Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

USE OF DASSF IN A SCALABLE MULTIPROCESSOR WIRELESS SIMULATION ARCHITECTURE

Trefor J. Delve
Nathan J. Smith

Wireless Access Technology Research
Motorola Labs
1501 West Shure Drive
Arlington Heights, IL 60004, U.S.A.

ABSTRACT

The problem of efficient load distribution and scaling of large-scale wireless communication system simulation on multiprocessor architectures (both shared memory and cluster arrangements) is considered. A flexible architecture based upon DaSSF, the Dartmouth Scalable Simulation Framework discrete event engine, is presented and evaluated. The architecture is designed to deal with the computationally intensive aspects of radio communication simulation in a distributed environment. Results are presented that show how the architecture scales on a shared memory multiprocessor SGI Origin with increasing problem size and available processors.

1 INTRODUCTION

With the complexity of current cellular communications systems and the time-to-market pressures on manufacturers, it is becoming increasingly important that the dynamics of a communications system are understood before committing to development and deployment. The means to this understanding is through detailed system simulations. Perhaps due to the way communication systems are structured, simulations tend to focus on certain aspects of the system (e.g., simulating the protocol stack or link level performance). There is, of course, a great deal to gain from these types of simulations in terms of understanding the processes involved and any inherent problems. Each of these components can be extremely complex in their own right; however, focusing on these smaller components of the system does not allow their interaction with the system to be investigated. What is really required is a much more detailed simulation which includes all of these smaller components. With a suitable means to evaluate the system, problems may be identified and investigated and solutions tested.

The scale of the simulation required for an in-depth knowledge of the system can sometimes appear daunting. In order to study dynamic problems such as call origination or handover, for instance, it is often necessary to simulate a densely populated virtual city containing buildings and streets, hundreds of basestation, and hundreds, if not thousands of mobile users. Simulation of the basestation, user, and infrastructure protocols must be performed to ensure system fidelity. For this virtual city, details of the RF propagation both inside and outside of buildings must be known in order to evaluate signal propagation for the communications equipment. Whenever one of the communications devices (basestation or user) changes a property that affects the RF environment (e.g., position or transmit power), the RF environment needs to be recalculated to make communications devices aware of the interference that they are receiving. These aspects of the simulation allow call statistics to be gathered and effects of protocol and signal processing algorithms to be evaluated.

One of the main problems associated with creating complex system simulations is clear: the large amount of time it takes to simulate a system. Clearly, there are a number of simulation approaches that can be used to attack this problem (e.g., sampled discrete time, discrete time with data flow, and conservative and optimistic discrete event). However, there is no escaping the fact that a considerable amount of processing is required. Whilst in principle this type of simulation may be possible on single processor architectures, the amount of time a simulation would take to complete would be unacceptable.

Accepting that system simulations of this scale are not practical on single processor architectures brings up the next question: how can we efficiently map the problem to a multiprocessor architecture to reduce the simulation time? It would be reasonable to state that a common method of performing simulations is to evaluate the outputs of system entities at discrete, but equally spaced, points in time, working through from the simulation start-time to the end-time. With this discrete time approach, it is often difficult to map the problem onto multiple processor architectures. When the problem can be mapped, it often requires a

shared memory architecture to efficiently access common data inherent in this type of simulation. Whilst this approach has been effective on this type of architecture, its effectiveness is limited on a cluster architecture. We require a more suitable approach for both multiprocessor shared memory and cluster architectures. This was the focus of our work.

We decided that the best approach to creating a system simulator was to make use of a third party simulation engine allowing the development team to concentrate on the main problem: development of a flexible architecture suitable for simulations of current and future technologies. Given the tight coupling between the RF environment and the communications entities in the simulation, we felt that a conservatively scheduled, parallel discrete event simulation engine was best suited to the task. After a review of possible options, the Dartmouth Scalable Simulation Framework (DaSSF) was chosen as the simulation engine. This paper presents the development of a suitable architecture based upon the DaSSF engine and the performance of the prototype system.

The organization of the paper is as follows. In Section 2 we present an overview of DaSSF, describing its features and general principles of operation. In Section 3 we describe the areas we have identified as the main bottlenecks in the system simulation and present a prototype architecture. In Section 4 we present a number of the key results showing simulation performance with various problem sizes and available processors. Conclusions are presented in Section 5.

2 OVERVIEW OF DASSF

DaSSF is an implementation of the Scalable Simulation Framework (SSF) as proposed and maintained by the SSF Research Network (SSFNET 2001) and sponsored by DARPA, the Institute for Security Technology Studies at Dartmouth, and the Renesys Corporation. It is a conservative, process-oriented, parallel, discrete event engine.

Simulations consist of a number of co-aligned entities grouped together into timelines. A timeline is considered to be a group of tightly coupled entities that are all dependent on one another. When running in a cluster or on a multiprocessor machine, timelines are spread across the available machines and/or processors.

The basic layout of a DaSSF simulation consists of a number of entities each containing processes and channels. The channels of the entities are connected together so that the entities can pass events back and forth amongst themselves. A breakdown and more detailed explanation of these components and their function is included below. A complete description of DaSSF may be found in the *Dartmouth Scalable Simulation Framework User Manual* (Liu and Nicol 2001).

2.1 Channels

A simulation channel provides a generic mechanism for delivering events from one entity to another. This is also the exclusive delivery mechanism for events between entities. There are two main types of channels: in and out.

Out-channels and in-channels are mapped together to create a network of connections. When an event is written to an out-channel it is delivered some non-negative time later on a corresponding in-channel. If an out-channel is mapped to more than one in-channel, then the transmitted event is delivered to all corresponding in-channels after applying all appropriate delay (see Section 2.4). The opposite it also true. Multiple out-channels can be mapped to the same in-channel.

While most channels are considered public, or external, an entity can also have a private channel, called an internal channel, that can only be accessed by the processes within the entity. Internal channels can be used to deliver events from one part of an entity to another.

Channels exist from the beginning of the simulation to the end. A channel cannot be deleted once the simulation has begun. While it is possible to *create* a new channel after the simulation has begun, the authors of DaSSF recommend against doing so due to the relatively high associated cost (Liu and Nicol 2001).

2.2 Processes

A process' responsibility is to react to triggered events and schedule future events to occur. Processes wait for events to arrive on one or many of the incoming channels belonging to the process' associated entity. When an event arrives, the process is woken up and given a list of all events that arrived on the channel of interest at the current simulation time. After processing is completed, the process returns to a wait state.

A process can schedule itself to be woken up at some discrete point in the simulation (at time 3.0), or after some amount of simulation time has elapsed (in 10 simulated time-units). Alternatively, a process can request to be woken up when either some amount of simulated time has elapsed or an event has arrived. The process will be woken up by whichever of the two conditions is satisfied first.

2.3 Entities

An entity is a simulated object such as a mobile telephone or basestation. As already described, a simulation is built out of entities and their connections to one another. The entities interact with one another through the processes and channels that they control. The processes use the entity's channels to send events to and receive events from other entities in the simulation.

2.4 Delay

There are three types of delay in DaSSF: channel, mapping, and per-write delay.

Channel delay (C) and mapping delay (M) are closely related as they both affect the delay on a given channel. Channel delay is delay assigned to a channel when it is created. Mapping delay is delay assigned when an out-channel is mapped to an in-channel and is specific to that mapping. When an event is written to an out-channel, the delay on that channel, calculated as C + M, is applied to the event and it is not delivered until that amount of simulation time has elapsed. For instance, if the current simulation time were 1.05 and an event were written to an out-channel with a channel delay of 0.25 and a mapping delay of 0.25, the event would not arrive at the destination in-channel until simulation time 1.55 (1.05 + C + M).

Channel and mapping delay apply to both internal channels (channels wholly contained within an entity) and external channels (channels between two entities), though there are some differences between how the delay types may be used with the different channels. External channels between non-aligned entities (entities in different timelines) cannot have both a channel and mapping delay of zero. One benefit of internal channels and channels between co-aligned entities, though, is that both the channel and mapping delay can be set to zero allowing an event to arrive at the same simulation time in which it was sent.

As noted in Section 2.1, it is possible to map a single out-channel to multiple in-channels. Different amounts of mapping delay can be assigned to each of these links causing an event to be delivered to the corresponding in-channels at different simulation times.

Event delivery can be further controlled through per-write delay. Per-write delay is delay applied when an event is written to a simulation out-channel. This can be used to schedule an event for arrival some time in the distant future and is useful for simulating state-dependent delays. Revisiting our previous example, if the current simulation time is 1.05 and the total delay on a channel (C + M) is set to 0.5 we can force an event to not be delivered to the corresponding in-channel until simulation time 3.0 by using a per-write delay value of 1.45. The channel, mapping, and per-write delay values are added together to determine the total event delay, which, in this case, is 1.95 simulation time units (0.25 + 0.25 + 1.45).

3 ARCHITECTURE COMPONENTS AND DESCRIPTION

Our architecture consists of three basic components: communications equipment, routers, and RF. These components work together to allow for the simulation of both simple and complex radio networks. Because DaSSF requires that all events being delivered from one entity to another be transmitted via a channel, we had to develop an efficient, yet versatile, method to allow for all parts of the simulation to be interconnected without having to create a connection pair (one in-channel, one out-channel) between each entity.

By connecting every piece of the simulator to an event router, and then routers to routers, we are able to create a virtual network over which events can flow. The power of a virtual network is that fewer channels are required in order to accomplish the same level of connection. The simulation network will be broken into at least two logical components: RF related communications will take place on the RF component of the simulation network while basestation to switch type communications will take place on the infrastructure component. The details of how these connections are established are described in the following sections.

3.1 Communication Equipment

Communication equipment (CE) includes any piece of radio equipment that has the ability to receive and transmit to one or many listeners such as users and basestations.

A CE is capable of communicating with many other CE simultaneously. A CE needs only to set the destination field of an event to have it delivered to a specific CE in the simulation network. The CE need not worry about how the event reaches its destination as that is taken care of by the simulation network routers. In order to deliver the same event to more than one CE, the destination field of the event should include all of the CE to which the event should be delivered. Alternatively, a CE can send events to more than one CE within the same simulation tick (see Section 3.2.3). There is no limit to the number of events that may be sent out at the same simulation time.

This is equally important to both users and basestations. Because a basestation is listening and talking to many users at the same time, it needs to be able to send out multiple events all within the same simulation time.

3.2 Routers

The simulation router is the backbone of all simulation communications. As noted above, a simulation using the proposed architecture is built out of CE, routers, and RF. It is the router's job to connect many entities together. The groups of entities form a logical simulation network. While the connected entities will most likely be groups of CE, a router's functionality is not limited to CE event routing. A router can be used to route events between *any* two connected entities in the simulation network.

3.2.1 The Simulation Network

The network, from a router's perspective, is broken into two conceptual pieces: local and remote entities. Local entities are those entities that are directly connected to the router. Remote entities are those entities that must be contacted through communications to some other router. Routers hold routing tables telling them how to direct an event from its originator to the destination.

Router-to-router communications help overcome an inherent problem in parallel simulations: cross-processor talk. Because, in general, there is a performance hit taken every time a piece of memory is accessed that belongs to a processor other than the one on which the current thread is operating, the number of cross-processor memory accesses needs to be kept to a minimum. Minimizing cross-processor communications is especially important when dealing with clusters. Cross-processor memory accesses are not local to the current processor on which the thread is executing, and not likely to be local to the *machine* on which the thread is executing (except in cases of clusters of multiprocessor machines, and even then the chances are not great).

By breaking the communications down to local and remote, we can optimize how events are routed to avoid unnecessary cross-processor memory accesses. If a single router were used (as was first attempted) almost every entity-to-entity event would be traveling past the bounds of a processor (see Figure 1). This produces extremely undesirable performance. By creating at least one router for each processor in use and grouping all entities connected to that router into the same timeline, the number of cross-processor events can be kept to a minimum.

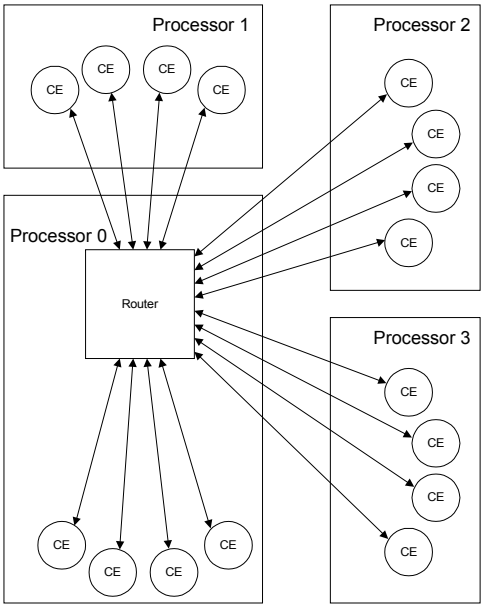

Figure 1: Single Router System

While routers must be concerned with whether an entity is local or remote, entities connected to the router never need to worry about such details. When an entity sends an event, that event is never sent directly to the destination. Instead, a router intercepts the event and determines how to deliver the event to its intended destination.

In addition, local events are never routed further than the source entity's router (see Figure 2). When the destination of an event is some entity with which a router is not in control, the source entity's router looks in its routing table to determine which router is in charge of that entity. The source entity's router then hands the event over to the router in charge of the event's destination entity (see Figure 3).

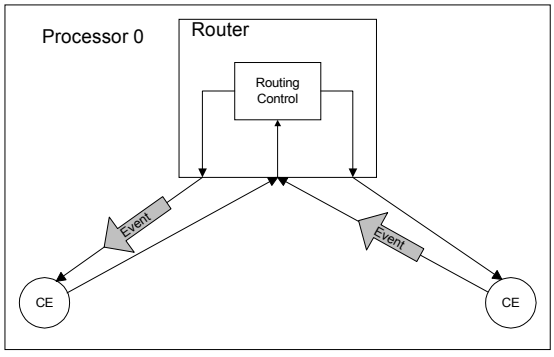

Figure 2: Local-to-Local Event Routing

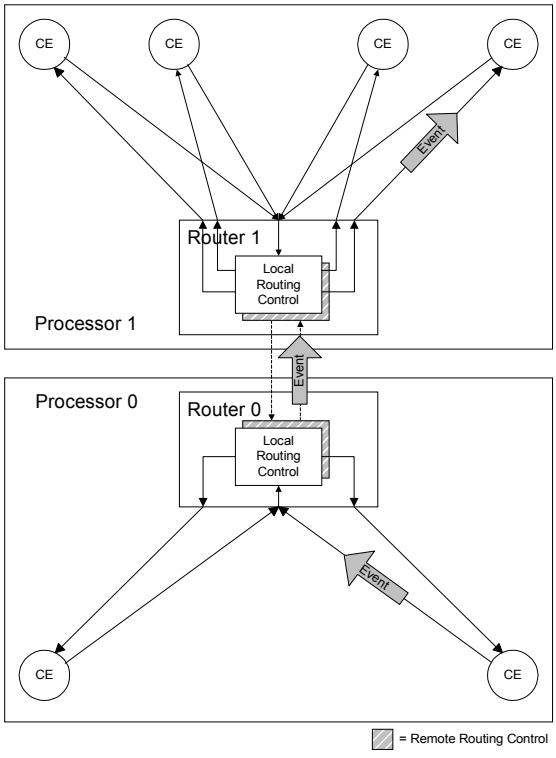

Figure 3: Local to Remote Event Routing

3.2.2 Routing Tables

A router contains two sets of connections: those to the entities being routed and those to other routers. All connected entities share one in-channel while all connected routers share another separate in-channel (See Figure 3). While all connected entities may share a common in-channel, the router must maintain a separate and unique out-channel for each router and each routed entity.

If a router used a common out-channel to communicate with all of its connected entities, every entity connected to that router would receive every other entity's events. In general, it is much more important for a router to maintain separate connections to other routers because those channels cross-processor boundaries. If all routers were connected to the same out-channel, the result would be that routers would work in a broadcast mode creating a disproportionate number of cross-processor events. However, if we maintain separate out-channels for each router and routed entity, we can make the most efficient use of our transfers (see Figure 3).

This introduces the need for routing tables. A router holds two routing tables: one for local entities and another for remote routers. By using the local routing table, a router can determine the link to which every local entity is connected. The remote routing table lists all of the entities that are connected to each remote router. Each remote router's entity list is held in the remote routing table. If an event's target is located on a remote router, the local router passes the event to that remote router which then handles delivery. While dedicated channels help to reduce the number of cross-processor events, there is an inevitable increase in cross-processor messaging as the simulation progresses. At the beginning of every simulation, the architecture attempts to place users on the same router as the basestation with which they are communicating. As priority is placed on balancing the number of entities on each router, not every user will be able to be placed local to its basestation. As users move and their signal strength rises and falls, they will perform handoffs to other basestations. The longer a user remains in a call in the simulation, the less likely it is for that user to be local to the basestation with which it is communicating.

3.2.3 End-to-End Event Delivery

The architecture utilizes two types of events: simulation events and framework events. A simulation event is anything that pertains to the system that we are simulating. For instance, a power change request in a CDMA system would be a simulation event. Framework events are events that do not directly pertain to the system being simulated, but are necessary to communicate some piece of information or state change to an entity in the simulation. For instance, a framework event could be sent to an entity in charge of movement when a user in the system changes its position.

The architecture handles these two types of events in different ways. Framework events are allowed to travel across the simulation network as quickly as possible without introducing any delay in addition to the channel and mapping delay. Simulation events, though, are guaranteed to take a constant amount of simulation time to travel between their source and destination. This enforced end-to-end delay is referred to as a **tick.**

Because the architecture uses routed event delivery, events sent between entities on the same router flow more quickly than events sent to entities on remote routers. The tick ensures that this does not occur to simulation events. When a router receives a simulation event destined for a local entity, some amount of per-write delay smaller than the tick is applied when the event is sent to its destination. The amount of delay applied is calculated so that the simulation event will be delivered exactly one tick after it was sent. This is performed for both simulation events received from local entities and those received over a routed link and is possible only because the router is aware of how much delay has been assigned to each channel between the source and the destination.

3.3 RF

It is important to distribute RF calculations across available processors as much as possible as they have been identified as the single most time consuming section of previous in-house simulators.

As noted in Section 1, in order to provide an accurate picture of the RF environment, interference must be calculated each time a CE moves or changes power. Proper interference calculation requires determining the power for every interferer in the system as received by every other CE receiving on the same carrier. These power calculations involve a number of non-trivial math functions such as logarithms, inverse logs, roots, and powers. While a number of steps have been taken to speed up these calculations (such as using tables rather than calculations, sacrificing precision for speed), their sheer volume requires an architecture sensitive to their needs.

For instance, each user in the simulation of a single carrier must calculate their receive power of every basestation in the system each time a basestation changes its power. Likewise, each basestation in the system much recalculate its receive power each time a user moves or changes its power. In a CDMA system simulation, these power changes could occur as often as every 1.25 ms, thus requiring the complete reevaluation of the RF every 1.25 simulated ms. By uniformly dividing the users up amongst the routers in the system and making each router responsible for the RF calculations for all of its local CE, we can easily distribute the computational work amongst all available processors.

4 ARCHITECTURE PERFORMANCE

In order to effectively evaluate the combined performance of the DaSSF engine and our architecture, we require a simulation scenario that is representative of the final application. We have identified and discussed some of the potential bottlenecks, such as RF interference power calculation, cross-processor events, and memory accesses. The simulation used to evaluate performance therefore includes as many of these aspects as possible.

We have created a cellular wireless simulation for the performance evaluation. Basestations and users are assigned connections between each other. Similar to a real cellular system, basestations will be connected to multiple users. However, unlike real systems the basestation-user connections remain constant throughout the simulation. To allow us to investigate the effect of cross-timeline and cross-processor events and memory accesses, the basestation-user connections can be created with an arbitrary number of cross-timeline connections (i.e., a user communicating with a basestation that is not connected to the same router). As each router is mapped to its own timeline, we make the cross-timeline connections by spreading the desired number of cross-timeline users as evenly as possible across the routers. Each of these spread users is mapped to a basestation that is least likely to be resident on the same processor. However, because there may be more than one timeline on each processor, there may be fewer cross-processor connections than cross-timeline connections.

The basestation-user communication models a simple power control loop. It is the aim of the basestations and users to receive the signal being transmitted to them at a predetermined target power. To achieve this, the receiving device measures the received signal power, computes the difference between it and its target power, and transmits a command back to the sender instructing them (if required) to modify their transmit power.

After each message is received by the local router (in the form of an event), a signal-to-noise ratio (SNR) calculation is simulated. When the received message is from a user, this is accomplished by performing one log and one inverse log for each basestation in the system. When the received message is from a basestation, the SNR calculation is simulated by performing one log and one inverse log for each user in the system. When required, we can disable the numerical loading provided by the power and interference calculations. The simulations can be run on an arbitrary number of processors with an arbitrary number of routers, basestation and users.

In the following results, we represent the performance of the system by making use of an efficiency factor, or utilization metric. The efficiency of the simulation (η) is computed based upon the total CPU-seconds used by the simulation (i.e., the sum of all CPU-seconds used on all processors in the simulation) relative to a reference CPU-seconds measure. The efficiency factor is given by:

$$\eta = \frac{S_{ref}}{S_x} \tag{1}$$

where S_{ref} is the reference CPU-seconds, and S_x is the total number of CPU-seconds used in the simulation of interest.

A performance metric used by a number of authors (e.g., Fujimoto 2000) is the speedup (K) in the simulation using N processors when compared to a single processor. Efficiency, η, is related to K (in terms of CPU-seconds) as follows:

$$K = \eta N \tag{2}$$

The following sections use Equation (1) to present key results demonstrating the performance of the DaSSF engine and the proposed architecture.

4.1 Performance with Numerical Loading Disabled

To investigate the effect of cross-timeline and cross-processor events, we ran a set of simulations, on various numbers of processors, with the numerical loading (interference calculations) disabled. In the absence of this numeric loading, the basestations and users simply reply to each other's messages with no calculations involved. The efficiency computed using Equation (1) gives a measure of the effect of the cross-timeline and cross-processor events. Figure 4 and Figure 5 show the simulation efficiency results for 128 and 512 users respectively. These figures show results for scenarios with 32 basestations and 32 routers for various values of user spreading.

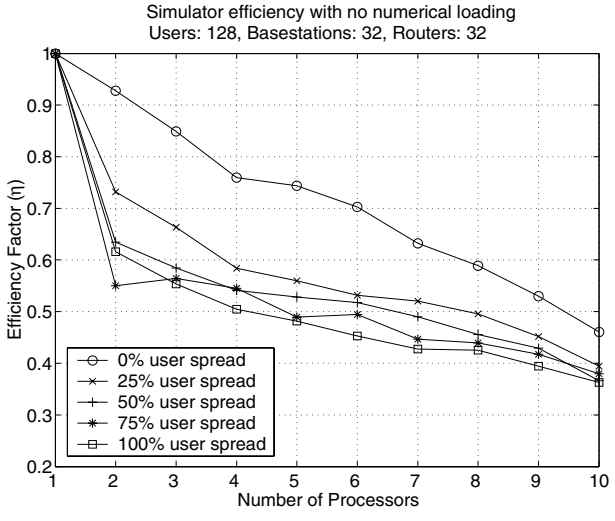

Figure 4: Simulation Efficiency as a Function of Number of Processors. 128 Users, Numerical Loading Disabled

Intuitively, simulations with a greater percentage of user spreading should be less efficient due to the increase in cross-timeline events. Inspection of Figure 4 and Figure 5 confirms this belief; a greater degree of user spreading leads to a decrease in efficiency.

Inspection of the figures shows that increasing the number of processors also decreases the efficiency of the simulation. This loss in efficiency is due partly to the DaSSF engine and partly to the architectural design. We can see from both Figure 4 and 5, the efficiency of the simulation for 0% user spreading also decreases with an increasing number of processors. Clearly, with no user spread there are no cross-timeline or cross-processor events. The drop in efficiency is due only to the engine. It is interesting to note (from both Figure 4 and 5) that the rate of decrease of efficiency is lower when user spreading is introduced. With user spreading introduced, another effect is present in the system. The number of cross-processor events increases as more processors are used. This is because timelines that had previously been resident on the same processor, become resident on different processors.

A closer inspection of Figure 4 and Figure 5 reveals that, in general, for the same user spread, a greater number of users in the system leads to a greater efficiency.

Whilst allowing an interesting insight into the simulation process, the no-load simulations are clearly not representative of the final application; the complete wireless system simulator. The next section will present results with the interference calculations enabled.

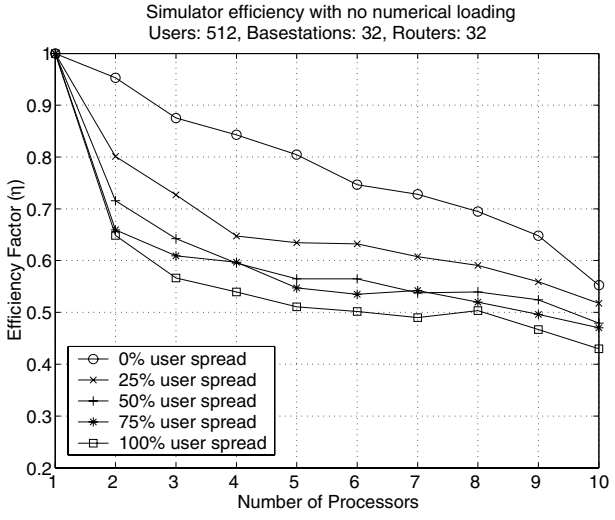

Figure 5: Simulation Efficiency as a Function of Number of Processors. 512 Users, Numerical Loading Disabled

4.2 Performance with Numerical Loading Enabled

We have already discussed the significance of the RF calculations that must be performed by the simulator and the potential performance bottleneck they represent. In the results presented in the following sections, the interference calculations within the simulation aim to be representative both in the type and number of calculations performed in the final application. It is of particular importance that the reader be aware that regardless of the number of processors on which the simulation is executed, the number of calculations in the system remains constant. It is the same dimension problem to be solved by the multiprocessor simulation as for the single processor case.

Figure 6 and Figure 7 show the simulation efficiency results for 128 and 512 users respectively. These figures show results for scenarios with 32 basestations and 32 routers for various values of user spreading. The initial observation that must be made is the efficiencies of both of the 128 and 512 user cases is higher than their corresponding cases with numeric loading disabled. The simulation themselves took longer to run, but because of the numeric loading, relatively, less time was spent with cross-timeline events.

The results of Figure 6 and Figure 7 again show that, in general, the efficiency drops off with increasing user spread and increasing number of processors. We can make a similar observation of the comparison of Figure 6 and Figure 7 as we did between Figure 4 and Figure 5. The 512 user simulation is more efficient than the 128 user case, although the difference appears more significant in this case.

We have seen in this and the previous section that increasing the number of users in the system leads to an improvement in simulator efficiency. This however must not be confused with a reduction in the time taken to simulate. More users will always take a longer time to simulate. However, the simulation will make better use of the available processors. In the next section we take a more detailed look at the effect of user spread on simulation efficiency.

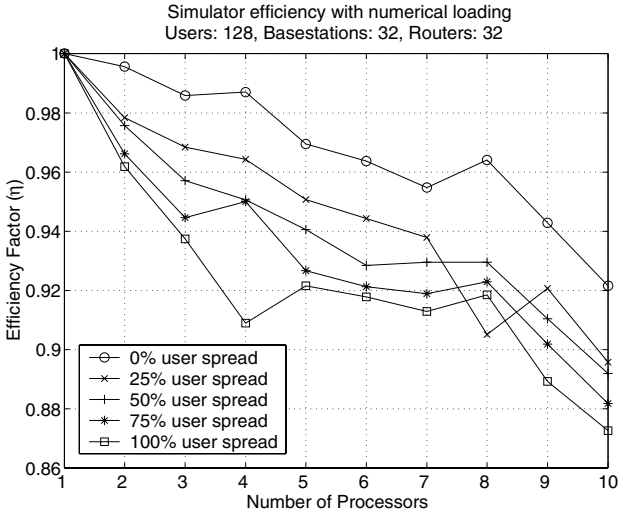

Figure 6: Simulation Efficiency as a Function of Number of Processors. 128 Users, Numerical Loading Enabled

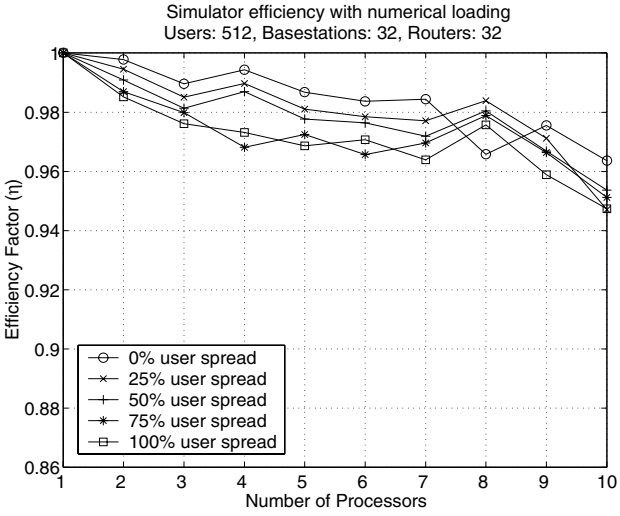

Figure 7: Simulation Efficiency as a Function of Number of Processors. 512 Users, Numerical Loading Enabled

4.3 Performance as a Function of User Spread

We have already seen that increasing user spread increases the number of cross-timeline events (and potentially the number of cross-processor events) thus reducing simulation efficiency. As we have previously discussed, in a wireless system simulator, the user spread will continue to increase through the simulation as users move and interact with basestations. In this section we examine this effect.

The efficiency results are computed using Equation (1) for a range of values of percentage user spread. S_{ref} is the number of CPU-seconds required to execute the simulation with 0% user spread for a particular number of users in the system. S_x is the number of CPU-seconds required to execut the simulation with x% user spread for the same number of users in the system.

Figure 8 and Figure 9 show simulation efficiency results as a function of percent-user-spread for simulation scenarios using 4 and 8 processors respectively. The figures show results for scenarios with 32 basestations and 32 routers for various values of total number of users in the system.

Figure 8 (4 processors) confirms the pattern exhibited in Figure 4 through Figure 7; namely that for a particular percentage of spread users in the simulation, the simulation with the greatest user population will be the most efficient (with the exception of 75% spread, 512 users). Also, that as the user spread increases, (for any particular number of total users) the efficiency decreases as expected.

Figure 9 (8 processors), also shows that for a particular percentage of spread users in the simulation, the simulation with the greatest user population will be more efficient (again with a few exceptions below 25% spread). However, of particular interest is the efficiency of the simulator with 512 users. Here, for user spreads of 25% and above, the simulator has an efficiency greater than one. The reasons for this are as yet unclear although they are most likely related to a thread-to-processor ratio greater than one resulting from multiple routers-per-processor.

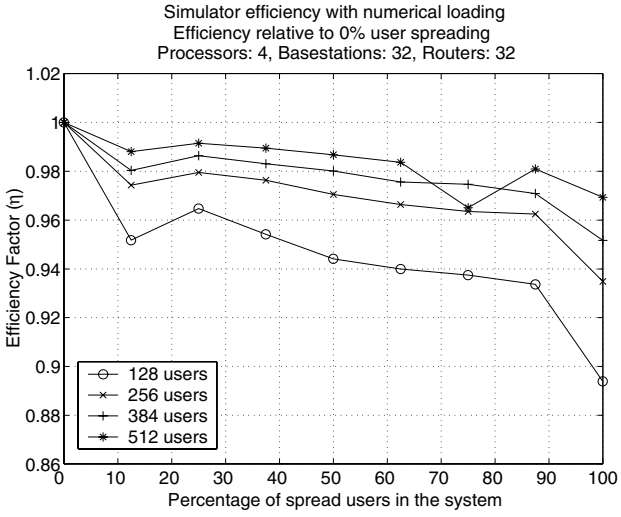

Figure 8: Simulation Efficiency as a Function of User Spread. 4 Processors, Numerical Loading Enabled

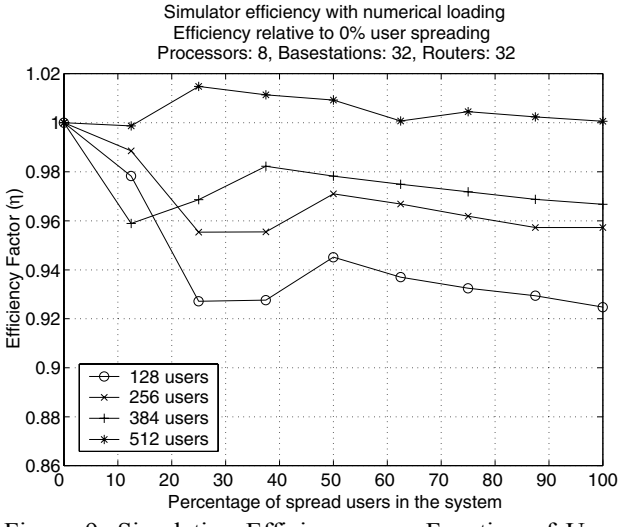

Figure 9: Simulation Efficiency as a Function of User Spread. 8 Processors, Numerical Loading Enabled

4.4 Performance as a Function of Routers per Processor

As each router represents a timeline and hence a separate thread, there is a potential benefit in having multiple routers per processor. A single router-per-processor would give little opportunity for the operating system to make use of context switching during I/O waits. In the results we have presented so far, the total number of routers in the system has been fixed at 32. This is because a fixed number of routers is required for deterministic results between

runs of different number of processors. However, for completeness, it is worth investigating the affect on efficiency of the number of routers per processor.

Figure 10 shows the simulator efficiency as a function the number of routers-per-processor. The figure shows the results for scenarios with 512 users with 50% user spread, 32 basestations for 4 and 8 processors. The figure shows that there is no significant loss in efficiency with an increasing number of routers per processor, indeed, with a few exceptions, there is an efficiency gain. Clearly, Figure 10 only shows the results for two processors and one user count and spread combination, however, the fact that the efficiency does not vary significantly over the range of routers per processor indicates that choosing a large but constant number of routers for determinism reasons will not unduly harm the simulation performance.

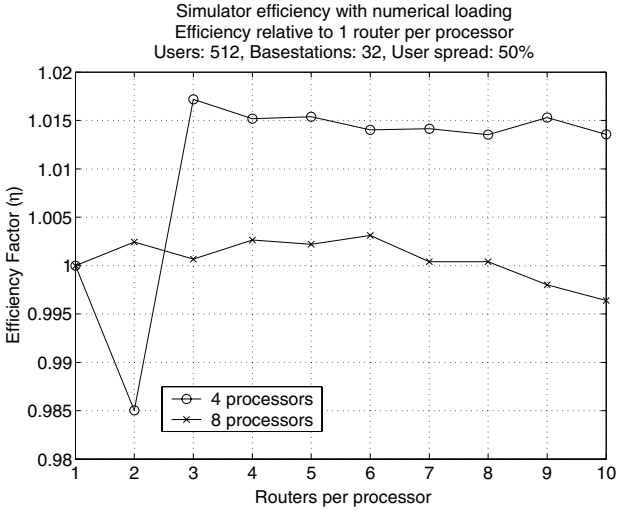

Figure 10: Simulation Efficiency as a Function of Routers per Processor. 512 users, Numerical Loading Enabled

5 CONCLUSIONS

We have presented an architecture suitable as a basis for a wireless system simulator and presented performance results to support the design.

To create a wireless simulator that scales across processors efficiently there are two key factors that must be considered before, during, and after the design: distributing the processing and reducing the cross-processor memory accesses. We have shown that utilizing a network approach to divide the problem and limit the complexity proves to be an effective method of accommodating both these factors.

Our prototype implementation of the architecture has proven to scale efficiently over both increased processor availability as well as increased workload. While the performance of the architecture is influenced by the number of cross-timeline and cross-processor memory accesses, the overall performance is still excellent. We have shown that a router-to-processor ratio greater than one has no adverse affect on the simulator performance. This is an important result given that for deterministic results across different number of processors, a constant number of routers in the system is required.

It is our desire to begin evaluation of this architecture on clusters of workstations. We believe that it will continue to exhibit favorable scaling behaviors, though they will most certainly not be of the same scale of those shown in this paper due to the lower data transfer rates between cluster nodes.

REFERENCES

Fujimoto, R. M. 2000. Parallel and Distributed Simulation Systems. John Wiley & Sons.

Liu, J. and D. M. Nicol. 2001. Dartmouth Scalable Simulation Framework Version 3.1 Users Manual. Department of Computer Science, Dartmouth College, Hanover, New Hampshire. Available online via <http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/papers/dassf-manual-3.1.ps> [accessed July 14, 2001].

SSFNet.org. 2001. SSF Research Network website. <http://www.ssfnet.org> [accessed July 16, 2001]

AUTHOR BIOGRAPHIES

TREFOR J. DELVE is a Communications Research Engineer with Motorola Labs. He received his B.Eng (Honors) degree from the University of Birmingham, U.K., in 1991. He has worked as a communications engineer for The MathWorks, a systems engineer for NEC and a research associate working on underwater communications for the Ministry of Defence, U.K. His research interests include channel coding and propagation modeling. His email address is <Trefor.Delve@motorola.com>.

NATHAN J. SMITH is a Software Research Engineer with Motorola Labs. He received his B.S. from Illinois State University in 2000. He has worked as a lead applications developer for Illinois State University and an embedded systems developer for Motorola. His email address is <Nathan.Smith@motorola.com>.