Conservative Synchronization of Large-Scale Network Simulations

Alfred Park Richard M. Fujimoto Kalyan S. Perumalla
College of Computing, Georgia Institute of Technology
Atlanta, Georgia, USA 30332-0280
{park,fujimoto,kalyan}@cc.gatech.edu

Abstract

Parallel discrete event simulation techniques have enabled the realization of large-scale models of communication networks containing millions of end hosts and routers. However, the performance of these parallel simulators could be severely degraded if proper synchronization algorithms are not utilized. In this paper, we compare the performance and scalability of synchronous and asynchronous algorithms for conservative parallel network simulation. We develop an analytical model to evaluate the efficiency and scalability of certain variations of the well-known null message algorithm, and present experimental data to verify the accuracy of this model. This analysis and initial performance measurements on parallel machines containing hundreds of processors suggest that for scenarios simulating scaled network models with constant number of input and output channels per logical process, an optimized null message algorithm offers better scalability than efficient global reduction based synchronous protocols.

1. Introduction

In conservative parallel discrete event simulation (PDES), an event cannot be processed unless it is safe to do so. Since all logical processes (LPs) do not have a consistent view of the state of the entire system, LPs must exchange information to determine when events are safe to process. This process of synchronizing LPs has been the subject of much past research [1].

Conservative synchronization algorithms can be broadly classified as asynchronous or synchronous. Asynchronous algorithms do not require global synchronizations. A well-known example is the null message algorithm for deadlock avoidance originally developed independently by Chandy and Misra [2] and Bryant [3]. Variants to the original Chandy-Misra-Bryant (CMB) null message algorithm to improve efficiency have been devised and evaluated [4-7]. On the other hand, synchronous algorithms use global synchronization and reduction computations to compute a Lower Bound on Timestamp (LBTS) of future messages that might be received by each LP, in order to determine when events are safe to process. A well-known example is the YAWNS protocol described in [8]. Other algorithms use techniques such as deadlock detection and recovery [9] and simulation time windows [10].

Although previous studies have evaluated the performance of conservative synchronization algorithms for inefficiencies and overhead (e.g., see [11-13]), the scale of these studies have been, for the most part, limited to modest sized configurations, with most using fewer than 100 processors. As the number of LPs that participate in a parallel and distributed simulation increase, performance conclusions based on small-scale simulation studies may not apply to a large-scale context and the appropriateness of a particular synchronization method can shift from one algorithm to another. Here, we address scalability concerns in order to compare an asynchronous CMB null message algorithm with a synchronous barrier algorithm for large-scale network simulations.

The paper is organized as follows. Related work is first discussed in the next section. A lazy null message algorithm and an associated analytical model are presented in section 3. Section 4 describes the simulation scenarios used for the experimental study, and presents comparison between analytical model predictions and measurements. Section 5 presents empirical data comparing the performance of CMB and global reduction algorithms for regular and irregular networks. Future work and conclusions follow in sections 6 and 7, respectively.

2. Related Work

While there has been much research comparing optimistic and conservative synchronization protocols, there has been comparatively little work devoted to comparing the performance of asynchronous and synchronous conservative synchronization algorithms for large numbers of processors. Bagrodia and Takai provide extensive performance comparisons of synchronous and asynchronous synchronization algorithms, but do not consider large numbers of processors [4]. Nicol describes scalability through analysis of model characteristics, partitioning, workload balance, and model complexity in a broad PDES context [14]. However, our work specifically analyzes the detailed costs associated with the operation of a lazy null message algorithm. Further, we show that better scalability can be achieved using CMB, compared to global reduction algorithms, in scaled network models with constant fan-in/fan-out. This is performed with an emphasis on quantitative results supporting our findings.

Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)
1087-4097/04 $20.00 © 2004 **IEEE**

Nicol and Liu [15] combine asynchronous channel scanning and synchronous global barrier algorithms with a good amount of experimental data examining fan-in/fan-out and lookahead up to 8 processors. We discuss performance implications on larger configurations, specifically comparing lazy null message-based and global reduction-based algorithms. These algorithms are applied to regular and irregular network models with varying load distributions and asymmetric lookahead values, on up to 128 processors. Nicol also presents models to provide a lower-bound on the performance of a synchronous global reduction algorithm that can apply to large numbers of LPs [16, 17]. Our analytic model differs from this work in its focus on scalability concerns for the CMB algorithm. While analyzers described in [18] illustrate speedup differences between synchronous and asynchronous algorithms, our focus lies in identifying detailed CMB performance characteristics, such as null message send frequency and overhead, and applied to analytical models for large-scale network simulation. Synchronization challenges are described by Naroska and Schwiegelshohn [19] for large number of processes. While their work focuses on VLSI design, we focus on network simulations. The work presented in [20] contains models that exclude lookahead values [21]. However, their approach requires extensive knowledge of various metrics and has no means to measure the effect of increasing scale.

3. Analytical Models for CMB

The impetus for utilizing an asynchronous rather than a synchronous algorithm is to avoid the potentially large amount of overhead in interrupting the simulation, calculating and distributing new LBTS values, and restarting the LPs. Synchronization based on the CMB algorithm depends upon null messages exchanged only between neighboring processors. The frequency and time stamp of these null messages dictate the behavior and performance of the simulation; this simple observation will drive the formulation of the analytical CMB model. We will iteratively build this model. The end result will be a model for the CMB null message algorithms that can approximate simulator performance in structured large-scale network simulations. In the following we assume the parallel simulator consists of a collection of LPs that communicate by exchanging messages.

3.1. Effect of Null Messages on Simulation Behavior

Lookahead is one of the most important factors to achieving good performance in conservative PDES. Each LP is allowed to process events without re-synchronizing with other LPs up to its LBTS value. In general, conservative PDES performs well when the difference between successive LBTS values is large relative to the simulation time between successive events, because each LP can process many events before it must re-synchronize with other processors.

In a traditional CMB algorithm, after an event is processed, null messages are sent to every output channel to neighboring LPs. It is well known that overly eager null message transmission schemes produce an excessive number of null messages, leading to other algorithms that improve CMB performance by sending null messages less often. However, a sufficient number of null messages must be sent in order to avoid deadlock. The frequency of null message sends can have a large effect on performance. As the number of null messages increases, the overhead to synchronize increases proportionately, and less useful work is accomplished each second of wallclock time. Lookahead, null message frequency, time advance conditions, and remote communication are all contributing factors in the analytical model characterizing the performance of algorithms based on null messages.

Here, we relax the constraints assumed in the original null message algorithm. Specifically, a LP need *not* send messages in time stamp order. We do assume that the communication channel delivers messages in the same order in which they were sent. These assumptions imply that all simulation time information for computing new LBTS values resides in the time stamps of null messages.

3.2. A Lazy CMB Null Message Algorithm

This variant of the CMB algorithm, which is traditionally referred to as *lazy CMB*, attempts to minimize the number of null messages by only sending them when absolutely necessary. Specifically, null messages are only sent when the LP reaches the end of its safe processing time, i.e., only when the LP must block. Failing to send null messages while in this blocked state could lead to deadlock situations.

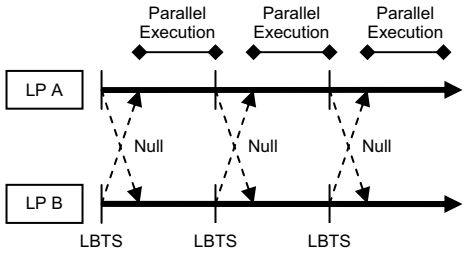

Figure 1. Execution Timeline of a Lazy CMB Algorithm

Figure 1 shows the progression of the simulation using the lazy CMB algorithm where each LP has the same lookahead. Note that "LBTS" in the figure represents when a new LBTS value is computed for time management. This lazy null message execution resembles that of a time-stepped simulation. With a minimal number of null messages exchanged between LPs, this scheme appears to be an efficient synchronization protocol. One can calculate the number of null messages N_Φ sent as:

$$N_{\Phi} = m \times n \times \left(\frac{s}{lookahead} \right) \qquad (1)$$

where m is the number of LPs, n is the number of output channels per LP, and s is the length of the run in simulation time. The number of execution windows (s divided by *lookahead*), gives the number of rounds or time-steps for which null messages are sent. Multiplying this result by the number of output channels yields the number of null messages sent each round. The product of this result and the total number of LPs gives the total number of null messages exchanged during the entire simulation. It is important to emphasize that formula 1 pertains to a CMB algorithm applied to a regular, structured network model. In other words, the lookahead of each LP is assumed to be the same.

In our sample simulation, let $m = 8$, $n = 2$, $s = 25$, and *lookahead* = 0.2 seconds. By formula 1, the total number of null messages would be 2,000. We can modify equation 1 to approximate the number of synchronization messages (N_{Ψ}) sent when a butterfly barrier is used (assuming that m is a multiple of 2):

$$N_{\Psi} = m \times \log_2 m \times \left(\frac{s}{lookahead} \right) \qquad (2)$$

The same simulation under a synchronous algorithm using global reductions through a butterfly barrier mechanism results in 3,000 synchronization messages from equation 2. More generally, if the number of output channels does not increase with the number of LPs, the number of null messages in CMB increases linearly with the number of LPs, yielding lower overhead than the approach using reductions.

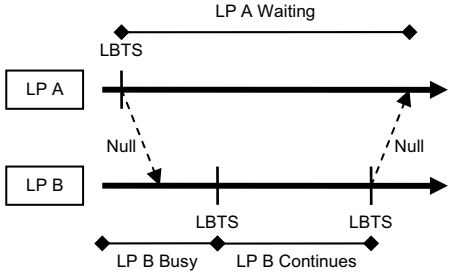

Figure 2. Blocking in a Lazy CMB Algorithm

It is important to note, however, that minimizing the number of null messages does not necessarily maximize performance. In particular, a modest load imbalance may lead to large waiting times that could be avoided by sending additional null messages. To see this, consider the example in figure 2, showing two LPs A & B with output channels between them. Here, LP A determines that it can no longer process events safely. LP A sends a null message to LP B and blocks. However, LP B is busy processing events within its own LBTS limit. Further, when LP B completes processing these events, it can process the null message from LP A, compute a new LBTS value, and begin processing events based on this new value, all *without* sending a null message to LP A. Because no new null message is sent to LP A, LP A will remain blocked. A null message will be sent to LP A only after LP B has finished processing these new events and is forced to block. Moreover, the delay to transmit this null message from LP A to LP B further increases the amount of time LP A remains blocked.

A solution to this problem is to have each LP send null messages more frequently. One must generate more null messages, but not so many that the system is burdened with an excessive amount. This is discussed next.

3.3. An Optimized Lazy Null Message Algorithm

One approach to generating additional null messages is to designate that each LP should send null messages every f units of simulation time advance, where f is less than the LP's lookahead (continuing to assume, for the moment, that all links have the same lookahead; we will relax this assumption later). By increasing the frequency of null messages sent to neighbors, potentially long blocking times can be avoided.

A second optimization to this approach is to eliminate null messages that carry no new useful information. For example, if two successive null messages carry the same time stamp it is clear there is no reason to send the second. Thus, we can cancel (suppress) any superfluous null messages that convey no new information, by not sending those messages at all.

To include these two changes, we can modify equation 1 to yield:

$$N_{\Phi} = m \times n \times \left(\frac{s}{f} \right) (1 - c) \qquad (3)$$

where c is the proportion of null messages cancelled and f is the frequency at which null messages are sent. Here, f is used in place of the lookahead value in equation 1. Due to the non-deterministic nature of the modified lazy null message algorithm, equation 3 provides only a lower-bound estimate on the number of null messages sent if lookahead is used for f. The quantity c cannot equal 1 nor can f equal 0 (which means all null messages are cancelled or no null messages are sent, respectively). If either of these conditions were true, then the deadlock avoidance guarantee of the CMB algorithm would be violated.

We now turn to the more general case of non-uniform lookahead on output channels. It is useful to construct an analytical model for which the simulation models are irregular. Here, we assume a lookahead value is assigned to each link.

In the case of different lookahead values for different LPs, formula 3 can be modified to yield:

$$N_{\Phi} = \sum_{j=1}^{m} \sum_{k=1}^{n} \frac{s}{f_{jk}} \left(1 - c_{jk}\right) \qquad (4)$$

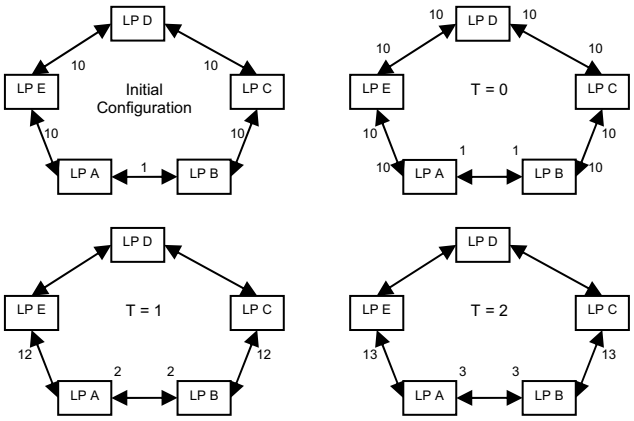

Figure 3. Example Illustrating the Operation of Null Message Exchange with Non-Uniform Lookahead

where f_{jk} represents either the minimum lookahead on the outgoing channel to LP *jk* or the frequency of null message sends. From a LP *j*, if there are multiple outgoing links to a neighboring LP *x*, they are consolidated into a single output channel with the lookahead assigned as the minimum of all outgoing links to *x*. In figure 3, a sample LP topology is constructed where the lookahead on all output links between LPs is 10, with the exception of the link between LP A and LP B. Although the lookahead is 10 between LPs A & E and LPs B & C, we can see that both LP C and LP E receive null message updates from LPs A & B at 1 time unit intervals. Therefore, the "localized effect" of non-uniform lookahead must be carefully considered before applying an estimate using equation 4. Equation 4 is applicable even in the case where all lookahead values are the same among all output channels, in which case the equation would then be equivalent to equation 3.

Equations 1-4 include lookahead and null message frequency to give an approximation on the number of null messages sent during the course of the simulation. As mentioned earlier, the number of null messages is useful, but it does not in itself offer an estimate on the efficiency of the synchronization algorithm. In addition to blocking, the type of communication (e.g. shared memory vs. Ethernet) and other related factors provide a better approximation on the overhead of a null message algorithm.

$$\alpha_{\Phi} = \frac{1}{a}\sum_{i=1}^{a} \ln\left[p_i w_i \times \frac{1}{m}\left(\sum_{j=1}^{m}\sum_{k=1}^{n} \frac{s}{f_{jk}}\left(1 - c_{jk}\right)\right)\right] \quad (5)$$

Equation 5 represents an overhead index for the lazy null message algorithm. Here *a* represents the number of different types of remote communication used in the simulation, *p* is the percentage and *w* is the weight given to a specific type of remote communication. The quantity *w* is a normalization factor for different communication media. For instance, if we had remote communication over both shared memory (w_1) and Ethernet (w_2), a possible normalization would be w_1=1 and w_2=10 to represent shared memory being faster by an order of magnitude compared to Ethernet. The natural logarithm is applied to represent the index as a monotonically increasing function of the product of the types of remote communication used and the average amount of null messages sent and received per LP.

Utilizing these equations, one could compute not only the number of null messages that would be sent over all LPs, but also the relative messaging overhead incurred with a particular null message algorithm or simulation scenario. This can be particularly useful when attempting to predict performance of a new scenario against an established baseline (e.g. scaled experiments).

3.4. Synchronizing Large-Scale Simulations

Equation 5 predicts that the overhead associated with these variations of the null message algorithm will remain relatively constant as a simulation scales, assuming the number of output channels for each LP does not increase with the number of LPs, and all other factors (such as relative amount of remote communication and lookahead) remain unchanged. On the other hand, synchronous time management algorithms have larger overheads as the simulation scales in the number of processors.

Notice that the only change from equation 1 to equation 2 is the substitution of $\log_2 m$ for *n*. For a synchronous time management algorithm, the number of synchronization messages increases at a rate of $m\log_2 m$, compared to a rate of *m* for CMB. In addition, this cost for synchronous algorithms does not include the penalty of jointly interrupting and restarting *all* LPs which increases linearly as the number of LPs increases, contributing to the overhead of time management for *every* LBTS computation. This claim will be discussed further with performance results in section 4.5.

4. Applying the Analytical Null Message Model to Predict Performance

Using the analytical model formulated in the previous section, we can forecast null message activity and overhead indices. These indicators can then be used to predict behavior and performance for different simulation models. We will first estimate null message activity and overhead and then verify our numbers through experimental measurements. For the following sections regarding experimental results, one LP corresponds to one physical processor (CPU).

4.1. Baseline Model

For our study, we used benchmarks developed at Dartmouth College as a set of baseline models for the network modeling and simulation community [22]. This

baseline configuration was created to facilitate and demonstrate network simulator scalability.

Each portion of the network is referred to as a Campus Network (CN). Figure 4 shows the topology for the CN. Each CN consists of 4 servers, 30 routers, and 504 end hosts for a total of 538 nodes. The CN is comprised of 4 separate sub-networks. Net 0 consists of 3 routers, where node 0:0 is the gateway router for the CN. Net 1 is composed of 2 routers and 4 servers. Net 2 consists of 7 routers, 7 LAN routers, and 294 clients. Net 3 contains 4 routers, 5 LAN routers, and 210 clients.

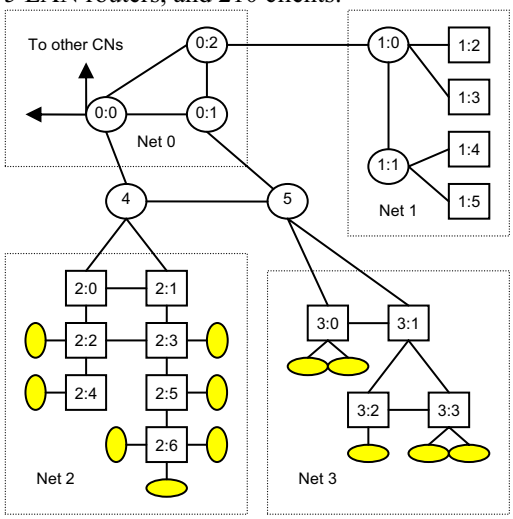

Figure 4. A Campus Network

All non-end host links have a bandwidth of 2Gb/s and have a propagation delay of 5ms with the exception of the Net 0 to Net 1 links, which have a delay of 1ms. End hosts are connected point-to-point with their respective LAN router and have links with 100Mb/s in bandwidth and have a delay of 1ms.

Multiple CNs may be instantiated and connected together to form a ring topology. This aspect of the network allows the baseline model to be easily scaled to arbitrarily large sizes. Multiple CNs are interconnected through a high latency 200ms "ring" link with 2Gb/s bandwidth via their Net 0 gateway router (node 0:0). The baseline model also contains chord links between CN *i* and CN *i + 4* where *i mod 4* is zero. A second set of chord links exist between CN *i* and CN *i + 10* where *i mod 2* is zero and *i* is less than half of the length of the ring. For our tests, we instantiated 7 CNs per LP, which is the maximum number of CNs representable within the total memory available per processor.

In our analysis and performance evaluation, we focus on pure TCP traffic transferred to and requested by end hosts from server nodes. We use a short transfer case of the baseline model, in which clients request 500,000 bytes from a Net 1 server. TCP sessions start at time selected from a uniform distribution over the interval from 0 and 10 seconds of simulation time. The baseline model specifies for TCP traffic to be requested from the neighboring CN that is one ring link hop away. This traffic model exercises only the ring links and no chord links, consequently, we have modified the traffic scenarios for some of our test cases to allow end hosts to request data from any Net 1 server in the ring network at random.

4.2. Scenarios

The baseline model was enhanced to test the effects of irregularity and asymmetry on time management performance. The modifications are intended to exercise irregularity of both traffic and topology. The five scenarios (see Table 1) represent five contrasting configurations, providing a range of benchmarks. Random server selection and varying propagation delay on ring and chord links were parameters that were modified to create scenarios used in the performance study.

Table 1. Benchmark Scenarios

Scenario	Traffic	Chord?	Special Properties
Baseline	Std.	No	None
Baseline-R	Rand.	No	None
Baseline-2ms	Rand.	No	Single 2ms ring link
Chord-R	Rand.	Yes	None
Chord-Asym	Rand.	Yes	Asym. Chord delays

Three scenarios were drafted where no chord links were used. The *Baseline* model consists of standard ring link delays and standard server selection. Traffic restrictions for *Baseline-R* are not limited to the next neighbor but any server in the ring network. *Baseline-2ms* is similar to the previous scenario, but has a single 2ms ring link that spans two LPs.

The other two scenarios contain chord links to add more paths to the network topology. *Chord-R* contains standard 200ms ring and chord delays with random server selection. *Chord-Asym* contains asymmetric, random chord delays from 10-50ms.

4.3. Software and Hardware Platforms

The Parallel/Distributed ns (*pdns*) [23] network simulator was used in the performance study. *pdns* is an HLA RTI-based network simulator that works by federating [24] sequential ns-2 simulators with additional syntax and functionality that allows individual ns-2 instances to communicate and send events to each other. *libSynk* [25, 26] was used as the underlying communications and time management library to manage the *pdns* instances. *libSynk* uses shared memory for communications within an SMP and TCP/IP for communication across SMPs. Optimizations to the RTI for efficient parallel simulation of networks [27] were included. The CMB null message algorithm was added to *libSynk* for the performance tests. The five benchmark scenarios were tested on two different hardware platforms:

Intel Pentium III Linux Cluster (P3) – Consists of 16 8-way 550MHz Pentium II Xeon SMP machines with 4GB RAM connected via Gigabit Ethernet. The operating

system is Red Hat Linux 7.3 running a customized 2.4.18-27.7.xsmp kernel. *libSynk* and *pdns* (based on ns-2.26) were compiled using gcc-3.2.3 with compiler optimizations for the Intel Pentium III architecture.

Intel Itanium 2 Linux Cluster (IA64) – Consists of 30 2-way 900MHz Itanium 2 SMP machines with 6GB RAM connected via Gigabit Ethernet. The operating system is Red Hat Advanced Workstation 2.1 running a 2.4.18-e.31smp kernel. *libSynk* and *pdns* (based on ns-2.26) were compiled using Intel's ia-64 C/C++ compiler 7.1 (ecc).

Scalability tests for the *Baseline* model were conducted on the Pittsburgh Supercomputing Center's "Lemieux" *Compaq Alpha Tru64 Cluster* (PSC). The cluster has 750 4-way 1GHz Alpha ES45 SMP nodes with 4GB RAM connected via a Quadrics switch. *libSynk* and *pdns* (based on ns-2.1b9) were compiled using gcc-3.0.

4.4. Approximating Null Message Transmissions

Using the analytical model for CMB algorithms, the number of null messages sent can be estimated. Table 2 shows null message activity estimates for the *Baseline* scenario along with measurements from the parallel simulator. In this scenario, s=25, *lookahead*=0.2, f=(0.2/3), n=2, and c=0.5.

Table 2. Null Message Send Estimations

Number of CPUs	Approximation	Measurement
4	1,500	1,424
8	3,000	2,974
16	6,000	6,064
32	12,000	11,890
64	24,000	23,586
128	48,000	48,628
256	96,000	96,034
512	192,000	193,862

Table 2 shows that our analytical model predicts the null message counts quite accurately. The discrepancy between the approximation and simulation numbers are due to the estimation of the null frequency and null message cancellation rate. It is nearly impossible to predict the exact interaction between event processing and the effects on null message frequency and cancellations, yet the approximation is very close to the actual null message activity from simulation.

4.5. Approximating Null Message Overhead

Utilizing the previous estimation results for the number of null messages sent and equation 3 derived from the analytical model, it is possible to estimate the synchronization overhead incurred for each scenario.

Simply comparing the overhead index for a particular scenario to its corresponding run time performance (wallclock seconds) is inadequate. This is because the elapsed time to run a simulation does not gauge the amount of work performed by the network simulator. Instead, a new metric is defined: *Packet Transmissions per Second of wallclock time* (PTS). PTS is effectively the number of "packet hops" processed in a second of wallclock time where a "packet hop" represents the transmission of a packet from one node (a router or end node system) to another over a link in the network. The PTS metric will be used to determine work performed by the simulator, and is adequate to compare against the overhead index as the PTS rate is dependent upon time management overhead.

The overhead indices and PTS rates for 32 CPUs are shown in Table 3. For each of the scenarios, it is known that 28 CPUs (87.5%) send null messages through shared memory buffers and the remaining 4 CPUs (12.5%) communicate via TCP/IP over Ethernet. Using the values m=32, p_1=0.875, p_2=0.125, w_1=1, and w_2=10, the overhead index for the *Baseline* scenario is 6.25.

Table 3. Overhead Approximation

Scenario	Overhead Index	PTS
Baseline	6.25	1,512,410
Baseline-R	8.52	865,385
Baseline-2ms	10.54	621,008
Chord-R	9.43	699,848
Chord-Asym	12.21	684,630

Clearly, an inverse relationship between the overhead index and PTS metric exists. The lower the overhead costs for the null message algorithm, the higher the performance of the particular scenario. However, the *Baseline-2ms* and *Chord-Asym* models appear to exhibit conflicting results. The *Chord-Asym* model has a higher overhead index and still manages to have a higher PTS rate than the *Baseline-2ms* model. This anomaly illustrates that the overhead index measures only penalties associated with time management, not for other performance degrading parameters which are a result of the model itself. Although the overhead index does capture lookahead, it does not consider other factors such as load imbalance and the amount of remote communication due to event transmission. Nevertheless, the overhead index can be used as an approximation tool to predict time management efficiency and thus, simulator performance.

4.6. Scalability of the CMB Algorithm

In section 3.3, we stated that, provided all parameters for the scenario remain constant, the overhead due to the null message algorithm does not change as the simulation model increases in scale. We verify the same experimentally here for the *Baseline* scenario. Due to the dynamic nature of the other scenarios, it was impossible to keep other factors such as remote communication due to events and lookahead constant between scalability runs.

Table 4. Run times for the *Baseline* Scenario

Number of CPUs	Null Messages	Reductions
16	784	736
32	783	747
128	787	892

The run time using a CMB null message algorithm remains constant when the model is scaled from 16 to 128

processors as shown in Table 4. Execution times based on global reductions steadily increase from the 16 CPU to the 128 CPU case while remaining relatively constant for null messages. Figure 5 ("Red" denotes global reductions) shows the PTS rate for the two synchronization protocols used in the *Baseline* scenario.

Table 5. Large-Scale Run times for the *Baseline* Scenario

Number of CPUs	Null Messages	Reductions
64	420	508
128	414	523
256	420	563
512	436	620

Table 5 shows the runtimes of the *Baseline* scenario on the Compaq Alpha Tru64 cluster. The corresponding PTS rates are plotted in Figure 6. The run time performance of the baseline scenario using global reductions increases logarithmically as the model scales in the number of processors. In contrast, the null message algorithm performance exhibits a relatively constant run time performance up to 512 processors. These simulation results provide strong support to verify the overhead model and proposition put forth in section 3.3.

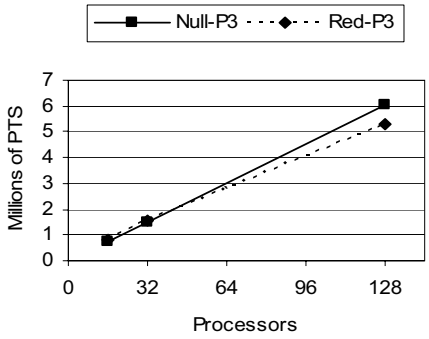

Figure 5. Effect of Scale on PTS Rate

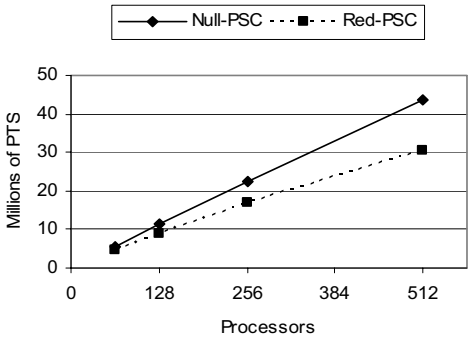

Figure 6. Large-Scale PTS Rate

The experimental results presented in this section have only investigated synchronization behavior of only a structured, regular network (e.g. *Baseline* model). In the next section, we will compare the performance of asynchronous and synchronous algorithms for network models of both regular and irregular structure.

5. Regular and Irregular Network Models

Due to the asynchronous nature of the null message algorithm, synchronization happens "locally" and remains constant even as the simulation model scales. This is the reason why CMB algorithms tend to scale better than schemes based on synchronous time management such as global reductions, if all other factors remain constant.

5.1. Simulation Performance of Structured, Regular Networks

First we examine the performance of synchronous and asynchronous time management in the context of relatively regular network models. The performance of the *Baseline* model on both the Intel Pentium III and Itanium 2 cluster is shown in Figure 7, which was discussed in the previous section.

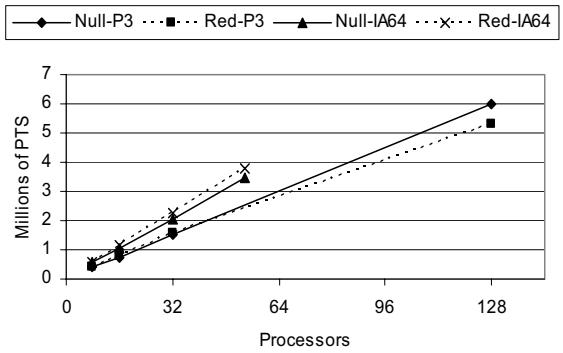

Figure 7. Performance of the *Baseline* Scenario

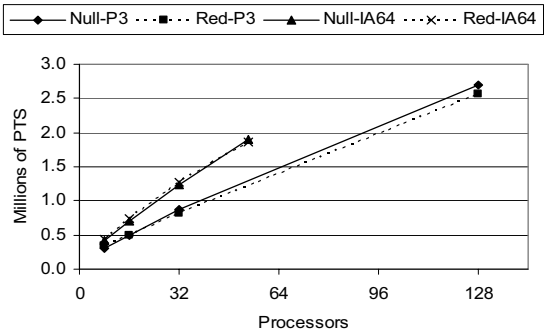

Figure 8. Performance of the *Baseline-R* Scenario

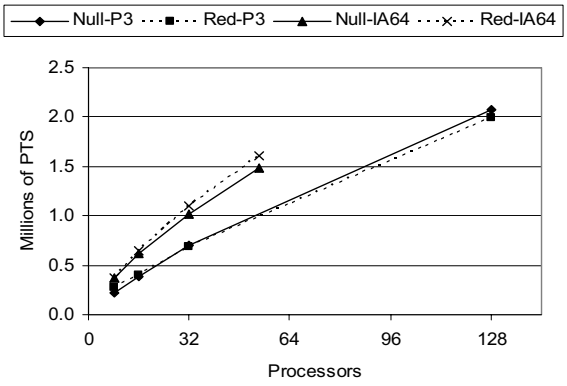

Figure 9. Performance of the *Chord-R* Scenario

The *Baseline-R* scenario is another network considered as a regular model with random server selection. Figure 8 illustrates that the trends of the *Baseline-R* scenario do not differ significantly from that of the *Baseline* model. The random Net 1 server selection pushes the performance asynchronous null message time management ahead of the global reduction algorithm at 16 CPUs for the P3 data and at 54 CPUs for the IA64 data.

The final regular network is the *Chord-R* scenario which is shown in figure 9. Although the addition of chords add asymmetry to the network, the structure and regularity of the network model is preserved by identical lookahead on all chord and ring links. Similar trends to the first two scenarios re-emerge in *Chord-R*.

These scalability runs thus far show that there exist only small gains in using one synchronization algorithm over the other in small- to medium-scale regular networks. However, simulation results suggest that the CMB algorithm with a constant number of output channels is more appropriate for large-scale simulations even in scenarios composed of nearly all consistent structure.

5.2. Simulation Performance of Asymmetric, Irregular Networks

The two irregular network scenarios, *Baseline-2ms* and *Chord-Asym* will highlight differences and deficiencies between synchronous and asynchronous time management algorithms.

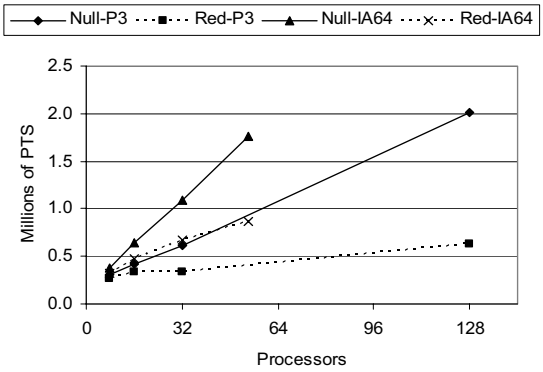

Figure 10. Performance of the *Baseline-2ms* Scenario

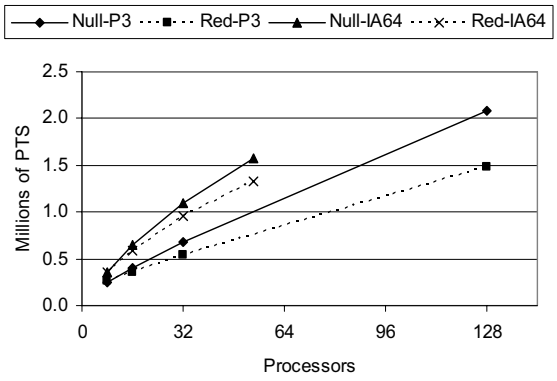

Figure 11. Performance of the *Chord-Asym* Scenario

Figure 10 shows a significant difference in PTS rate between the null message algorithm and global reductions for the *Baseline-2ms* network model. Moreover, the null message trends remain linear while the synchronous global reductions have constant or less than linear increases in PTS rates as the number of processors scale. This scenario shows the pathological case for global reductions, as the time management system must stop and synchronize the simulation every 2ms.

The *Chord-Asym* scenario shown in Figure 11 demonstrates the benefits of a null message algorithm in a typical irregular network. A null message algorithm can take advantage of the varying lookahead values for each output channel while a synchronous algorithm must interrupt execution of the simulation at intervals dictated by the global minimum lookahead.

6. Future Work

It is reasonable to assume that CMB allows simulations to scale to arbitrarily large sizes (assuming parameters discussed earlier are held constant), provided the software infrastructure supports very large scale simulation. Increasing the number of processors beyond 512 to test this conjecture would be the logical next step in furthering the scalability study. We have performed initial 1,024 CPU runs, which seem to indicate that CMB algorithm performs at expected levels extrapolated from table 5. Additional scenarios could be conceived to further test the suitability and adaptability of each type of synchronization algorithm, not only for network simulations, but for other applications as well. Other network simulators, such as *GTNetS* [28] could be used to gather additional data points in order to ascertain if similar improvements in performance is experienced across simulators.

Empirical data gathered on scalability tests over a heterogeneous mix of hardware consisting of varying network and platform (CPU, OS) components would be useful for simulation environments across different computational clusters. It would be interesting to see whether synchronous or asynchronous time management would prove to be the better option in these mixed environment cases.

7. Conclusions

We presented an analytical model for an asynchronous lazy null message algorithm for time management. The proposed equations can be used to approximate null message activity and a corresponding overhead index. In particular, the overhead index predicts that the optimized null message algorithm ensures no appreciable increase in overhead if the fan-in/fan-out of channels is held constant as the simulation scaled with the number of processors.

We also showed that a null message algorithm offers more flexibility for irregular and asymmetric network models. An optimized CMB algorithm can tailor

synchronization messages according to output channel lookahead values to local neighbors only, in contrast to global communication in a synchronous time management system. These properties prove to be advantageous in large-scale network simulation.

Acknowledgments

Funding for this research was provided by DARPA Contract N66001-00-1-8934 and NSF Grant ANI-0136939. Large-scale computations designated as "PSC" were performed on the National Science Foundation Terascale Computing System at the Pittsburgh Supercomputing Center.

References

[1] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*. New York: Wiley-Interscience, 2000.

[2] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, vol. 5, pp. 440-452, 1979.

[3] R. E. Bryant, "Simulation of Packet Communications Architecture Computer Systems," Massachusetts Institute of Technology. MIT-LCS-TR-188 MIT-LCS-TR-188, 1977.

[4] R. Bagrodia and M. Takai, "Performance evaluation of conservative algorithms in parallel simulation languages," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, pp. 395-411, 2000.

[5] W.-K. Su and C. L. Seitz, "Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm," presented at the SCS Multiconference on Distributed Simulation '89, Miami, FL, 1989.

[6] J. Porras, V. Hara, J. Harju, and J. Ikonen, *Improving the Performance of the Chandy-Misra Parallel Simulation Algorithm in a Distributed Workstation Environment*. Arlington, VA, 1997.

[7] W. Cai and S. J. Turner, "An Algorithm for Reducing Null-Messages of the CMB Approach in Parallel Discrete Event Simulation," University of Exeter, 1995.

[8] D. M. Nicol, "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations," *Journal of the ACM*, vol. 40, pp. 304-333, 1993.

[9] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM*, vol. 24, pp. 198-205, 1981.

[10] B. D. Lubachevsky, "Efficient Distributed Event-Drive Simulations of Multiple-Loop Networks," *Communications of the ACM*, vol. 32, pp. 111-123, 1989.

[11] Z. Xiao, B. Unger, R. Simmonds, and J. Cleary, "Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation," presented at the 13th Workshop on Parallel and Distributed Simulation, Atlanta, GA, 1999.

[12] H. Y. Song, R. A. Meyer, and R. Bagrodia, "An Empricial Study of Conservative Scheduling," presented at the 14th Workshop on Parallel and Distributed Simulation, Bologna, Italy, 2000.

[13] M. L. Bailey and M. A. Pagels, "Measuring the Overhead in Conservative Parallel Simulations of Multicomputer Programs," presented at the 23rd Winter Simulation Conference, Phoenix, AZ, 1991.

[14] D. M. Nicol, "Scalability, Locality, Partitioning and Synchronization in PDES," presented at the 12th Workshop on Parallel and Distributed Simulation, Banff, Alberta, Canada, 1998.

[15] D. M. Nicol and J. Liu, "Composite Synchronization in Parallel Discrete-Event Simulation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 433-446, 2002.

[16] D. M. Nicol, "Analysis of Synchronization in Massively Parallel Discrete-Event Simulation," presented at the 2nd ACM SIGPLAN, Seattle, WA, 1990.

[17] D. M. Nicol, C. Michael, and P. Inouye, "Efficient Aggregation of Multiple LP's in Distributed Memory Parallel Simulations," presented at the 21st Winter Simulation Conference, Washington, D.C., 1989.

[18] C.-C. Lim, Y.-H. Low, B.-P. Gan, S. Jain, W. Cai, W. J. Hsu, and S. Y. Huang, "Performance Prediction Tools for Parallel Discrete-Event Simulation," presented at the 13th Workshop on Parallel and Distributed Simulation, Atlanta, GA, 1999.

[19] E. Naroska and U. Schwiegelshohn, "Conservative Parallel Simulation of a Large Number of Processes," *Simulation*, vol. 72, pp. 150-162, 1999.

[20] J. Lemeire and E. Dirkx, "Performance Factors in Parallel Discrete Event Simulation," presented at the 15th European Simulation Multiconference, Prague, 2001.

[21] V. Jha and R. Bagrodia, "A Performance Evaluation Methodology for Parallel Simulation Protocols," presented at the 10th Workshop on Parallel and Distributed Simulation, Philadelphia, PA, 1996.

[22] "NMS Baseline Model. http://www.cs.dartmouth.edu/~nicol/NMS/baseline/."

[23] "pdns. http://www.cc.gatech.edu/computing/compass/pdns/."

[24] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, "A Generic Framework for Parallelization of Network Simulations," presented at the 7th MASCOTS, College Park, MD, 1999.

[25] "libSynk. http://www.cc.gatech.edu/computing/pads/kalyan/libsynk.htm."

[26] R. M. Fujimoto, T. McLean, and K. S. Perumalla, "Design of High Performance RTI Software," presented at the 4th Workshop on Distributed Simulation and Real-Time Applications, San Francisco, CA, 2000.

[27] K. S. Perumalla, A. Park, R. M. Fujimoto, and G. F. Riley, "Scalable RTI-Based Parallel Simulation of Networks," presented at the 17th Workshop on Parallel and Distributed Simulation, San Diego, CA, 2003.

[28] G. F. Riley, "The Georgia Tech Network Simulator," presented at the ACM SIGCOMM '03, Karlsruhe, Germany, 2003.