Parallel Shared-Memory Simulator Performance for Large ATM Networks

BRIAN UNGER and ZHONGE XIAO
University of Calgary
JOHN CLEARY and JYA-JANG TSAI
University of Waikato
and
CAREY WILLIAMSON
University of Saskatchewan

A performance comparison between an optimistic and a conservative parallel simulation kernel is presented. Performance of the parallel kernels is also compared to a central-event-list sequential kernel. A spectrum of ATM network and traffic scenarios representative of those used by ATM networking researchers are used for the comparison. Experiments are conducted with a cell-level ATM network simulator and an 18-processor SGI PowerChallenge shared-memory multiprocessor.

The results show the performance advantages of parallel simulation over sequential simulation for ATM networks. Speedups of 4-5 relative to a fast sequential kernel are achieved on 16 processors for several large irregular ATM benchmark scenarios and the optimistic kernel achieves 2 to 5 times speedup on all 7 benchmarks. However, the relative performance of the two parallel simulation kernels is dependent on the size of the ATM network, the number of traffic sources, and the traffic source types used in the simulation. For some benchmarks the best single point performance is provided by the conservative kernel even on a single processor. Unfortunately, the conservative kernel performance is susceptible to small changes in the modeling code and is outperformed by the optimistic kernel on 5 of the 7 benchmarks. The optimistic parallel simulation kernel thus provides more robust performance, but its speedup is limited by the overheads of its implementation, which make it approximately half the speed of the sequential kernel on one processor.

These performance results represent the first comparative analysis of parallel simulation for a spectrum of realistic, irregular, low-granularity, communication network models.

Financial support for this research was provided by an NSERC Collaborative Research and Development grant (CRD 183839), and by our many industrial sponsors: Netera Alliance, Newbridge Networks, New Zealand Telecom, Nortel, Siemens, Silicon Graphics, Stentor, and Telus. This support, and the ongoing interactions with our industrial sponsors, are greatly appreciated.
Authors' addresses: B. Unger and Z. Xiao, Department of Computer Science, University of Calgary, Calgary, T2N 1N4, Canada; J. Cleary and J.-J. Tsai, University of Waikato, Waikato, New Zealand; C. Williamson, University of Saskatchewan, Saskatchewan, Canada.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*Single-instruction-stream, multiple-data-stream processors* (SIMD); C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Asynchronous Transfer Mode* (ATM); I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Parallel*

General Terms: Experimentation, Performance

Additional Key Words and Phrases: ATM network modeling, conservative synchronization, optimistic synchronization, parallel discrete event simulation, time warp

1. INTRODUCTION

There is a long history of attempts to speed up simulations through parallel execution. Two main families of synchronization algorithms have been proposed. The first "conservative" algorithm is described in Chandy and Misra [1979]. The optimistic TimeWarp algorithm [Jefferson 1985] was developed in the early 1980s. Since then there have been many published variations and implementations [Avril and Tropper 1995; Bagrodia and Liao 1994; Blanchard et al. 1994; Cai et al. 1997; Fujimoto 1989; Martine 1995; Nicol and Heidelberger 1995; Steinman 1992; Su and Seitz 1989]. This work has clearly demonstrated that both algorithms are able to achieve significant speedups for small artificial problems. The question of whether they can be routinely used as a mechanism for speeding up real applications remains open. We are aware of only one application, an air traffic control simulation, where an optimistic parallel simulation system is in regular production use [Wieland 1995].

A number of researchers have reported significant speedups for a range of applications. However, most, if not all, of these applications are either small models, models that have very regular structure, or models created specifically for use in parallel simulation performance studies. Examples include TimeWarp-based simulators used for mobile personal communications systems (PCS) networks [Carothers et al. 1995]; conservative parallel simulators for wireless networks [Meyer and Bagrodia 1998; Zeng et al. 1998]; and ATM networks [Nicol 1996; Nicol and Heidelberger 1995]; and VLSI circuit simulation [Chen and Bagrodia 1998]. Further, there are many examples where conservative and optimistic approaches failed to achieve any significant speedup.

There has been previous work on conservative parallel simulation of large ATM networks [Pham et al. 1998], but primarily at the call-level (e.g., routing), as opposed to the fine-grain cell-level models.

To the best of our knowledge, this article is the first to compare a conservative to an optimistic simulator for a set of detailed models of large broadband ATM (asynchronous transfer mode) networks [Minzer 1989], developed and used to support experimental research in ATM network protocols. These detailed benchmark models characterize cell-level behavior, including all of the major ATM standard communication protocols for two ATM networks: the 11 switch Western Canadian regional network

(Wnet), and the 54 switch Canadian National Test Network (NTN). 3 Wnet and 4 NTN traffic load scenarios are used to define 7 large, irregular, benchmark models. The 7 benchmarks were used by researchers to study call-admission control and congestion-control algorithms [Wang et al. 1999; Zoranovic and Williamson 1999] and estimates of the effective bandwidth of aggregate Ethernet, video, and Web ATM traffic [Gurski and Williamson 1996; Patel and Williamson 1997].

Simulation is a vital tool in the design and analysis of high-speed ATM networks. Due to the high costs of installing and operating an ATM network, and the rapid pace of changes in ATM technology and standards, networking researchers and telecommunications engineers rely on simulation to evaluate, for example, ATM switch design, the effectiveness of ATM traffic-control policies, and the scalability of traffic-management approaches to large heterogeneous ATM networks.

Simulations in the ATM context typically have two key requirements: the need for detailed cell-level simulation, so that fine-grain performance differences between traffic control policies or ATM switch architectures can be understood; and the need for a large number of simulation events (e.g., billions to trillions of ATM cells) in order to assess quality of service at an appropriate level (e.g., cell loss ratios in the range from 10^{-6} to 10^{-9}). These two requirements combine to challenge current uniprocessor computing platforms, typically producing multiday rather than same-day turnaround time for most commercial ATM network simulators on medium to large ATM network scenarios. Parallel simulation is one approach for addressing this problem. A parallel simulator, if properly designed, can exploit the inherent parallelism in a large ATM network scenario, offering significant speedup.

ATM simulations pose a significant set of challenges for parallel simulators. On the one hand, they are large problems with significant amounts of parallelism, which makes the task easier. On the other hand, the workload is irregular, varying over both time and the objects in the simulation. This makes the problem of load-balancing the simulation across multiple processors difficult. The granularity (the time to execute one event in the simulation) is very low as well. On the SGI platform, the granularity averages about 10 microseconds [Arlitt et al. 1995]. The average per-event overheads of any simulator (including complex parallel ones) cannot be much larger than this if good performance is to be achieved, i.e., if, given n processors, the speedup achieved is a significant fraction of n.

In this article, we evaluate and compare the performance of two different shared-memory parallel simulation kernels for ATM network simulation, one optimistic and one conservative. The optimistic parallel simulation kernel, WarpKit [Xiao and Unger 1995b], is a shared-memory TimeWarp System [Fujimoto 1989; 1990] with its associated mechanisms for state-saving, rollback, and fossil collection. The conservative parallel simulator [Cleary and Tsai 1996] uses lookahead techniques to ensure that events are executed locally (at a single logical process) in strict time order. Thus, it

avoids the overheads and complexities of rollback, albeit at the cost of reduced parallelism.

Both simulation kernels are evaluated using the cell-level ATM traffic and network (ATM-TN) simulator [Unger et al. 1995], for two ATM network topologies (Wnet and the NTN) with a range of network traffic loads. The experiments were conducted on a commercially available shared-memory multiprocessor, namely an 18-processor SGI Power Challenge.

The results of our study demonstrate the performance benefits achievable with parallel simulation for ATM network simulation. For example, speedups of 2-3 over sequential simulation are possible for all network scenarios studied, and speedups of 4-6 are possible on some scenarios. The performance results for the two parallel simulators are found to be sensitive to the size and structure (i.e., number and type of traffic sources) of the ATM networks simulated. In general, the optimistic parallel simulator provides more robust performance across the range of ATM network scenarios studied, while the conservative simulator gives the best performance on selected scenarios.

The remainder of this article is organized as follows. Section 2 provides some background on the sequential and parallel simulation kernels evaluated in this study. Section 3 describes the ATM-TN simulator used in the study, and Section 4 the ATM benchmark scenarios. Section 5 presents the results of our study, including the sequential and parallel performance results for each of the ATM benchmark scenarios considered. Finally, Section 6 summarizes our conclusions.

2. SIMULATION KERNELS

SimKit is the application programmer's interface (API) that we use for writing simulation applications [Gomes et al. 1995]. SimKit is an object-oriented library for discrete-event simulation implemented in C++. A model is described in terms of objects called *logical processes* (LPs). Each LP can communicate with other LPs via messages only (that is, they can share no writable state). Communication is via time-stamped messages that execute an *event* at the *receive time* of the message. The code for a model uses an event-based rather than a process-based view; that is, it is impossible to receive events at inner points in the model code, but only at the point where the execution of an event is initiated.

SimKit defines three base classes: *sk_simulation*, from which all the controlling information for a simulation is instantiated (e.g., initialization process, warm-up period, simulation end time); *sk_lp*, from which the logical processes that carry out simulation activities are derived; and *sk_event*, from which all simulation events (i.e., messages) are derived. Each logical process (LP) executes its received events in time-stamp order, updating local state information, and possibly generating more events.

As noted earlier, due to the small amount of user computation in the ATM-TN application, it is important to keep kernel overheads low. SimKit itself includes a number of optimizations designed to keep these overheads

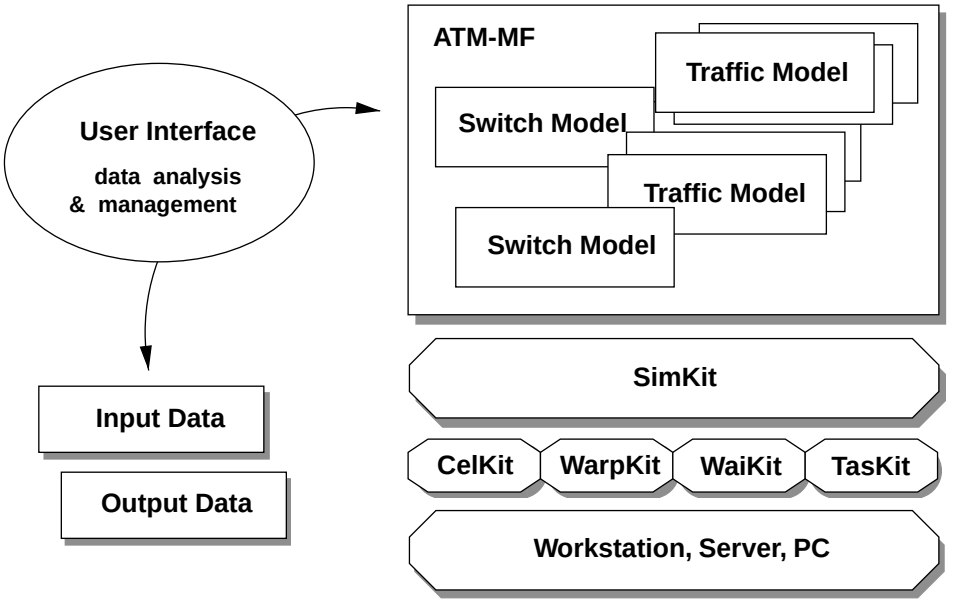

Fig. 1. The structural relationships between the simulation application (the ATM-TN and its components: the ATM-MF, i.e., the modeling framework, and the traffic and switch models), the simulation API (SimKit), the simulation kernels (CelKit, WarpKit, WaiKit and TasKit), and the simulation platforms (e.g., SUN SPARC, SGI PowerChallenge).

low. The main ones are fixed-size buffers for events, and extensive use (where possible) of pointer-passing, rather than data-copying, when forwarding messages between LPs.

SimKit applications, such as the ATM-TN, can be executed on a variety of simulation kernels (see Figure 1). To date, we have run ATM-TN on four kernels: *CelKit*, a central-event-list-based sequential simulator; *WarpKit*, an optimistic parallel simulation kernel; *WaiKit*, a conservative parallel simulation kernel; and *TasKit*, a new optimized conservative kernel. This article compares the performance of *WarpKit* with *WaiKit* and the sequential kernel. The next three sections provide further background on each of these three kernels. A subsequent article will present comparative performance results for the TasKit kernel, which is still under development.

2.1 Central-Event-List-Based Sequential Simulator (CelKit)

CelKit is a sequential event-list-based simulator. As each event is generated it is placed into a time-ordered event list. CelKit has an optimized event list that uses a splay tree [McCormack and Sargent 1981], which is able to efficiently handle both a large number of events and irregular and varying time distributions within the event list. Insertion into, and sequential extraction from, the event list are the dominant overheads in CelKit. There are event-list algorithms, e.g., the Calendar Queue [Brown 1988], which can be more efficient than a splay tree even for large event lists. Our experience, however, is that the Calendar Queue is not robust, as it can

give poor performance for some distributions of time-stamps. As a consequence, we do not use it in the current system.

2.2 Parallel Execution

Any parallel execution of a simulation has a number of problems with which to contend. The main one is that parallel algorithms tend to be more complex, which implies higher overheads. The sections below, on individual parallel kernels, outline techniques for minimizing these extra overheads.

Another crucial problem is partitioning or load-balancing, referring to how the LPs are mapped to the physical processing elements (PEs) of the shared-memory multiprocessor. Clearly, the goals are to balance the computation load among the PEs, and to minimize the communication requirements between PEs (i.e., messages between LPs that are in different partitions). However, automatic partitioning and dynamic load-balancing are still open research topics. We currently rely on a static (i.e., manual) partitioning of the parallel simulation across PEs, based on application-level knowledge (see Section 4.3). In the performance results reported in this article, both WaiKit and WarpKit use the same static partitions.

2.3 WarpKit

WarpKit is a parallel simulation kernel developed at the University of Calgary [Xiao and Unger 1995b], derived from the 1993 version of GTW, a TimeWarp system developed for shared-memory multiprocessors at Georgia Tech [Das and Fujimoto 1993; Das et al. 1994]. The initial version of WarpKit has been extensively revised and extended to achieve robust performance for low-granularity applications [Xiao and Unger 1995b]. Several of these optimizations are outlined in this section.

TimeWarp provides an environment for optimistic parallel execution [Jefferson 1985; Jefferson et al. 1987]. That is, each LP is allowed to progress forward in time at its own pace, based on the messages it receives, without requiring explicit synchronization with other LPs. However, if an LP receives a message with a time-stamp t from the past (compared to the LP's local time), then the LP must perform a *rollback*, undoing the possibly incorrect steps that it has taken since time t, canceling the possibly incorrect messages that it has sent to other LPs since time t, and resuming forward progress from time t. Clearly, being able to do rollback implies the need for *state-saving* on the forward execution path. This state must be maintained until the *global virtual time (GVT)* of the simulation, determined by the minimum time of all LPs and events, advances far enough for events to be *committed*, at which time old state information can be discarded and its memory space reclaimed, a process called *fossil collection*.

State-saving is thus one of the attendant overheads in optimistic parallel simulation; minimizing its overhead is essential for good performance. An incremental state-saving (ISS) mechanism, which saves state variables only when they are modified, is built into SimKit on top of WarpKit [Gomes

1996]. This keeps the state-savings cost low, since the percentage of state changes for most of the events in ATM-TN is low.

Another source of kernel overhead is the global control mechanisms, including GVT computation, fossil collection, and buffer management. WarpKit employs a two-tiered distributed buffer management scheme and a fast asynchronous GVT algorithm [Xiao et al. 1995]. Each processor uses its local buffer pool for events. A central pool is used to adjust the local pool dynamically, in such a way that the number of buffers (both in the local pool and those occupied by events) is kept roughly constant. The GVT algorithm allows any PE to initiate GVT calculation and fossil collection without direct collaboration with other PEs. As a result of this asynchronous approach, the PE executing on the critical path will be able to progress with less interference, and the fast-running PEs will be engaged in more frequent GVT calculations. This also tends to restrict optimism. Furthermore, the GVT calculated by the fast PEs can be utilized by slow PEs, effectively reducing the overhead on the critical path. The global control overhead of WarpKit is now the lowest of all overheads. This is in contrast to the original version of WarpKit, in which global control is the performance bottleneck [Xiao and Unger 1995b].

Furthermore, minimizing the number and size of rollbacks on the critical computation path is also crucial. An effective way of reducing excessive rollback is using a small number of buffers for each PE, as mentioned above. There are also other mechanisms to constrain rollbacks, which are more specific to the ATM-TN simulator.

Event flow control is one such mechanism [Xiao and Unger 1995a]. It places a constraint on how far ahead in time one LP can be compared to GVT. Event flow control is particularly important for two classes of LPs: (1) those that produce but never consume simulation events (e.g., the video traffic sources described in Section 3.1), since they may "flood" the system with many future events, exhausting the local buffer pool and causing many GVT computations; and (2) those LPs that consume but never produce events, since they may "dry out" the system, increasing the chance of rollback. This problem can be solved most efficiently at the ATM-TN level, not in the underlying kernel. The solution is to strike a balance between the number of events generated by the source and the number of events consumed by the sink. In principle, this is like sliding window flow control in network communication protocols, but at the level of simulation events. The benefits of this mechanism are most evident on small network scenarios with highly asymmetric traffic flows. The need for the mechanism on large network models with well-balanced loads is less obvious.

Rollbacks are also a problem for LPs that have self-initiated events widely spaced in time (e.g., reporting end-of-simulation statistics, reporting periodic statistics, communication protocol time-outs, and retransmissions). These events tend to be selected repeatedly and executed prematurely, causing a large number of unnecessary rollbacks. The native Time-Warp mechanism is incapable of dealing with such situations. In our implementation, we solve this problem by using *wait queues*: a separate

"waiting area" at each processor for "far off" future events that precludes these events from early processing by LPs. These waiting messages are moved to the PE's event queue for processing only when the simulation time draws close enough.

There are many other implementation optimizations in WarpKit to keep kernel overhead low. In summary, WarpKit provides a fast and reliable parallel simulation executive, with several optimizations specifically designed to support fast parallel simulation of ATM networks.

2.4 WaiKit

The WaiKit parallel simulation kernel was developed at the University of Waikato in New Zealand [Cleary and Tsai 1996]. Since WaiKit is a conservative kernel, it differs significantly from WarpKit. In particular, LPs always execute events in nondecreasing time-stamp order; no rollbacks are possible.

Clearly, conservative parallel simulation requires stronger coordination and synchronization among the LPs. This coordination usually results in reduced parallelism. In the most extreme case, all LPs must proceed in lock step at the pace of the slowest LP, producing a performance that can be slower than sequential execution. However, such extreme synchronization requirements are rarely needed in practice; more generous forward progress is possible, particularly if application-level knowledge is available. Furthermore, WaiKit does not have the complexity and overheads required in WarpKit for state-saving and rollback.

WaiKit exploits the use of *safe-times* to determine bounds on the forward execution permitted by each LP in the simulation [Cleary and Tsai 1996]. The safe-time for an LP is a lower bound on the earliest possible time in the future at which a new message could arrive. In order to compute such safe-times effectively, WaiKit needs to know about (logical) communication *channels* between LPs. All messages must be sent via channels, and, for each possible communication path between LPs, a channel must be constructed before the simulation begins. For ATM-TN, this is effective because the set of used channels is sparse, compared with all possible communication paths between LPs. To remove the burden of explicitly constructing channels from the modeler, a presimulation pass is done by WaiKit in order to construct the set of channels allowed. This construction depends on knowledge of the ATM-TN application, and has to be rewritten if ATM-TN is modified or if a new application is written. This approach has a small runtime overhead because the source code references a destination LP only, not an actual channel. Thus, to find the channel, given the current LP and the destination LP, a dynamic lookup is necessary. If the existence of channels could be seen by the application programmer, then this overhead would be removed and construction of the presimulation channel would not be necessary. To preserve compatibility with existing code and the other kernels, this was not done.

The safe-time for an LP is computed by taking the minimum of the safe-times for all channels that arrive at the LP. The safe-time for a

channel is the safe-time of its source LP plus a *lookahead* time. This is the minimum time-stamp increment that any message sent down the channel can have. In ATM-TN, it is computed statically before the simulation begins, from the propagation delay specified for a signal to traverse a (simulated) physical link.

For WaiKit, the basic execution cycle is to first select an LP and then execute all messages for that LP with receive times less than the LP's current safe-time. This causes the safe times for all the LP's outgoing channels to be updated. This is different from both CelKit and WarpKit, where activity is scheduled on the basis of individual events, rather than LPs. Provided the average number of events to be executed each time an LP is selected can be kept high, then this mechanism has potentially very low overhead. The average number of events depends upon the connectivity of the ATM network and traffic flows. More specifically, it is determined by the shortest potential loop in the channel graph. The expected number of messages around such a loop limits the number of messages to be executed. Thus, if the lookaheads are small or there are few messages, then LPs may be scheduled more than once for each event processed, causing poor performance. This aspect of conservative algorithms can make them sensitive to small changes in network structure, either as the result of the modeling strategy or due to physical topology in a particular ATM network. All that is needed for overall poor performance is one loop with a total lookahead close to zero, and execution time will be dominated by executing the LPs in that loop. In some networks, there were no loops at all, so the average was potentially infinite. WaiKit performed very well in these cases (see Section 5).

WaiKit uses another simple scheduling strategy to keep overheads low for LPs. There is a single static list of all the LPs partitioned onto a particular PE. This list is repeatedly scanned and each LP in turn is scheduled. This gives very low overheads (little more than dereferencing a pointer for each LP) and performs well, as long as each LP has something to do on each cycle. However, if there are some LPs with poor lookaheads, then they will make repeated small advances in time and the entire list will be rescanned each time.

Other strategies for keeping overheads low include carefully tuned code for merging the incoming channels for each LP; a lock-free method for transferring messages on channels between PEs; and the use of fixed arrays for the buffers in a channel to avoid the overheads due to allocating and deallocating buffer space.

In summary, WaiKit provides a simple conservative parallel simulator designed to have low overhead, and tuned to the characteristics of ATM cell-level simulations.

3. ATM-TN SIMULATOR

The asynchronous transfer mode traffic and network (ATM-TN) simulator is a cell-level ATM network simulator developed collaboratively as part of

the TeleSim project [Unger et al. 1995; Williamson et al. 1998]. With ATM-TN a user can define an arbitrary network topology, various ATM switch types and links for instantiation within that topology, and a set of traffic flows on that topology. The ATM-TN is intended to support network design, configuration, and performance analysis.

There are three main components in ATM-TN: traffic models, switch and network models, and a modeling framework. The traffic models define the behaviors of the traffic sources, which generate patterns of simulated ATM cells according to the specified traffic types and parameters, and traffic sinks, which consume incoming ATM cells. The switch and network models specify the behaviors (e.g., processing delays, traffic-control policies, cell-scheduling disciplines, buffer sizes, transmission speeds, and propagation delays) of the switches, ports, and links in the simulated ATM network, which take ATM cells as inputs and produce ATM cells as outputs. The modeling framework defines the interfaces to the switch and traffic models, as well as to the input, output, and statistics-reporting routines that are common to many of the switch and traffic models.

The following sections provide further details on the design and operation of the traffic and switch models.

3.1 Traffic Models

There are six different source traffic models in the ATM-TN simulator: a deterministic traffic model, a Bernoulli traffic model, an Ethernet LAN data traffic model [Chen et al. 1995], an MPEG/JPEG compressed video traffic model [Arlitt et al. 1995], a World Wide Web client traffic model [Arlitt and Williamson 1995], and a TCP/ATM traffic model [Gurski and Williamson 1996].

The first two traffic models, deterministic and Bernoulli, are simple traffic models used primarily for validating simulations. The deterministic traffic model generates a stream of ATM cells with constant (i.e., deterministic) spacing between cells. This model is useful for generating constant bit rate (CBR) traffic flows. The Bernoulli model is the discrete-time analog of a Poisson traffic source. That is, the Bernoulli traffic model generates a stream of ATM cells in which the interarrival times between cells are geometrically distributed and independent. This model is useful for generating a variable bit rate (VBR) background traffic load in an ATM network. In both of these traffic source models, the simulation user specifies the mean bit rate for the traffic source, as well as the number of cells to be generated. These parameters are specified separately for the two directions of traffic flow, which operate independently.

The Ethernet traffic model generates variable-size data bursts typical of local area network (LAN) traffic. In particular, the burst sizes are highly correlated in time, to reflect both the short-range and long-range dependence observed in real Ethernet LAN traffic [Leland et al. 1994]. Simulation users specify a utilization parameter U that controls the mean bit rate of the traffic source, as well as a Hurst parameter H that controls the

degree of correlation in the traffic bursts. The Ethernet traffic model is open loop; that is, it produces an infinite duration unidirectional traffic flow. The source produces a sequence of closely spaced ATM cells for each data burst, and the sink simply consumes incoming cells.

The video traffic model produces a VBR traffic stream representative of the JPEG and MPEG video compression standards [LeGall 1991; Wallace 1991]. Video traffic has a periodic structure, determined by its frame rate (e.g., 30 frames per second). Furthermore, the size (in bytes) of each frame is highly correlated with recent frames of the same type [Garrett and Willinger 1994], as determined by the encoding algorithm used (e.g., an intraframe coding algorithm, such as JPEG, or an interframe coding algorithm, such as MPEG). These aspects are all captured in the video traffic model [Melamed 1992]. Simulation users specify the frame rate and the encoding pattern to be used for the video stream, as well as a scaling parameter to control the mean bit rate of the flow. Again, the video traffic model is open-loop: it produces an infinite duration unidirectional flow of ATM cells from source to sink.

The World Wide Web traffic model represents the behavior of Web users. Each such user generates sporadic small requests to Web servers, which then return a simulated Web document, with the size of the document drawn randomly from an arbitrary distribution function (in the benchmarks the Erlang distribution is used) [Arlitt and Williamson 1997; Crovella and Bestavros 1996; Paxson and Floyd 1994]. A random hyperexponential think time occurs before the next request, which might go to the same server or to a different one. Simulation users specify the mean and standard deviations of the distributions used for request sizes, response sizes, and think times. Note that the Web traffic model is closed-loop. That is, the time of the next outgoing request depends on the time at which the previous response is received (as well as the randomized think time, of course). The data flow is bidirectional, though highly asymmetric. Thus this traffic model has a very different behavior than the Ethernet and video traffic models do, particularly in terms of parallel execution.

The final traffic model in ATM-TN is a detailed model of the transmission control protocol (TCP) over ATM. TCP is the cornerstone protocol of the Internet, providing reliable data transfer for many network applications, including file transfer, electronic mail, and the World Wide Web [Tanenbaum 1996]. Our TCP model captures all behaviors of TCP, including sequence numbers, acknowledgments, sliding window flow control, slow start and congestion avoidance, time-outs, and retransmission [Stevens 1993]. Simulation users specify TCP configuration parameters, such as send and receive socket buffer sizes, maximum segment size, maximum flow-control window size, protocol timeout values, and the number of data bytes to transmit in each direction. The model can be used for either unidirectional or bidirectional data transfers. In either case, the traffic source transmits the simulated user data according to the TCP protocol, with the traffic sink generating acknowledgments for received data (piggybacking the acknowledgments on outgoing data packets, if any). The TCP

model is thus a closed-loop bidirectional traffic model. There is also detailed modeling of AAL-5, an ATM adaptation layer protocol used for translating between higher-layer protocol data units, such as TCP/IP packets, and the ATM cells used for transmission across an ATM network.

Each of the foregoing traffic models is implemented in SimKit using two LPs: one for the source and one for the sink. Multiple instances of a traffic model are permitted in an ATM network simulation. Though each traffic model represents very different application-level behaviors, each source model produces as output simulated ATM cells for transmission across the links and switches in the simulated ATM network.

One other characteristic to note is that the four main traffic models (i.e., Ethernet, MPEG, Web, and TCP) conceptually generate higher-layer data units (e.g., bursts, frames, files, and packets) that become "bunches" of ATM cells (tens or hundreds) at the lowest layer simulated. This fact has some implications for how well different parallel simulation performance optimizations may work.

3.2 Switch Models

The switch models in ATM-TN are responsible for receiving ATM cells on input ports, performing necessary traffic control functions (e.g., cell scheduling, usage parameter control, selective cell discard [ATM Forum 1996]) and forwarding cells to their prespecified destination path via output ports. The switches use a simulated ATM signaling protocol for establishing and releasing state information for end-to-end user-level connections, which are called virtual channels (VCs).

There are four different switch types currently defined in ATM-TN: a generic output buffered switch, a shared-memory switch, a crossbar switch, and a multistage switch using a multistage interconnection network. Only the generic output buffered switch model is used in the experiments in this article.

Each switch model is vendor-independent, with configuration options available to "customize" the switch. The main configuration parameters are the number of ports on the switch, the port speeds (in bits per second), the size (in ATM cells) of the buffers at each output port, and the cell-scheduling mechanism. The latter is used to arbitrate access to the output port among the five ATM service class queues per output port, namely constant bit rate (CBR), real-time variable bit rate (VBR-RT), nonreal-time variable bit rate (VBR-NRT), available bit rate (ABR), and unspecified bit rate (UBR) [ATM Forum 1996]. Each service class queue operates in a First-In-First-Out (FIFO) fashion, and is shared by all user-level ATM connections in that service class; per-VC queuing is also supported in the latest version of ATM-TN. Exhaustive priority or round-robin scheduling is used among the service classes, with priorities in the order given above.

The switch models are implemented in SimKit, and use one LP for each input port, one LP for each output port, and two LPs (one for the signaling protocol and the other for the segmentation and reassembly (SAR) function)

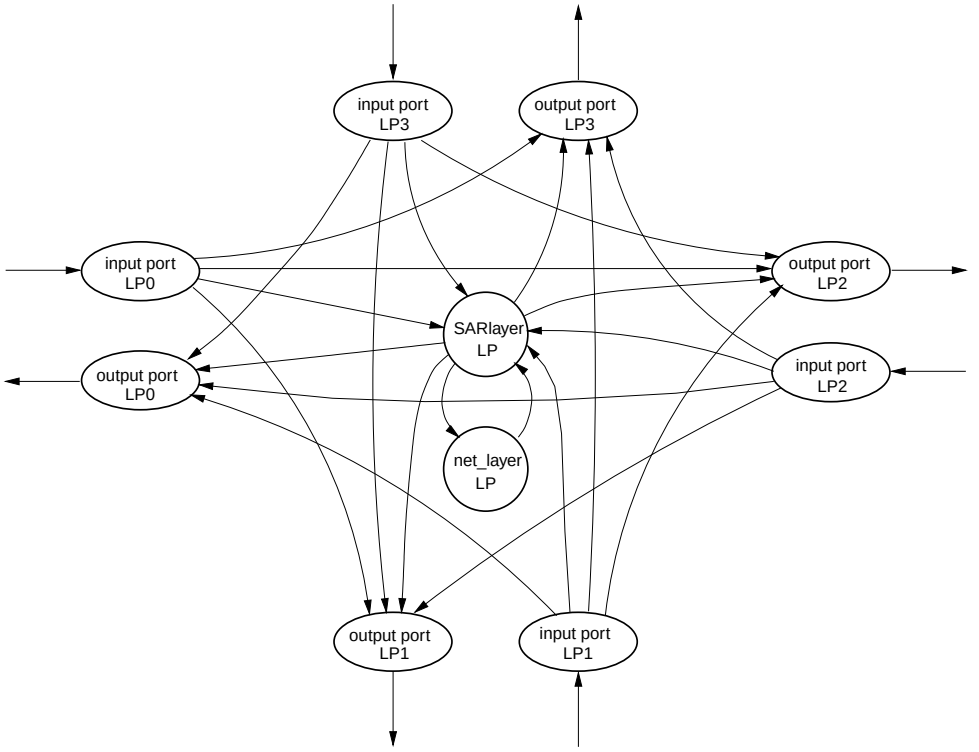

Fig. 2. Logical processes in a 4-port output-buffered switch model..

to represent the control processor, which handles all call signaling and traffic management functions. The model structure for a 4-port output buffered switch is illustrated in Figure 2. Also shown in the figure are all the possible message flows between various LPs in this switch model.

4. BENCHMARK SCENARIOS

This article presents and compares the performance of the WarpKit and WaiKit parallel simulation kernels against a sequential kernel CelKit, using a number of ATM network scenarios. The simulation results, in terms of simulated ATM network and traffic performance, are not the focus of this article. Rather, we are interested in which parallel simulation mechanisms can speed up the execution of simulations and to what extent. To accomplish this, we use a spectrum of irregular low-granularity ATM network benchmark scenarios.

We use two ATM network topologies as the basis for seven benchmark scenarios. The first of these, called Wnet, models the physical topology of a regional ATM testbed network in western Canada. The second, called the NTN, models the physical topology of a Canada-wide experimental National Test Network. For each network topology, we use a range of traffic

mixes to produce network traffic loads that range from light to medium to heavy.

While the traffic models have been validated against empirical traffic data so that each traffic model instance can be considered realistic, the actual deployment of traffic sources and sinks in the benchmarks are not necessarily realistic. In fact, the scenarios intentionally include congestion "hot spots" to explore how the parallel kernels perform under these conditions. For example, all the TCP sources are exhaustive, in the sense that each source tries to send (at the full link rate) as many packets as allowed based on TCP flow control rules. Also, zero host-level protocol processing time is assumed in TCP sources and sinks. While this is not realistic, it does stress the simulation kernel (as well as the network being modeled). The Ethernet sources generate background IP (over ATM) traffic. TCP sources/sinks, which implement the full TCP algorithms, introduce traffic loops that model the interactions between hosts and network.

4.1 Wnet Scenarios

The Wnet benchmark topology, shown in Figure 3 along with traffic loads for scenarios Wnet-1 and Wnet-2, characterizes a regional ATM testbed in western Canada. This network connects five universities in three provinces. The network consists of 11 ATM switches, spanning a geographic distance of approximately 800 kilometers. The backbone links in the network have a transmission capacity of 45 Megabits per second (Mbps); other links are 155 Mbps.

The traffic load in the Wnet simulation model is chosen to represent typical traffic in the real Wnet testbed: JPEG video for distance education between universities, TCP for file transfers between researchers and supercomputer centers, Web users, and Ethernet traffic as a generator of background traffic load between sites.

Three different levels of traffic load are defined on the Wnet model. Scenario Wnet-1 represents light load, with 12 traffic sources (10 Ethernet, 2 MPEG). Scenario Wnet-2 represents medium load, with 25 traffic sources (2 deterministic, 2 Bernoulli, 10 Ethernet, 8 MPEG, 3 TCP). Scenario Wnet-3 is the same as Wnet-2, but with the addition of 4 Web traffic sources.

The Wnet benchmark scenarios are summarized in Table I. Each of the three Wnet scenarios runs for 10 seconds of simulated time, producing approximately 20 million simulation events.

4.2 NTN Scenarios

The NTN network topology, shown in Figure 4, represents the Canada-wide ATM National Test Network (as of March 1996). The simulated network topology has 54 ATM switches, and spans a geographic distance of approximately 3000 kilometers. The backbone of the network is 45 Mbps, and provides connectivity between six regional ATM testbeds (one of which is Wnet).

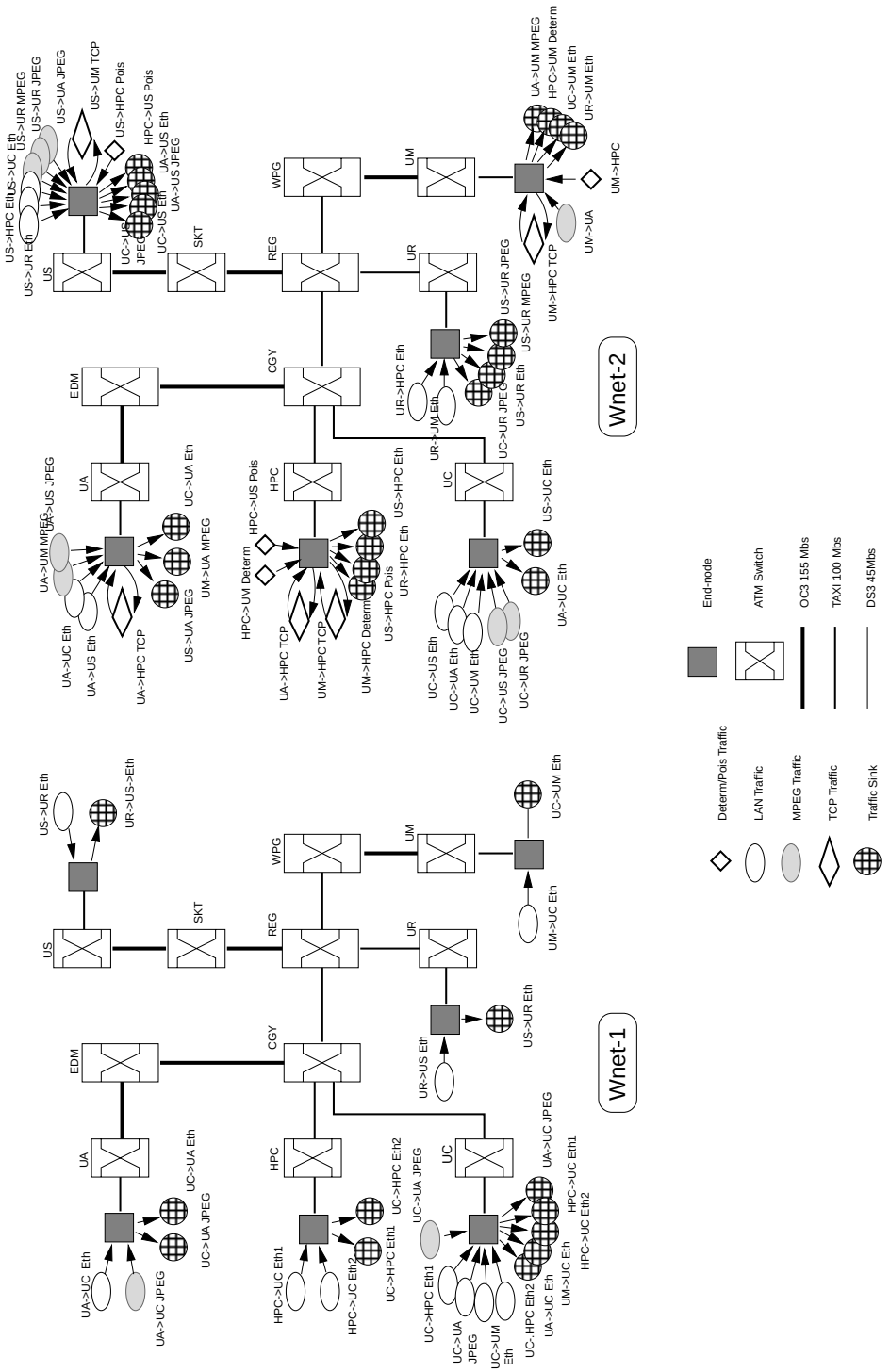

Fig. 3. Simulation models for the Wnet ATM network benchmark.

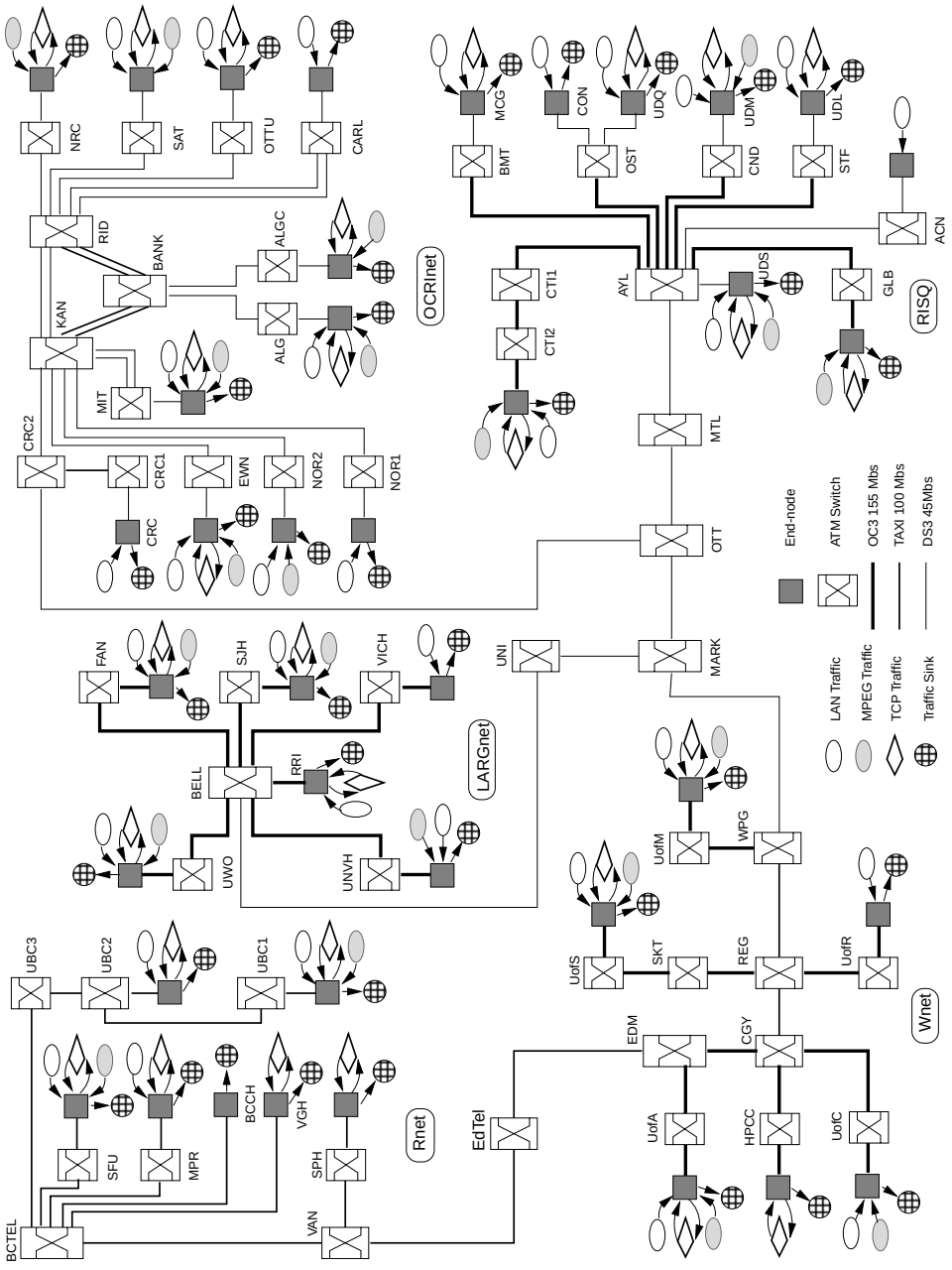

Fig. 4. Simulation model for the NTN ATM network benchmark.

The NTN is a good example of a network with a high delay-bandwidth product. That is, the combination of high transmission speeds (45-155 Mbps) and large end-to-end propagation delays (10-20 milliseconds) means that hundreds or thousands of ATM cells can be in transit at any time in the network links. These types of network scenarios are of particular

Table I. Wnet and NTN Benchmark Scenario Summary

benchmark scenario	number of traffic sources						ATM switch	num. LPs	events ($\times 10^6$)	time (sec)
	det	bern	ether	mpeg	web	tcp				
Wnet-1	0	0	10	2	0	0	11	181	22	10
Wnet-2	2	2	10	8	0	3	11	173	18	10
Wnet-3	2	2	10	8	4	3	11	173	18	10
NTN-0	0	0	53	30	0	16	54	869	48	5
NTN-1	0	0	62	269	0	24	54	1381	73	5
NTN-2	0	0	62	269	0	24	54	1381	132	5
NTN-3	0	0	62	269	0	24	54	1381	216	5

interest to ATM networking researchers in evaluating, for example, the effectiveness of feedback-based congestion control policies, the effectiveness of network traffic management on a national scale, and end-to-end quality of service guarantees in large internetworks. These networks are also challenging to simulate, due to their memory-space and computational requirements (caused, for example, by a large number of traffic sources, a large number of switches, large delay-bandwidth products, and very large event queue sizes). These are some of the motivating factors for including the NTN in our parallel simulation benchmarks.

The traffic loads we use on the NTN model are a hypothetical traffic mix for the national network: MPEG/JPEG video streams, TCP transfers, and highly bursty LAN data traffic. Approximately 75% of the traffic sources have their traffic sinks on the same regional network to model "communities of interest." The remaining traffic flows traverse the national backbone.

Again, three different traffic loads (light, medium, and heavy) are used for this benchmark. All three scenarios have the same 355 traffic sources (62 Ethernet, 269 MPEG, and 24 TCP). However, different traffic loads are produced by changing the parameters of the sources, rather than by adding traffic sources, as was done with the Wnet benchmarks. This approach makes the partitioning task easier on the large NTN benchmark, and makes simulation comparisons across the three scenarios easier to understand. The first scenario, NTN-1, produces an average link utilization of 30%. The second scenario, NTN-2, corresponds to an average load of 50%. The third scenario, NTN-3, corresponds to an average load of 80%. Even in the lightly loaded NTN-1 and Wnet-1 there are "hot spots" where links are saturated.

Another NTN scenario, NTN-0, contains 53 Ethernet, 30 MPEG, and 16 TCP sources. This scenario was chosen to illustrate performance problems in the WaiKit kernel when TCP sources are dominant.

The NTN benchmark scenarios are summarized in Table I. Each NTN scenario is run for 5 seconds of simulated time, producing 48-216 million simulation events.

4.3 Partitioning

Partitioning the Wnet and NTN benchmark scenarios was done manually, based on a prerun of the simulation and the load data collected in the prerun. The main target is to make the sum of the LP loads on each partition roughly equal. The load of an LP is the time it spends processing events for the entire duration of a simulation run. While there are many ways to divide the LP loads into equal portions, other considerations are important for performing parallel execution.

Partitioning remains a trial-and-error procedure, because numerous factors in a large irregular network model often interact in complex ways. The effect of these factors on execution is mainly determined by event buffer usage and rollback rates. Here, we discuss the major criteria used in partitioning the benchmark scenarios (LP mapping, buffer cache locality, grouping source and sink LPs, minimizing message exchanges).

The LP is the basic element in parallel execution. The simulation kernel forces correct event ordering for each LP. Proper design of LPs in a model will affect the available parallelism. Generally, if there are two event streams that are unrelated, or at least not closely related, then separate LPs should be used for them. This approach may eliminate unnecessary rollbacks for WarpKit, and improve lookaheads in the case of WaiKit. In this respect, the ATM-TN simulator used for parallel execution differs slightly from its original sequential version.

When a cell travels in the simulated network it is carried by an event buffer at each point where it stops for processing, and a new buffer is used to carry it to the next point. If the cell remains on the same processor on the way from source to sink, the buffers that carry it will stay on the same processor. Thus, the assignment of LPs to processors should try to minimize the number of processors that a cell has to visit in order to reduce undesirable cache effects (i.e., cache misses).

Most of the LPs in ATM-TN have a message "fanout" of one; that is, for each message received, one message will be sent to the next LP. The traffic source and sink LPs are exceptions. Source LPs typically have a fanout greater than one (e.g., several cells in each traffic burst produced by the Ethernet source model), and sink LPs typically have a fanout less than one (e.g., Ethernet sinks consuming incoming cells). Assigning source and sink LPs of the same traffic flow to the same processor will balance the buffers consumed at the source and returned at the sink, reducing buffer migration between processors. This is especially good for open-loop traffic sources (e.g., Ethernet, MPEG), where sinks never send any messages. However, for closed-loop TCP traffic flows, where the sinks send acknowledgments back to the source, the source and sink LPs are best grouped together with the LPs of the switches to which they are connected. Otherwise, cells arriving at the sink require a different processor than the acknowledgments (or other sink-initiated cells) returning to the source. The result is a higher chance of rollbacks. Furthermore, rollbacks at the start of message

flow are particularly bad, as they tend to cascade, causing further rollbacks downstream [Tay et al. 1998].

Another criterion is the reduction of message exchanges between processors. For large scenarios with heavy traffic loads, such as NTN-3, "block partitioning" is better. Block partitioning divides the network into geographic regions based on vicinity, under the assumption that most communication is "local." (The same idea is used in military and wireless network simulations [Meyer and Bagrodia 1998; Zeng et al. 1998].) As a result, most messages in the model remain on the same processor. For models with few switches and light traffic loads, partitioning by traffic flows may yield better performance, since messages originating from a source will traverse fewer processors, thus reducing communication between processors as well.

The same partitions of all the benchmarks are used for both WarpKit and WaiKit experiments. A good partition for WaiKit is also a good partition for WarpKit. The reverse also turned out to be true, i.e., a good partition for WarpKit resulted in good performance for WaiKit. This strongly suggests that the dominant factor in partitioning lies with the characteristics of the application. The benefits of efficient buffer use are the same for both kernels, and reduced rollbacks often translate into better lookaheads for WaiKit, as mentioned previously.

5. PERFORMANCE RESULTS

This section presents the results of kernel performance for the seven benchmark scenarios. All experiments were run on a dedicated SGI Power Challenge shared-memory multiprocessor with 18 R8000 CPUs running IRIX64 6.1. Section 5.1 discusses the statistical significance of the kernel performance results presented in subsequent sections. Section 5.2 gives the sequential simulation performance results for the Wnet and NTN benchmarks. Section 5.3 presents the results for the WarpKit optimistic parallel simulation kernel. Section 5.4 shows the results for the WaiKit conservative parallel simulation kernel. Finally, Section 5.5 compares the performance of the different simulators for each benchmark scenario.

5.1 Validating Performance Results

The ATM-TN simulator has been used for studies of ATM traffic; TCP over ATM performance, call-admission control and congestion control algorithms [Chen et al. 1995; Gurski and Williamson 1996; Patel and Williamson 1997; Wang et al. 1999; Zoranovic and Williamson 1999]. The validation of network model simulation results has been explored over several years at a number of levels, including individual LP event tracing, use of simple scenarios for which comparable analytic results can be calculated, and comparisons of simulation results with measurements collected within carefully controlled network experiments. Here, in addition to model validation, it is important that model simulation results are essentially identical when executed on different kernels and on different numbers of processors.

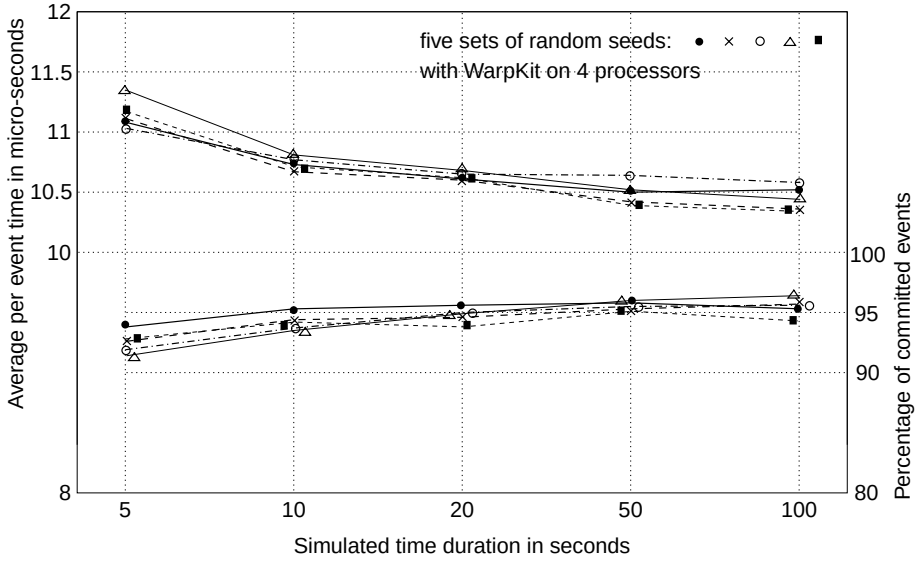

Fig. 5. Stability of performance results for NTN-0.

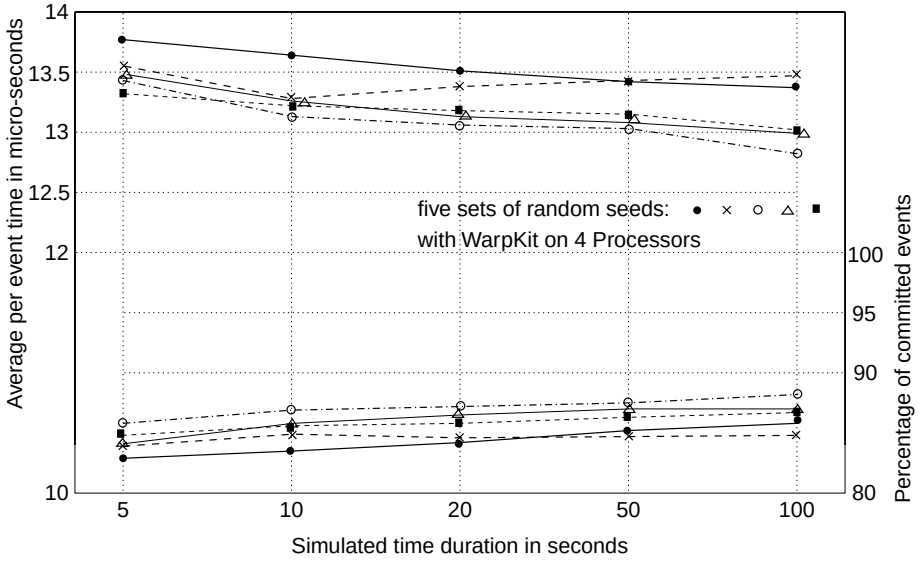

Fig. 6. Stability of performance results for NTN-1.

The ATM-TN has extensive statistics' collection for all network components being simulated. A valid parallel simulation should produce the same (or very close) statistical results for all network components as the sequential simulator, regardless of the kernel or the number of processors used. For the seven benchmark scenarios, all model simulation results are identical when run sequentially on a single processor, or when executed in parallel on different numbers of processors. The only difference is the order

of records (lines) in the output reports. Due to the asynchronous nature of parallel execution, different components may report their statistics data at different (real) times in each run. For Wnet-1 with WarpKit, the total number of committed events (events correctly processed) is always exactly the same for runs on different numbers of processors, which demonstrates the stability of the WarpKit kernel. For other test scenarios, the total number of events varies slightly (e.g., a difference of at most several thousand events out of 20 to 200 million events). The reason for these different event counts is that Wnet-1 is a deterministic model in which no two events for the same LP have identical timestamps, while the other scenarios are nondeterministic. The latter occurs because the TCP model can generate many events with exactly the same timestamp (due to the fact that zero host-processing time is assumed). It is well known that nondeterministic models typically result in different event-processing orders for those events with identical timestamps. This nondeterminism may or may not affect the simulation results, depending on the application. In the experiments reported here, no variation in simulation statistical results occurred (within the precision of the statistics reported by the simulations).

The main focus of this article is on the execution speed of alternative simulation kernels for a range of benchmark scenarios. Our primary concern in this section is the validity and stability of our measurement of simulation execution speed. For measuring kernel performance, we are interested in the total execution time for a particular scenario. The simulation runs used to collect performance data are relatively short (e.g., 5-10 seconds of simulated time) compared to those used in real ATM network simulation studies (e.g., 10-1000 seconds). The primary reason for this is the need to fit multiple runs into the weekly time periods during which we have exclusive use of the entire SGI Power Challenge. However, these run lengths are adequate for comparing the performance of the three simulation kernels.

To justify our claim, we present the results of two performance stability tests. Figure 5 shows the results for NTN-0 with WarpKit on four processors. Five sets of random number seeds are used. For each set of seeds, five test runs with increasing simulated time (5, 10, 20, 50, and 100 seconds) are carried out. The upper part of the graph gives the average time in microseconds spent on each event, which is used to measure the speed of simulation progress. The lower the per-event time, the faster the simulation. The lower part of the figure shows the percentage of committed events. For example, a value of 80% means that out of every 10 processed events, 8 events are correct, 2 events are erroneous and had been rolled back, that is, a 20% rollback rate. The graph indicates the inverse relationship between per-event time and percentage of committed events. With increasing simulation time duration, the percentage of committed events increases and per-event time decreases. The decrease of per-event time is more pronounced than the increase in the percentage of committed events, due to the extra cost of rollbacks. The variation in per-event time for

Table II. Per-Event Time in Microseconds for CelKit Sequential Simulation Kernel (SGI Power Challenge R8000)

benchmark scenario	per-event execution time in microseconds	total execution time in seconds	potential parallelism
Wnet-1	11.50	253	11
Wnet-2	11.43	206	15
Wnet-3	11.32	204	19
NTN-0	14.80	710	42
NTN-1	18.75	1369	66
NTN-2	20.32	2682	60
NTN-3	21.30	4601	64

different random seeds is less than 4%. We also observe that the degree of variability in the per-event time from one run to the next is not noticeably lower after 100 seconds of simulated time than it is after 5 seconds of simulated time; hence our choice of 5-second runs for the NTN scenarios.

Figure 6 gives the test results for NTN-1, which shows the same trend as that for NTN-0. The variation in per-event time from one run to another for this scenario is higher, but still less than 7%. The influence of rollback rate on per-event time is more dominant in this case. Another type of test is to repeatedly run the same simulation without any changes. As an example, the per-event times for 6 runs of NTN-1 (each for 5 seconds of simulated time) on 4 processors vary between 13.50 and 13.54, which is a variation of 0.3%.

5.2 CelKit Performance Results

The performance results for the event-list-based Sequential Simulator (CelKit) are presented in Table II, which shows the per-event time in microseconds on the SGI Power Challenge for the 7 ATM-TN benchmark scenarios. (The per-event time is defined as the total execution time divided by the total number of (committed) events, for both sequential and parallel execution.) The table gives the total wall-clock execution time for 5 or 10 seconds of simulated time as well. The wall-clock time for sequential execution of the benchmarks ranges from about 4 minutes for the Wnet benchmarks to 77 minutes for the NTN-3 model. The per-event time is approximately 11.5 microseconds for all three Wnet benchmarks, while the per-event time grows slightly with the traffic load on the NTN benchmarks, up to 21.3 microseconds per simulation event in NTN-3. The growth is largely a result of the increase in the number of events on the event list and larger memory required to run the simulation.

Table II also includes a column indicating the *potential*, or *intrinsic parallelism*, for each network scenario. The time it takes to run a simulation model sequentially is called the *sequential execution time* of the model. A model's intrinsic parallelism is defined as the ratio of its sequential execution time over the *least time* needed to execute that model, assuming there are always enough processors to enable the execution of any LP ready

to execute. The least time is determined by the user execution time along the model's *critical path*. Events along the critical path have to be processed in sequence. The critical path is the path that takes the longest time to process. No parallel execution can take less time than this time. The value of intrinsic parallelism is calculated using a feature of CelKit. It measures the total execution time along the critical path in the simulation. The critical path is discussed more completely in Xiao and Unger [1995b]. The value given in the table is the total sequential time divided by the critical path time. The resulting ratio is a rough measure of the best relative speedup possible for the parallel simulators on that particular problem. The Wnet scenarios have potential speedups of 11 to 19, NTN-0 is somewhat larger at 42, and the 3 NTN scenarios are grouped at approximately 60 to 66.

The per-event execution time for NTN-0 is substantially lower than other NTN scenarios, due to its much lighter traffic load. The length of the event list increases with the traffic load as more events are generated and scheduled for processing. As a result, the cost of queue operations, which is the major part of system overhead in a central event list-based sequential kernel, increases significantly.

5.3 WarpKit Performance Results

Performance results for the WarpKit optimistic parallel simulation kernel are shown in Figure 7. The figure shows the relative speedup for the WarpKit simulator on the ATM-TN benchmarks as the number of processors on the SGI Power Challenge is varied from 1 to 16. Relative speedup is the ratio of the time required (for WarpKit) on one processor divided by the time required (for WarpKit) using N processors ($N \geq 1$). This kind of result is mainly useful for analyzing how well a particular algorithm scales as the number of processors increases. Note that these results do not answer the question of how well the algorithm performs compared to either sequential or other parallel algorithms. (The latter comparisons are deferred to Section 5.5.)

The results in Figure 7 show that WarpKit achieves good relative speedup as the number of processors increases. The results are also consistent, in that at each number of processors, the speedup curves for the different benchmark scenarios are nearly always in the same relative order, which implies that the relative performance is primarily determined by the size, structure, traffic load, and partitioning of each problem. The only exception to this is at 8 processors, where Wnet-3 has slightly better performance than Wnet-2.

There is some evidence that similar ATM-TN simulation problems with more traffic give better speedup. The sequence NTN-1, NTN-2, and NTN-3, for example, shows steadily increasing speedup (these scenarios differ only in their loads). The tendency is less clear for the other scenarios, however. The Wnet scenarios are likely to be limited by their potential parallelism of less than 15 (for example, Wnet-1 achieves half of its potential on 16

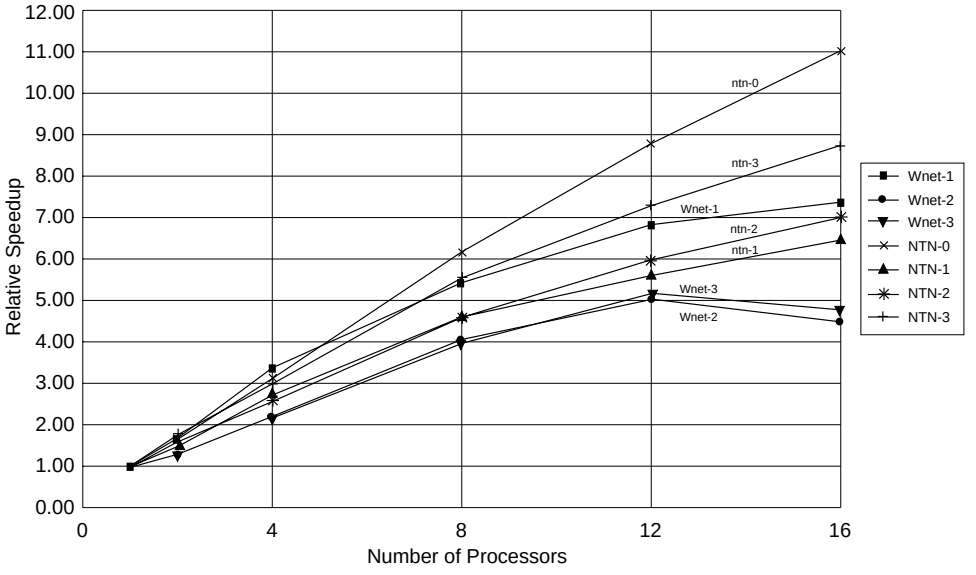

Fig. 7. Relative speedup of WarpKit.

processors). Due to limited parallelism, idling time and the rollback rate increase on 16 processors, resulting in higher system overhead. The speedup curve for Wnet reaches its peak near 12 processors. The best relative speedup observed is for NTN-0, which has traffic loads that are largely local to each of the five regions of the network, and light TCP traffic across different regions (a scenario especially good for "block partitioning.")

5.4 WaiKit Performance Results

The performance results for the WaiKit conservative parallel simulation kernel are shown in Figure 8. The results show that WaiKit has generally lower (and more erratic) relative speedup than WarpKit (although, as we will see later, it did achieve excellent speedup on one scenario). The different networks give inconsistent results as the number of processors is increased. It is unclear why this occurs, although it is possible that WaiKit is more sensitive to small imbalances in load caused by imperfect partitioning. Also, no speedup was achieved at all for the Wnet-3 network (this case is discussed further in Section 5.5). Contrary to the case with WarpKit, NTN-0 performs far worse than the other three NTN scenarios. This is caused by the TCP traffic that traverses the entire network. The presence of the (closed-loop) TCP traffic limits the lookaheads in the conservative WaiKit kernel, resulting in small simulation time advances that slow down the simulation. This TCP traffic has little impact on the performance of the WarpKit kernel.

5.5 Comparative Results

Figures 9 to 15 compare the performances of the three simulators (CelKit, WarpKit, and WaiKit) on each individual benchmark scenario. We have

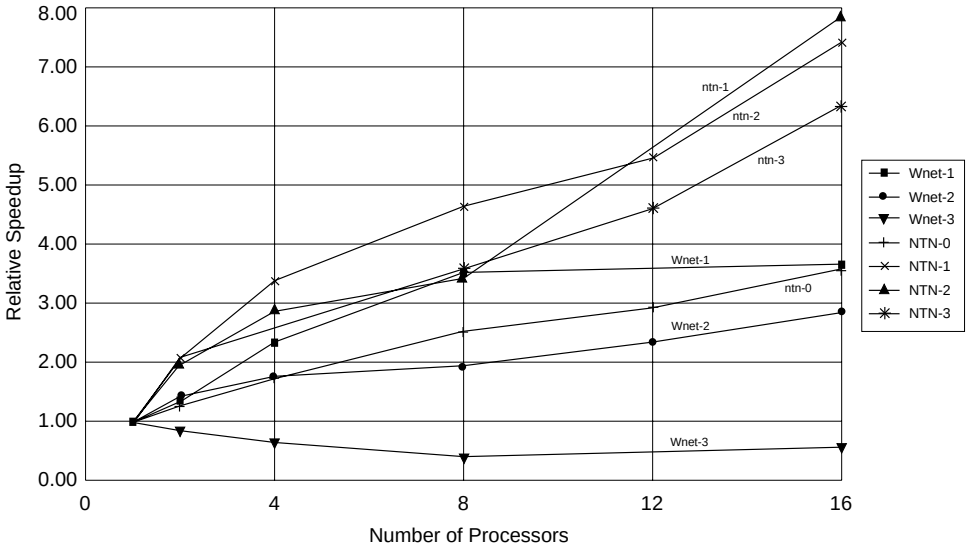

Fig. 8. Relative speedup of WaiKit.

separated the results in this way because each of the scenarios has different amounts of computation, potential parallelism, and partitioning. However, for a given number of processors the results should be directly comparable. Each graph plots the total number of events per second achieved against the number of processors for each of the three kernels. For the sequential simulator (CelKit), a single horizontal line has been added at its uniprocessor performance point.

5.5.1 *Wnet-1.* For the Wnet-1 scenario, WaiKit achieves very good performance (see Figure 9). On one processor it is already faster than CelKit, and it improves from there. WaiKit reaches close to its best performance at 8 processors, where it is over 4 times faster than CelKit. While WarpKit achieves good smooth relative speedup, it only performs better than CelKit on 4 and more processors. Wnet-1 is a special scenario because it has only Ethernet and MPEG traffic sources. All traffic flows are thus unidirectional, and there are no loops in the traffic flows. As a result, the average number of events executed each time an LP is scheduled in WaiKit is potentially very high (in practice it is limited by available buffer space). This explains its very good performance, including why it can outperform a sequential simulator. What it is effectively doing is avoiding the overheads of event list insertion and removal by its simple scheduling algorithm.

5.5.2 *Wnet-2.* The Wnet-2 scenario has fewer LPs and events than Wnet-1, but a wider range of traffic source types. In particular, it includes TCP sources, which introduce feedback loops into the traffic flows. The speedup of both WaiKit and WarpKit behave erratically as the number of processors is increased (see Figure 10). For WaiKit, the performance at 8 processors seems anomalously low, but it achieves the best performance

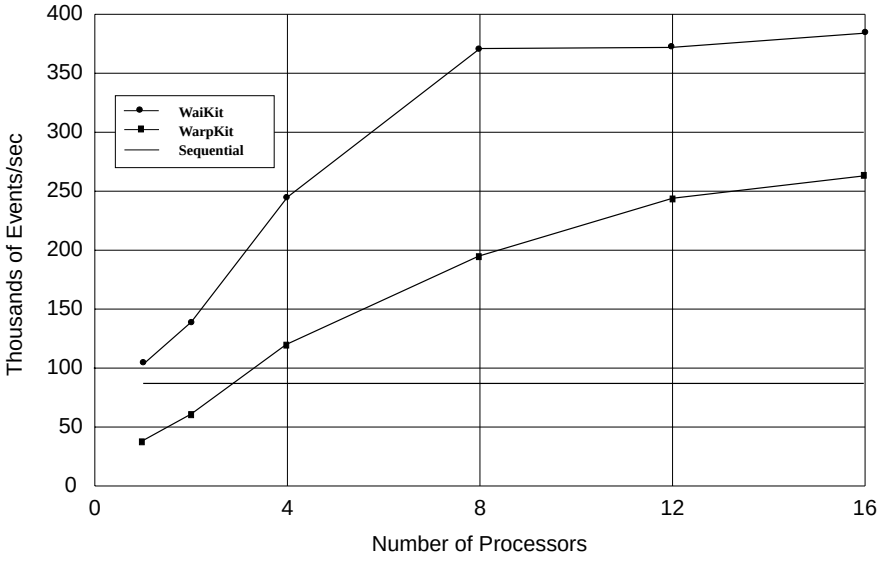

Fig. 9. Performance results for Wnet-1 benchmark scenario.

overall on 16 processors. The WarpKit performance drops on 16 processors, due to TCP feedback loops and a consequent high rollback rate. On one processor, CelKit is slightly faster than WaiKit, but WaiKit overtakes CelKit on 2 and more processors. WarpKit does not break even with CelKit until 4 processors are used.

5.5.3 *Wnet-3*. The Wnet-3 scenario differs from Wnet-2 by the addition of 4 Web traffic sources. Unfortunately, due to the way the Web source is modeled, the performance of WaiKit on this scenario is abysmal (see Figure 11). The Web model includes a tight almost-zero-delay loop between source and destination objects for simulated file transfers. This was done as an easy way to communicate statistical information about the file size of the simulated transfers. Unfortunately, this created a situation where each time WaiKit scanned the list of LPs, it was able to make only a tiny time advance. The result emphasizes the sensitivity of conservative kernels to such application-level traffic-modeling decisions. The WarpKit kernel demonstrates fairly robust performance, even with this scenario.

5.5.4 *NTN-0*. The NTN network is larger than any of the Wnet cases, and it includes significant traffic loops in the form of TCP sources. WaiKit performs poorly in this scenario (see Figure 12), although it does achieve good relative speedup, and breaks even with CelKit by 16 processors. In contrast, WarpKit achieves its best overall performance in this scenario, with a steadily improving speedup that breaks even with CelKit before 4 processors and achieves a 5-fold speedup over CelKit by 16 processors. WaiKit performs poorly due to the TCP loops discussed earlier.

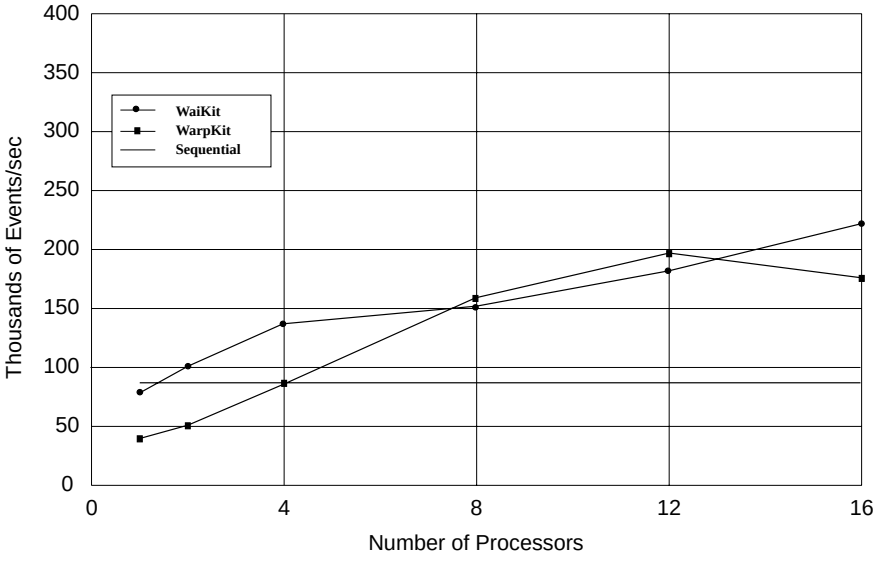

Fig. 10. Performance results for Wnet-2 benchmark scenario.

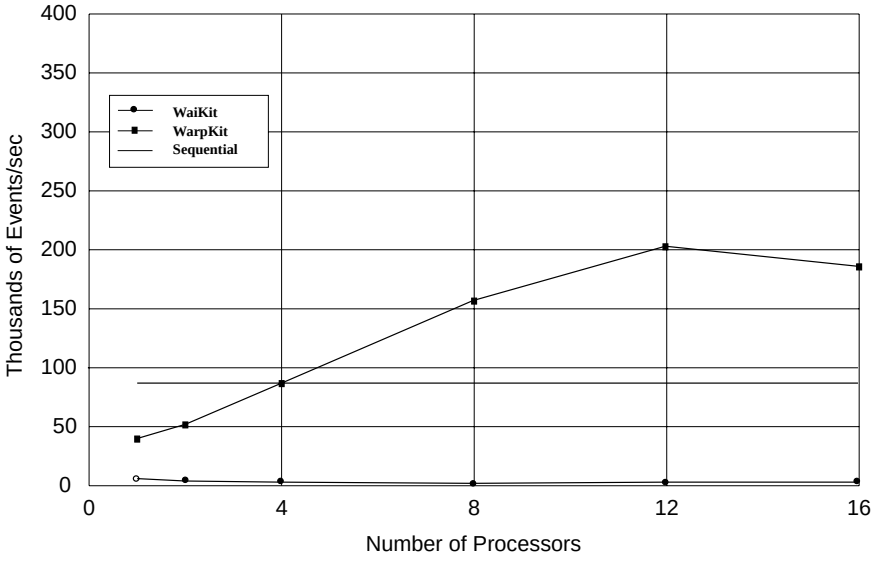

Fig. 11. Performance results for Wnet-3 benchmark scenario.

5.5.5 *NTN-1, NTN-2, and NTN-3*. The NTN-1, NTN-2, and NTN-3 networks are identical, except for the volume of traffic loads. Figures 13, 14, and 15 show that the performance of the sequential kernel drops slightly with increased load, due to the increase in the size of the event list. The performance of WaiKit and WarpKit is consistent across the three scenarios. In each case, WarpKit breaks even with CelKit between 2 and 4 processors and proceeds to achieve its best performance at 16 processors.

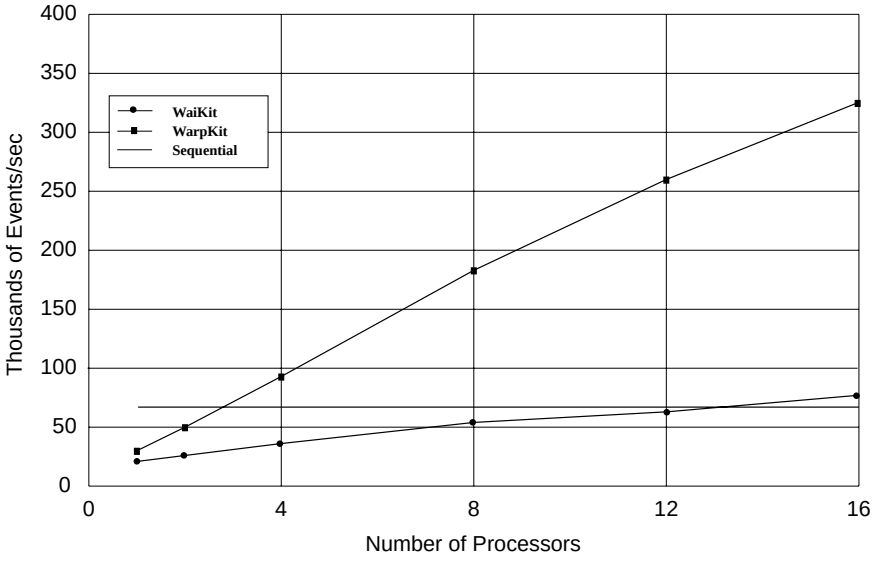

Fig. 12. Performance results for NTN-0 benchmark scenario.

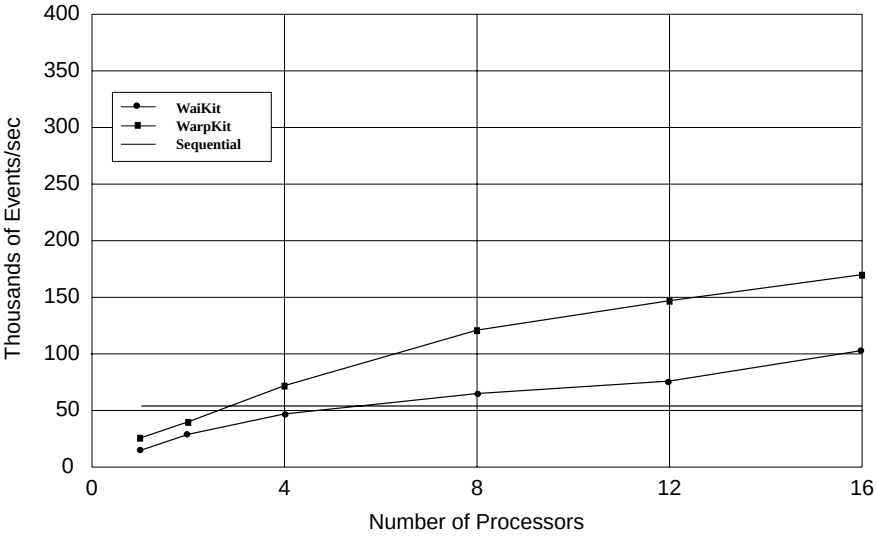

Fig. 13. Performance results for NTN-1 benchmark scenario.

The overall performance of WarpKit also improves steadily with the increased loads. WaiKit has more erratic performance, particularly between 4 and 8 processors. However, it improves significantly as the load increases from NTN-1 (where WaiKit is significantly below WarpKit across the full range of numbers of processors) to NTN-3 (where WaiKit actually outperforms CelKit and WarpKit, in the 2-processor case only). This improvement is to be expected, as the increasing load increases the average number of events to be processed each time an LP is scheduled, thus

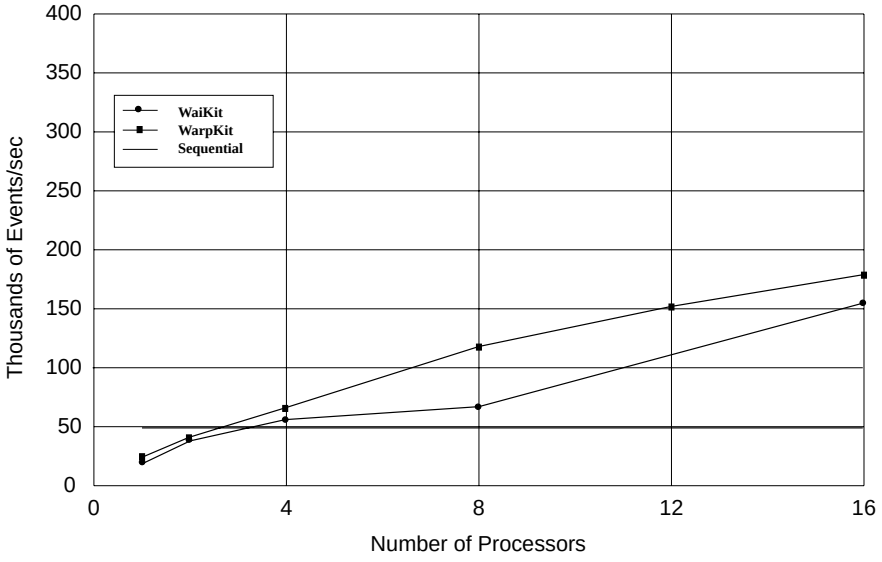

Fig. 14. Performance results for NTN-2 benchmark scenario.

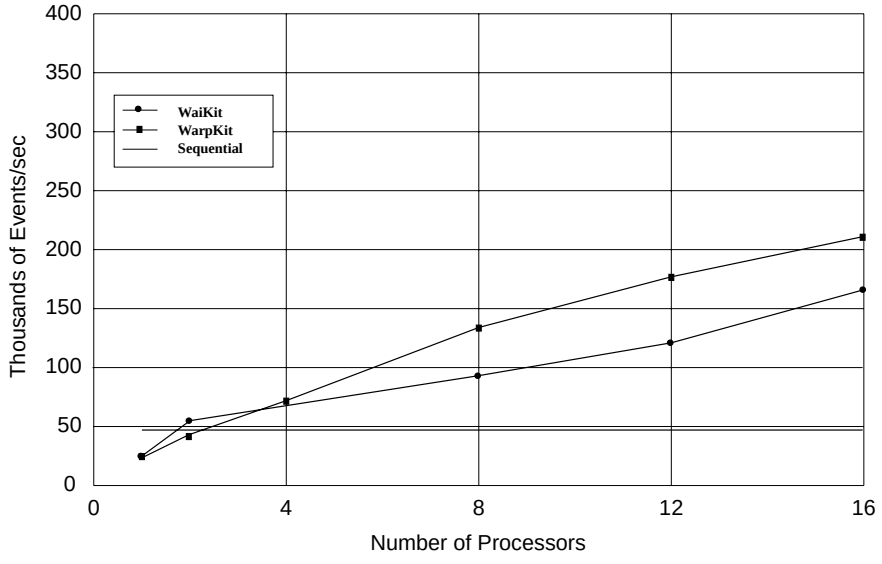

Fig. 15. Performance results for NTN-3 benchmark scenario.

reducing the overheads for WaiKit (recall that the lookaheads are static, based primarily on network topology).

6. SUMMARY AND CONCLUSIONS

This article presents a performance study comparing conservative and optimistic parallel simulation kernels, as well as a sequential simulation kernel, on a set of large irregular ATM network benchmarks. The study

evaluates two different parallel simulation kernels with respect to relative speedup, and also absolute speedup compared to a sequential kernel, as well as the sensitivity of simulator performance to the size, structure, topology, and traffic flow in the simulated networks.

Our experimental results show that parallel simulation can offer substantial execution speedup for ATM network applications. While the application-level event granularity in a cell-level ATM simulator is small, the high delay-bandwidth products of real ATM networks translate into simulations with high potential parallelism. We show that within a circumscribed and well-characterized domain (i.e., cell-level ATM networks), it is possible to make effective use of parallelism. The major remaining barrier to the use of such a system (by users who are not expert in parallelism) is the partitioning of the simulations across multiple PEs.

Our results also highlight the fragile nature of conservative parallel simulation performance. In particular, we found that the conservative kernel is very susceptible to small changes in the modeling code, which makes it possible for it to give the best performance on a few scenarios and the worst performance on others. The optimistic kernel, on the other hand, provides excellent relative speedup and more robust performance on the benchmarks tested. However, its performance is limited by the implementation overheads, which are actually higher than the grain of the model computation for these low granularity applications.

It is worth noting what this work has *not* demonstrated. It has not shown that arbitrary simulation problems can be executed in parallel by inexpert users. The success of the system depends on careful attention to detail and the influence of parallel execution at all levels of the system. This started at the language interface (SimKit), which was constrained to use a logical process decomposition and communication via messages, together with a simple lightweight set of calls. The ATM-TN model was engineered to fit the LP decomposition by careful placement of parallel activities in different LPs and by putting in the extra effort needed to work with the SimKit interface (for example, by eschewing shared state). As described earlier, the two parallel kernels also include many careful optimizations to ensure low overheads. The relative importance of these is probably particular to the ATM-TN problem. Different optimizations might be important for different problem classes.

The performance results presented in Section 5 depend on the underlying shared memory platform. In particular, the absolute performance results directly depend on the SGI platform. However, we believe the comparative results and conclusions for SMP architectures are not sensitive to the specific SGI platform. We have performed similar experiments on an 8-processor SPARC SMP platform with similar results. In general, implementing kernels and kernel optimizations also has a significant impact on performance—for example, implementations must consider cache performance.

The original intention was that the SimKit interface provide a transparent portable interface independent of the underlying simulation kernel.

This has generally been successful, but did manifest two problems. In the case of WarpKit, it was necessary to extend this kernel to deal with event-flow control issues, as well as add code for state-saving (although, in principle, we understand how to do state-saving efficiently and transparently [Gomes 1996]). In the case of WaiKit, it was necessary to specify the channel connectivity between LPs and to compute lookahead information for the channels. More work is also needed to improve modeling practice and avoid small lookahead loops.

For future work, the two major problems are those of load-balancing and further reducing kernel-level overheads. For ATM-TN, dynamic load-balancing is very difficult. The low per-event granularity means that little time can be spent thinking about where work is to be done or on moving LPs around between processors. For example, the overheads for measuring actual execution times by LPs is too high to be useful. We believe this problem would be even harder in distributed memory architectures, due to the extra complexity and overheads of LP migration in such an environment (the cost of migration still exists in shared-memory systems—it is just hidden as cache-miss overheads). The success of LP-level rather than event-level scheduling in WaiKit points to one possible solution, namely to do the load-balancing at a larger grain size than that of events or even LPs. This is a promising idea that we explored in a new conservative algorithm and the *TasKit* simulation kernel that implements it [Xiao et al. 1999]. The performance of the new kernel has demonstrated dramatic improvement. (Comparisons of *TasKit* kernel performance will be presented in a subsequent article.)

The second major problem is that of continuing to reduce the overheads due to parallel execution. WaiKit shows that it may be possible to do this by paying careful attention to the grain size at which different overheads occur and by using simple algorithms on these larger grain sizes. Unfortunately, WaiKit also shows extreme sensitivity to the application-level modeling code; it has not yet been demonstrated that these problems can be overcome. Progress may require further violation of SimKit's interface (for example, by making channels and their lookahead explicitly accessible to users), which makes the modeling problem even harder.

One interesting feature of these results is the general robustness and good relative speedup of the WarpKit kernel. The problem is that it has high overheads which are difficult to overcome in a low-granularity application such as the ATM-TN. However, in other problem domains with higher granularity (say over 100 μ-*sec* per-event), TimeWarp is probably a good choice for obtaining speedup, although still requiring expert guidance on issues such as model partitioning and event flow control.

ACKNOWLEDGMENTS

The authors wish to thank the many people who have directly contributed to the design, implementation, and use of the ATM-TN simulator in its current form, and to its ongoing existence in our collaborative TeleSim

research project. To name a few, these people include: Martin Arlitt, Ying Chen, Alan Covington, Adi Damian, Julie Doerksen, Mark Fox, Steve Franks, Pawel Gburzynski, Fabian Gomes, Remi Gurski, Tim Harrison, Xiaoming Li, Guang Lu, Theodore Ono-Tesfaye, Srinivasan Ramaswamy, Rob Simmonds, and Jiayun Zhu. In particular, Remi Gurski and Tim Harrison were instrumental in establishing the Wnet and NTN benchmark scenarios in ATM-TN.

REFERENCES

ARLITT, M., CHEN, Y., GURSKI, R., AND WILLIAMSON, C. 1995. Traffic modeling in the ATM-TN telesim project: Design, implementation, and performance evaluation. In *Proceedings of the 1995 Summer Conference on Computer Simulation* (SCSC'95, Ottawa, Ont.). 847–851.

ARLITT, M. AND WILLIAMSON, C. 1995. A synthetic workload model for internet mosaic traffic. In *Proceedings of the 1995 Summer Conference on Computer Simulation* (SCSC'95, Ottawa, Ont.). 852–857.

ARLITT, M. F. AND WILLIAMSON, C. L. 1997. Internet Web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw. 5*, 5, 631–645.

ATM FORUM. 1996. Traffic management 4.0 specification. ATM Forum Technical Committee.

AVRIL, H. AND TROPPER, C. 1995. Clustered time warp and logic simulation. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation* (PADS '95, Lake Placid, NY, Jun 14-16). IEEE Computer Society Press, Los Alamitos, CA.

BAGRODIA, R. AND LIAO, W. 1994. Masie: A language for design of efficient discrete-event simulations. *IEEE Trans. Softw. Eng. 20*, 4 (Apr.), 225–238.

BLANCHARD, T. D., LAKE, T. W., AND TURNER, S. J. 1994. Cooperative acceleration: robust conservative distributed discrete event simulation. *SIGSIM Simul. Dig. 24*, 1 (July), 58–64.

BROWN, R. 1988. Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem. *Commun. ACM 31*, 10 (Oct.), 1220–1227.

CAI, W., LETERTRE, E., AND TURNER, S. J. 1997. Dag consistent parallel simulation: a predictable and robust conservative algorithm. *SIGSIM Simul. Dig. 27*, 1, 178–181.

CAROTHERS, C. D., FUJIMOTO, R. M., AND LIN, Y.-B. 1995. A case study in simulating PCS networks using Time Warp. *SIGSIM Simul. Dig. 25*, 1 (July), 87–94.

CHANDY, K. M. AND MISRA, J. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng. 5*, 5 (Sept.), 440–452.

CHEN, Y.-A. AND BAGRODIA, R. 1998. Shared memory implementation of a parallel switch-level circuit simulator. *SIGSIM Simul. Dig. 28*, 1, 134–141.

CHEN, Y., DENG, Z., AND WILLIAMSON, C. 1995. A model for self-similar Ethernet lan traffic: Design, implementation, and performance implications. In *Proceedings of the 1995 Summer Conference on Computer Simulation* (SCSC'95, Ottawa, Ont.). 831–837.

CLEARY, J. G. AND TSAI, J.-J. 1996. Conservative parallel simulation of ATM networks. *SIGSIM Simul. Dig. 26*, 1, 30–38.

CROVELLA, M. E. AND BESTAVROS, A. 1996. Self-similarity in World Wide Web traffic: evidence and possible causes. *SIGMETRICS Perform. Eval. Rev. 24*, 1, 160–169.

DAS, S. R. AND FUJIMOTO, R. M. 1993. A performance study of the cancelback protocol for Time Warp. *SIGSIM Simul. Dig. 23*, 1 (July), 135–142.

DAS, S., FUJIMOTO, R., PANESAR, K., ALLISON, D., AND HYBINETTE, M. 1994. GTW: A time warp system for shared memory multiprocessors. In *Proceedings of the 1994 Winter Conference on Simulation* (WSC '94, Lake Buena Vista, FL, Dec. 11–14), M. S. Manivannan and J. D. Tew, Eds. Society for Computer Simulation, San Diego, CA, 1332–1339.

FUJIMOTO, R. M. 1989. Time warp on a shared memory multiprocessor. In *Proceedings of the International Conference on Parallel Processing* (ICPP '89, Aug.). Pennsylvania State University, University Park, PA, 242–249.

FUJIMOTO, R. M. 1990. Parallel discrete event simulation. *Commun. ACM 33*, 10 (Oct.), 30–53.

GARRETT, M. W. AND WILLINGER, W. 1994. Analysis, modeling and generation of self-similar VBR video traffic. *SIGCOMM Comput. Commun. Rev. 24*, 4 (Oct.), 269–280.

GOMES, F. 1996. Optimizing incremental state-saving and restoration. Ph.D. Dissertation. University of Calgary, Calgary, Canada.

GOMES, F., CLEARY, J., COVINGTON, A., FRANKS, S., UNGER, B., AND ZIAO, Z.-E. 1995. SimKit: a high performance logical process simulation class library in C+. In *Proceedings of the 1995 Winter Conference on Simulation* (WSC '95, Arlington, VA, Dec. 3–6), C. Alexopoulos and K. Kang, Eds. ACM Press, New York, NY, 706–713.

GURSKI, R. AND WILLIAMSON, C. 1996. TCP over ATM: Simulation model and performance results. In *Proceedings of the 1996 IEEE International Conference on Computers and Communications* (Phoenix, AZ, Mar.). IEEE Computer Society Press, Los Alamitos, CA, 328–335.

JEFFERSON, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst. 7*, 3 (July), 404–425.

JEFFERSON, D., BECKMAN, B., AND WIELAND, F. 1987. Distributed simulation and the time warp operating system. In *Proceedings of the Eleventh ACM Symposium on Operating System Principles* (Austin, TX, Nov. 8-11), L. Belady, Chair. ACM Press, New York, NY.

LE GALL, D. 1991. MPEG: a video compression standard for multimedia applications. *Commun. ACM 34*, 4 (Apr.), 46–58.

LELAND, W. E., TAQQU, M. S., WILLINGER, W., AND WILSON, D. V. 1994. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Trans. Netw. 2*, 1 (Feb.), 1–15.

MARTINE, D., WILSEY, P., AND MCBRAYER, T. 1995. WARPED (version 0.5). University of Cincinnati, Cincinnati, OH.

MCCORMACK, W. AND SARGENT, R. G. 1981. Analysis of future event set algorithms for discrete event simulation. *Commun. ACM 24*, 12 (Dec.).

MELAMED, B. 1992. TES modeling of video traffic. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E75-B*, 1292–1300.

MEYER, R. AND BAGRODIA, R. 1998. Improving lookahead in parallel wireless network simulation. In *Proceedings of the Sixth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS '98, Montreal, Que., July). 262–267.

MINZER, S. E. 1991. Broadband ISDN and asynchronous transfer mode (ATM). In *Broadband Switching: Architectures, Protocols, Design, and Analysis*, C. Chas, V. K. Konangi, and M. Sreetharan, Eds. IEEE Computer Society Press, Los Alamitos, CA, 81–89.

NICOL, D. M. 1996. Principles of conservative parallel simulation. In *Proceedings of the 1996 Winter Conference on Simulation* (WSC '96, Coronado, CA, Dec. 8–11), J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, Eds. ACM Press, New York, NY, 128–135.

NICOL, D. AND HEIDELBERGER, P. 1996. On extending more parallelism to serial simulators. *SIGSIM Simul. Dig. 26*, 1, 202–205.

PATEL, A. AND WILLIAMSON, C. 1997. Effective bandwidth of self-similar traffic sources: Theoretical and simulation results. In *Proceedings of the IASTED Conference on Applied Modeling and Simulation* (Banff, AB, Canada, July). 298–302.

PAXSON, V. AND FLOYD, S. 1995. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. Netw. 3*, 3 (June), 226–244.

PHAM, C. D., BRUNST, H., AND FDIDA, S. 1998. Conservative simulation of load-balanced routing in a large ATM network model. *SIGSIM Simul. Dig. 28*, 1, 142–149.

STEINMAN, J. 1992. SPEEDES: A unified approach to parallel simulation. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation* (PADS '92). ACM Press, New York, NY.

STEVENS, W. R. 1994. *TCP/IP Illustrated: The Protocols (Vol. 1)*. Addison-Wesley Longman Publ. Co., Inc., Reading, MA.

SU, W. AND SEITZ, C. 1989. Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm. In *Proceedings of the Conference on Distributed Simulation* (Miami, FL).

TANENBAUM, A. 1996. *Computer Networks*. 3rd ed. Prentice-Hall, New York, NY.

TAY, S. C., TEO, Y. M., AND AYANI, R. 1998. Performance analysis of time warp simulation with cascading rollbacks. *SIGSIM Simul. Dig. 28*, 1, 30–37.

UNGER, B. W., GOMES, F., ZHONGE, X., GBURZYNSKI, P., ONO-TESFAYE, T., RAMASWAMY, S., WILLIAMSON, C., AND COVINGTON, A. 1995. A high fidelity ATM traffic and network simulator. In *Proceedings of the 1995 Winter Conference on Simulation* (WSC '95, Arlington, VA, Dec. 3–6), C. Alexopoulos and K. Kang, Eds. ACM Press, New York, NY, 996–1003.

WALLACE, G. K. 1991. The JPEG still picture compression standard. *Commun. ACM 34*, 4 (Apr.), 30–44.

WANG, Y., WILLIAMSON, C., AND DOERKSEN, J. 1999. CAC performance with self-similar traffic: Simulation study and performance results. In *Proceedings of the Seventh International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOT '99, Baltimore, MD, Oct.). 102–111.

WIELAND, F., BLAIR, E., AND ZUKAS, T. 1995. Parallel discrete-event simulation (PDES): a case study in design, development, and performance using SPEEDES. *SIGSIM Simul. Dig. 25*, 1 (July), 103–110.

WILLIAMSON, C., UNGER, B., AND XIAO, Z. 1998. Parallel simulation of ATM networks: Case study and lessons learned. In *Proceedings of the Second Canadian Conference on Broadband Research* (CCBR '98, Ottawa, Ont., June). 78–88.

XIAO, Z., GOMES, F., UNGER, B., AND CLEARY, J. 1995. A fast asynchronous GVT algorithm for shared memory multiprocessor architecture. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation* (PADS '95, Lake Placid, NY, Jun 14-16). IEEE Computer Society Press, Los Alamitos, CA.

XIAO, Z. AND UNGER, B. 1995a. Notes on parallelizing ATM-TN cell-level simulation models. Tech. rep. 98-629-20. University of Calgary, Calgary, Canada.

XIAO, Z. AND UNGER, B. 1995b. Report on WarpKit: Performance study and improvement. Tech. rep. 98-628-19. University of Calgary, Calgary, Canada.

XIAO, Z., UNGER, B., SIMMONDS, R., AND CLEARY, J. 1999. Scheduling critical channels in conservative parallel discrete event simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation* (PADS '99, Atlanta, GA, May). IEEE Computer Society Press, Los Alamitos, CA, 20–28.

ZENG, X., BAGRODIA, R., AND GERLA, M. 1998. GloMoSim: a library for parallel simulation of large-scale wireless networks. *SIGSIM Simul. Dig. 28*, 1, 154–161.

ZORANOVIC, M. AND WILLIAMSON, C. 1999. Performance and robustness testing of explicit-rate ABR flow control schemes. In *Proceedings of the Seventh International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOT '99, Baltimore, MD, Oct.). 11–20.

Received: May 1999; revised: April 2000; accepted: April 2000