labeled Su4 Run Time is the Sun 4 without RBC of Figure 18. As indicated in the discussion of Figure 18, the Sun 4 run time is essentially linear and tracks with the Sun 4 copy time. Also, Sun 4 copy time consumes approximately 2/3 of the elapsed time and the RBC savings plot shows good correlation with the copy time for state sizes under 25 Kbytes/object. After the 25 Kbyte/object point the RBC savings deviates significantly from the Sun 4 copy time, as expected.

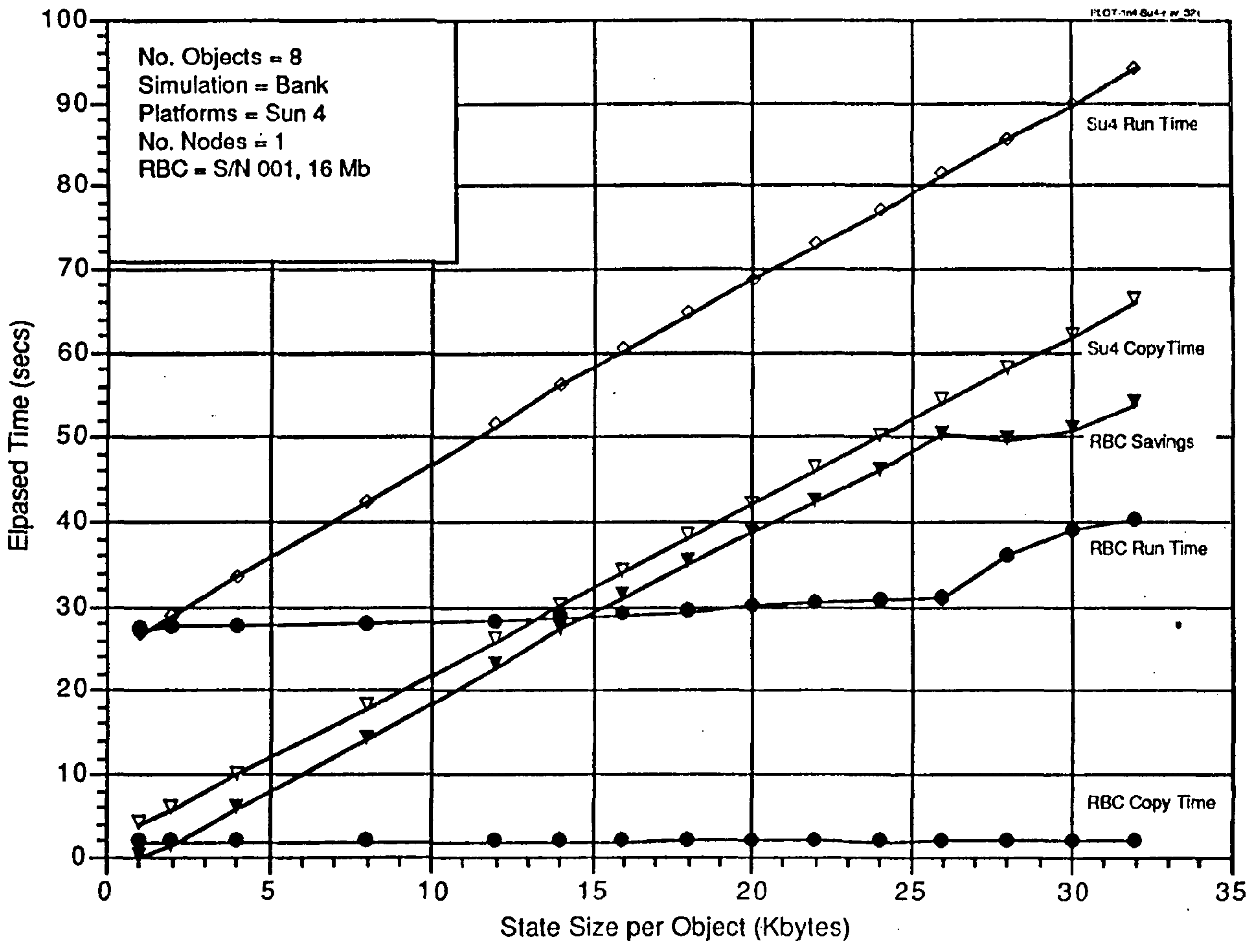

Figure 20, Sun 4, RBC Savings vs Copy Times

6.4.3 Sun 4, Single Node Testing With Pucks

Single node Sun 4 test results for Pucks are provided in Figure 21.

In Figure 21, RBC vs Sun 4 Elapsed Time for Pucks, the elapsed run time, object time and state copying time, in seconds, are plotted as a function of test run number, for the Sun 4, with and without the RBC. Plots of elapsed time versus state size are not provided because Pucks does not have the capability to vary its state size per object as Bank does.

Of interest in Figure 21 is that the elapsed time for the RBC actually exceeds that for the the Sun 4 by approximately 11 seconds. The reasons for this apparent discrepancy in RBC performance are:

- As is readily seen in Figure 21, the average state copy time (St_cp Su4) for pucks with the Sun 4 accounts for 15 seconds out of the 23 second run time. Thus state copy time limits the potential RBC savings to 15 seconds or less.

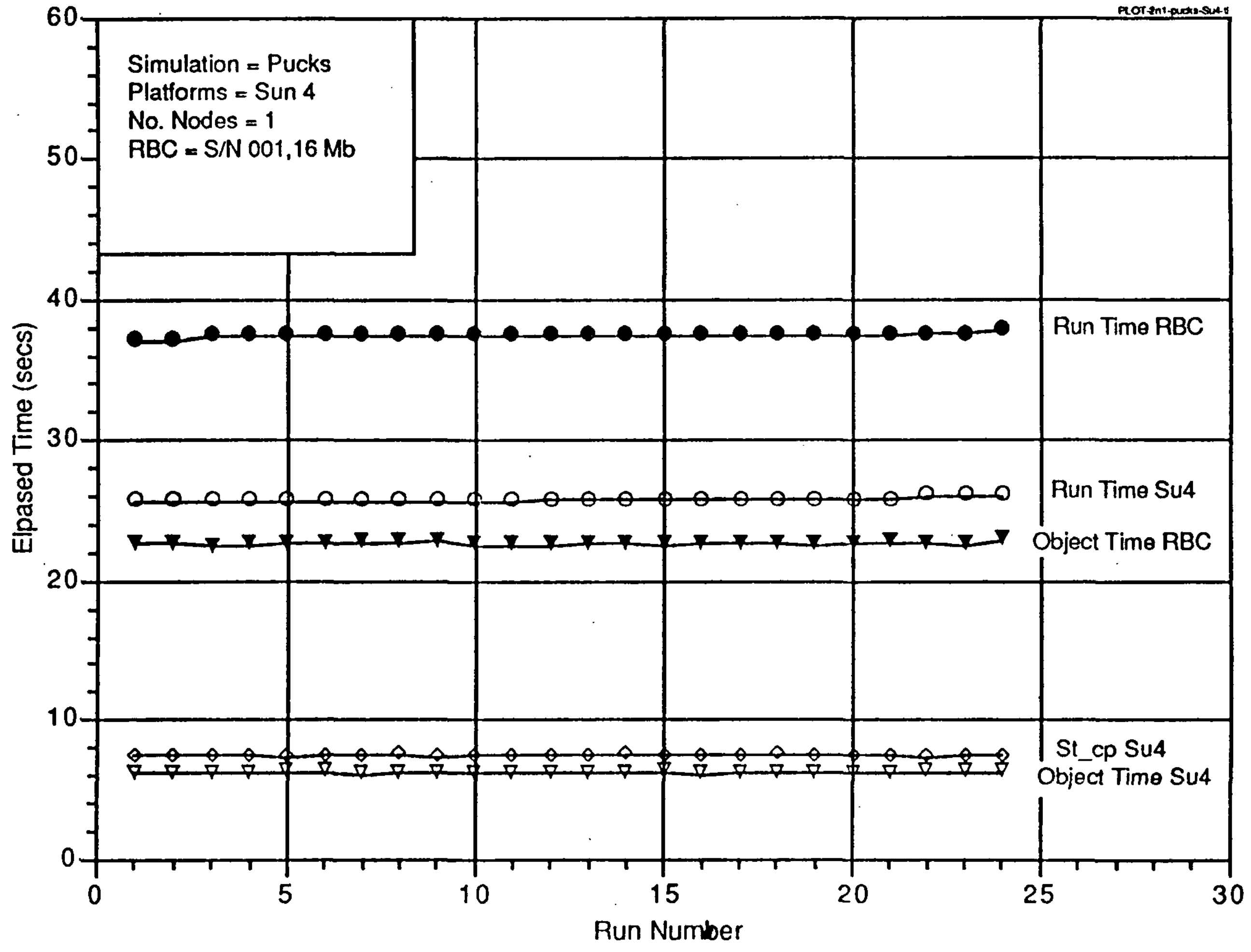

Figure 21, RBC vs Sun 4 Elapsed Time for Pucks

- Also in Figure 21, the RBC object time is approximately 21 seconds while the Sun 4 object time is only 6 seconds. The discrepancy in both object times and elapsed run times is resolved by recalling the state read/write behavior of Pucks and the configuration of the Sun 4 test system.

As mentioned in the 2-node Sun 3 results, Pucks essentially reads the entire state vector on each event. However for the Sun 4 test configuration each state read or write must travel over the Sbus to VMEbus translator to reach the RBC. The translation latency in the Sbus to VME bus link is approximately 1 µsec. Additionally, the RBC itself has a slower response time than standard Sun 4 memory and the IPX is further enhanced with a 40 mhz, 64 Kbyte on-board cache. Clearly under the limited test conditions used, the RBC cannot compete with the Sun 4 's memory speed. The high number of state accesses in Pucks merely enhances the effect.

Thus in Figure 21, we see that the difference between the run times is -11 secs and the primary contributors to that difference are the -15 seconds of object time difference and some fraction of the +6 seconds of state copy time.

The intent of this test with the Sun 4 is primarily to verify that the RBC can be interfaced to a standard VME device with little or no modification, and to verify that Pucks will run and give correct output results. The large latency inherent in the use of the general purpose translation module precludes its likely use in a deliverable configuration.

6.5 Bank Consolidated Performance

Finally, in Figure 22, Consolidated Elapsed Times, the elapsed run times for Bank on one node, in seconds, is plotted as a function of State size per object, for the Sun 4 and Sun 3, with and without the RBC. In Figure 22, RBC run times are once again shown with black symbols and Sun 3 and Sun 4 with clear symbols. Of particular interest in Figure 22 is the crossover point which occurs at a state size of approximately 26 Kbytes/object, and which shows that the Sun 3 with RBC can out perform a Sun 4 without RBC.

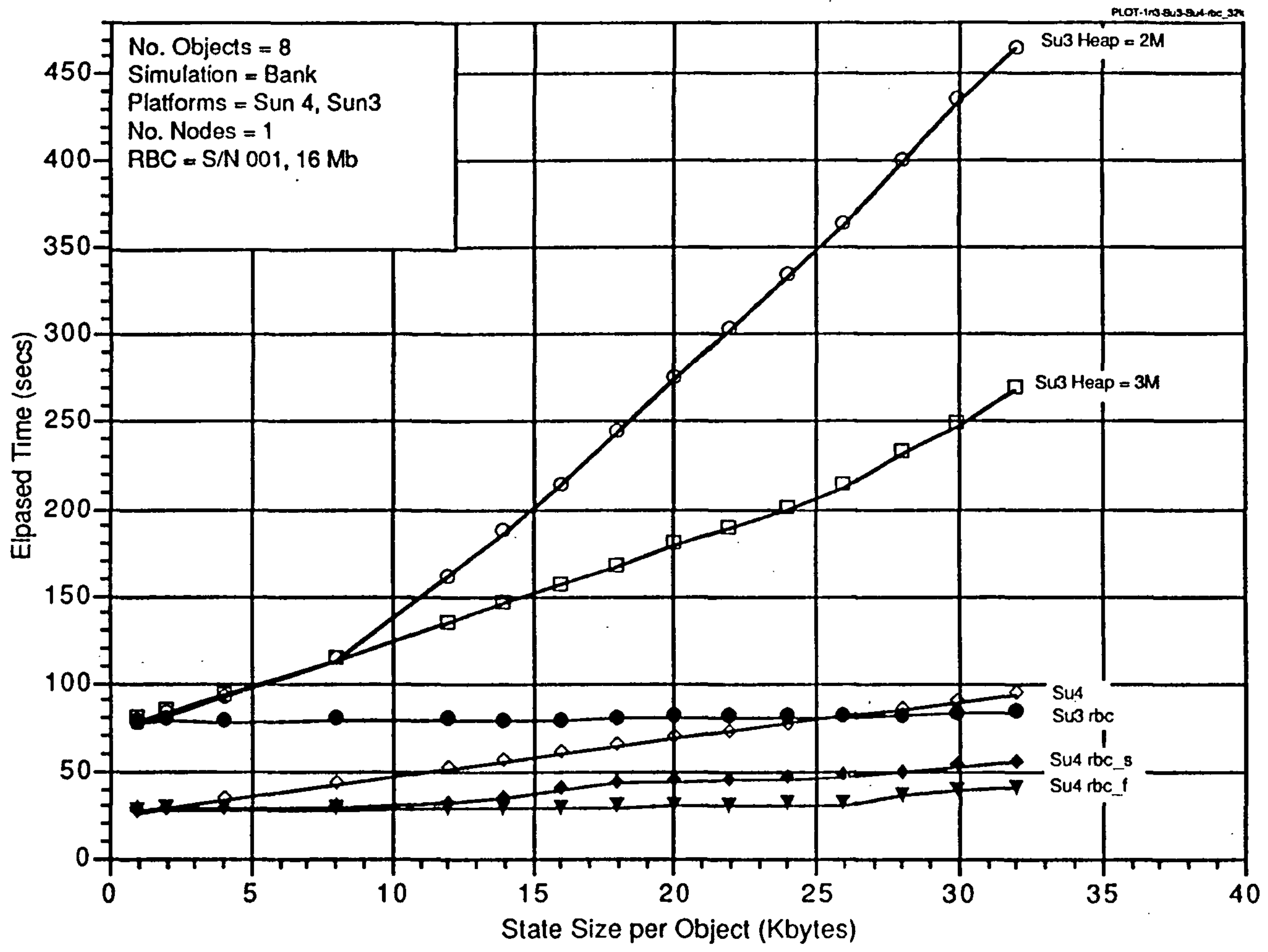

Figure 22, Consolidated Elapsed Times

7.0 RBC IMPROVEMENTS

As discussed in various sections of this report the prototype RBC modules built under this effort exhibited a number of performance anomalies during testing. Some of these have direct influence on the test results while others relate to RBC produceability. The major areas for improvement are:

(1) Module Cross-Talk

Due to the initial implementation of the RBC module as a wire wrapped rather than printed circuit board and the high speed circuits used, the modules have exhibited random cross-talk problems during some simulation runs. Additionally, during initial integration testing of RBC S/N 001 the majority of "bug fixes"were related to cross-talk rather than design errors (e.g., rerouting wires, moving I.C.'s, etc.). Build of S/N's 002 -004 allowed for a re-layout of IC placement resulting in much improved performance. The cross-talk effects have generally been manifested either as an incorrect number of events completed in the simulation, due to a missread of a state variable, or as a VME bus time out, caused by an aborted or spurious transfer attempt.

VME bus time outs are reported to the user on the terminal and therefore are easily detected. To determine if the correct number of events were run the user can either rerun the simulation a few times to verify its stability, or run the simulation without the RBC to obtain a benchmark.

The noise problems encountered with the first four modules coupled with the likely requirement to increase the module's speed will require subsequent versions of the RBC to be implemented as a printed circuit card rather than a wire wrap module.

(3) State Read/Write Access Times

As noted in section 5, the RBC access time for state reads and writes is slower than that for both the Sun 3 and Sun 4. Because state saving is typically the dominant overhead effect, especially for large states, the slower access time is not of concern for most applications on the Sun 3. However Sun 4 access times are substantially faster than both the RBC and the Sun 3. The major component of the slower RBC access time is the MRV extraction process for reads and the MRV extraction and copy forward in writes. We believe that subsequent versions of the RBC could allow for inclusion of the MRV extraction algorithm in a single integrated chip, which operates at substantially higher throughput rates than the current five chip implementation. Additionally, RBC architecture improvements should allow for the copy forward algorithm to complete after temporarily latching incoming node data, and thus not hold up the node.

While faster versions of the RBC will reduce the difference between the memory reference times it is clear that an add-on memory device cannot perform at speed with a CPU's main memory, especially when caching is involved. This will always result in the RBC spending more time in its object and have to"make it up" with state copying savings. The intent of speeding up the RBC is to reduce the difference and thus be as efficient as possible in capturing the state copying time.

(3) VME Bus Time out Variance

The protocol established between the node and the RBC for implementation of background and foreground tasks generally attempted to perform as many tasks as possible in foreground to simplify the background to foreground transitions. The protocol relies on a fixed value for the VME bus Time out from system to system and the board relies on that time for its foreground processing. When moving from the Sun 3 to the Sun 4 we found that there is considerable variance in this time out value, resulting in the potential for a time out to occur while processing node commands.

Subsequent versions of the RBC should provide the capability to either change the time out value on the module from system to system or restructure the firmware to only handle tasks in foreground which can be handled in less than the minimum time out value. The restructuring could be considered a sub component of the modifications required to speed up the firmware in item (4) below.

(4) DSP Code Implementation

The DSP firmware coded under this effort was implemented in the C programming language due to C's relative ease of programming and maintenance, when compared to assembly language. Use of C allowed for rapid prototyping, at the expense of routines which are slower than originally forecasted and which do not always take advantage of the DSP 56000's unique features. As shown in the performance results of Section 6, the slower execution does not become readily apparent until running on the Sun 4, with the appearance of the mark rollovers and corresponding impact on elapsed time. However, the potential for degraded performance is partially dependant on state size and is therefore present in both configurations, at larger state sizes. Additionally, the slower firmware also is a component in the read/write response times discussed in item (2) above, and in the VME bus time outs discussed in item (3) above.

Future versions of the DSP firmware could be rewritten in assembly for faster execution.

8.0 CONCLUSIONS AND ESTIMATE OF TECHNICAL FEASIBILITY

Based on the work performed in Phase II effort and the results and observations described in previous sections of this report, each of the objectives established for the program have been met.

From the performance results presented in Section 6, the following conclusions can be drawn:

- The RBC has been shown to capture essentially all of the potential savings, based on state copy times, available in the applications used for this effort. Further, in applications where state copy times do not account for a large percentage of the overall elapsed time, RBC speed up is limited to modest gains as predicted.

- An additional component of Time Warp overhead which is related to state block manipulations, but is not part of state copying, has been identified and shown to exist in the simulations used in this effort. This overhead associated with state memory management time has also been shown to be eliminated for systems utilizing the RBC.

- Speed up values measured and estimated for the Sun 3 range from 1 to 22 depending on the application, the state size and the granularity of the simulation. Sun 4 values range from less than 1 to 7 again depending on simulation parameters.
- The RBC modules built under this effort have demonstrated the RBC's ability to be easily configured with varying memory configurations and to free memory in the node's local memory space.
- The RBC has been shown to interface with more than one processor type with essentially no modifications required.

The production, integration and test of the four RBC modules used in this effort has demonstrated the feasibility of the proposed concept in deliverable form. The modifications proposed for subsequent versions of the RBC will provide enhanced performance and increased reliability over that of the prototype units. Potential platforms and applications for production versions of the RBC is dependant on the future use of Time Warp in commercial and government applications.

9.0 REFERENCES

[B*88] B.Beckman et al. Distributed Simulation and Time Warp: Part 1: Design of Colliding Pucks. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):56-60,July 1988.

[Bel92] Dr. S. Bellenot, State Skipping Performance with the Time Warp Operating System (performed at NASA-JPL), Math Dept., Florida State University, Tallahassee, FL 32306, bellenot@math.fsu.edu

[CM,79]K.M. Chandy and J. Misra, Distributed Simulation, *IEEE Transactions on Software Engineering*, SE-5(5):440-452, September 1979.

[Fuj87]R.M.Fujimoto, et al, The Roll Back Chip: Hardware Support for Distributed Simulation using Time Warp, *Technical Report AD-A187823*, University of Utah, Computer Science Department, October 1987.

[Fuj88]R.M.Fujimoto, et al, Design and Performance of Special Purpose Hardware for Time Warp, 15th Annual International Symposium of Computer Architecture, June 1988.

[Fuj89] Dr. R. Fujimoto, TWOS Measurements for Rollback Hardware, Technical Report (performed NASA- JPL), August 9, 1989, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332

[Jef82]D.Jefferson, et al, Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control, *A RAND Note*, Report No. AD-A129431, December 1982.

[Jef85]D.R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404-425, July 1985.

[Jef87] D. Jefferson et al, Distributed Simulation and the Time Warp Operating System. Technical Report, Computer Science Dept., UCLA, August 1987.

[LCUW88]G.Lomow, J.Cleary, B.Unger, and D. West. A Performance Study of Time Warp. Proceedings of the SCS Multiconference on Distributed Simulation, 19(3):50-55, July 1988.

[Mar88] J.Marti. RISE: The RAND Integrated Simulation Environment. *Proceedings of the SCS Multiconference on Distributed Simulation,* 19(3), July 1988.

[PWM79] J.K. Peacock, J.W.Wong, and E.G. Manning, Distributed Simulation Using A Network of Processors. *Computer Networks*, 3(1):44-56, February 1979.

[RBF88] M. Robb, C. Buzzell, R. Fujimoto, Modular VME Rollback Hardware for Time Warp, *1990 Society for Computer Simulation Western Multi-Conference, Distributed Simulation,* 1988

[RB89] M. Robb, C. Buzzell, IPT Report - Phase I SBIR Final, VME Rollback Hardware for Time Warp, Doc No. 18-210006, Aug 1989.

[RM87] D.A.Reed and R.M. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing Computer Science*, MIT Press, 1987.

[RMM88] D.A.Reed, A.D. Malony, and B.D.McCredie. Parallel Discrete Event Simulation Using Shared Memory. IEEE Transactions on Software Engineering, 14(4):541-553, April 1988.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle VME Rollback Hardware for Time Warp Multiprocessor Systems - Phase II Final Report		5. Report Date Oct 11, 1992
		6. Performing Organization Code
7. Author(s) Michael J. Robb and Calvin A. Buzzell		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address Integrated Parallel Technology Inc. 5994 W. Las Positas Blvd. Ste 209 Pleasanton CA. 94588		11. Contract or Grant No. NAS7-1102
12. Sponsoring Agency Name and Address NASA Washington DC 20546-0001	NASA Resident Office -JPL 4800 Oak Grove Dr. Pasadena CA. 91109	13. Type of Report and Period Covered Final
		14. Sponsoring Agency Code

15. Supplementary Notes

none

16. Abstract

The purpose of the research effort is to develop and demonstrate innovative hardware to implement specific rollback and timing functions required for efficient queue management and precision timekeeping in multiprocessor discrete event simulations. The previously completed Phase I effort demonstrated the technical feasibility of building hardware modules (referred to as RBC or Rollback Modules) which eliminate the state saving overhead of the Time Warp paradigm used in distributed simulations on multiprocessor systems. The current Phase II effort will build multiple pre-production Rollback hardware modules integrated with a network of Sun workstations, and the integrated system will be tested by executing a Time Warp simulation. The rollback hardware will be designed to interface with the greatest number of multiprocessor systems possible. The authors believe that the Rollback hardware will provide for significant speedup of large scale discrete event simulation problems and allow multiprocessors using Time Warp to dramatically increase performance.

This is the Final report for the Phase II effrot.

17. Key Words (Suggested by Author(s)) Time Warp Discrete Event Simulation Multiprocessors Combat Simulation Asynchronous Simulation	18. Distribution Statement Approved for Public Release; Unclassified - Unlimited Mathematical and Computer Sciences

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages	22. Price

NASA FORM 1626 OCT 86