Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

CHANNEL BASED SEQUENTIAL SIMULATION

Cameron Kiddle
Rob Simmonds
Brian Unger

Department of Computer Science
University of Calgary
Calgary, Alberta, T2N 1N4, CANADA

ABSTRACT

Sequential discrete event simulation is widely employed to study the behavior of many systems. Events are typically managed in a central event list which is implemented as a priority queue ordered by event timestamps. Most research to improve sequential simulation performance has focused on improving the priority queue implementations. Recent work has demonstrated that asynchronous conservative parallel discrete event simulation systems can achieve better sequential performance under some conditions, but worse performance under other conditions. This paper introduces a new sequential discrete event simulation algorithm that can exhibit some of the same performance advantages of asynchronous conservative parallel discrete event simulation algorithms and has complexity no more than that of central event list algorithms in the worst case.

1 INTRODUCTION

Discrete event simulation (DES) is used to test and analyze the behavior of many systems. Events are used to model changes in the system that occur at discrete points in time. Each event has a timestamp to indicate the time that the state change should occur.

Most sequential DES systems employ a single *central event list* (CEL) to manage future events. The CEL is implemented as a priority queue ordered by event timestamps. In many cases the simulator performance depends on on the efficiency of insert and remove operations on the CEL. Many different priority queue algorithms and implementation techniques have been explored in the literature (Jones 1986, McCormack and Sargent 1981, Vaucher and Duval 1975).

Parallel discrete event simulation (PDES) systems have been developed that can decrease the run length of individual simulation runs. However, they do this at the expense of extra complexity and decreased efficiency. It is often the case that many thousands of individual simulation runs are required to complete a study. In this case the efficiency of use of the available resources has a far greater impact on the time taken to complete the study than the speed of any individual run.

Recent work (Curry et al. 2005) has demonstrated that asynchronous conservative discrete event simulation algorithms based on the *Chandy-Misra-Bryant* (CMB) algorithm (Bryant 1977, Chandy and Misra 1979) can show significant performance improvement sequentially over other CEL algorithms. This is due to increased cache locality and decreased priority queue maintenance under certain conditions. However, there are also many conditions under which CMB algorithms exhibit much worse performance than CEL algorithms.

This paper introduces a new sequential discrete event simulation algorithm called the *CHAnnel based SEquential (CHASE)* algorithm that can exhibit the performance advantages of the CMB algorithm while having the same complexity as CEL algorithms in the worst case. A theoretical analysis of the CHASE algorithm is provided along with empirical results that confirm the theoretical analysis.

The rest of the paper is organized as follows. Section 2 provides an overview of CEL and CMB algorithms. Section 3 introduces the CHASE algorithm, provides an informal proof of correctness and also contains a theoretical cost analysis of CHASE. The experimental methodology used to compare performance of the CEL, CMB and CHASE algorithms is present in Section 4, with the experimental results given in Section 5. Conclusions and future work are discussed in Section 6.

2 BACKGROUND

This section provides an overview and cost analysis of CEL and CMB algorithms. Variables used in the cost analysis are defined in Table 1.

Table 1: Variables Used in Cost Analysis

Term	Definition
P	event population (total # events)
N	number of LPs
D	event density = P/N (avg # events at each LP)
C	avg # of channels per LP
L	min lookahead of all channels
E	avg # events per LP execution
μ	avg lifetime of an event

2.1 CEL Algorithms

Most sequential discrete event simulation systems use a single central event list (CEL) to manage all future events. Events are removed from the CEL and executed in non-decreasing timestamp order. During the execution of an event, new events could be created that are inserted into the CEL. Many studies have been performed that compare the performance of CEL algorithms using different priority queue implementations (Jones 1986, McCormack and Sargent 1981, Vaucher and Duval 1975).

Examples of priority queue implementations include the *linked list*, *heap*, *splay tree* (Sleator and Tarjan 1985) and the *calendar queue* (Brown 1988). Per event costs of these CEL algorithms can be found in Table 2. The per event cost is the cost of inserting and removing an event from the priority queue. P is the event population (i.e., number of events in the priority queue).

Table 2: Per Event Cost of CEL Algorithms

linked list	heap	splay tree	calendar
$O(P)$	$O(\log P)$	$O(\log P)$	$O(1)$

2.2 CMB Algorithm

Most asynchronous conservative PDES systems employ algorithms derived from the Chandy-Misra-Bryant (CMB) (Bryant 1977, Chandy and Misra 1979) algorithm. In conservative PDES algorithms, causality errors, where events that affect the same state variable are executed out of order, are strictly avoided. This is in contrast to optimistic PDES algorithms that can execute related events out of order, but employ mechanisms to recover when this situation is detected (Jefferson 1985).

The real system is viewed as a set of *physical processes* that interact only by exchanging messages. Each physical process is mapped to a *logical process* (LP) in the simulation system and messages are mapped to events. Unidirectional channels are set up between any pair of LPs that could communicate with each other. Associated with each channel is a clock that represents the lower bound on the timestamp of future events to be received from the channel. A minimum lookahead value is also associated with each channel representing the minimum *lifetime* of any event that can be sent on the channel. The lifetime of an event is defined to be the difference between the timestamp assigned to the event and the timestamp of the event whose execution caused this event to be generated.

Events must be sent on a channel in nondecreasing timestamp order and received in the same order. This guarantees that the timestamp of the last event received from the channel is a lower bound on the timestamp of any future events that will be received. As long as an LP has an event on each input channel, it is safe to execute the event with the smallest timestamp. To avoid deadlock in the absence of events, LPs send *NULL messages* on output channels to update channel clocks. Note that in sequential and shared memory parallel computers it is not necessary to explicitly send NULL messages. The channel clock variable can simply be updated by the sender LP.

The CMB algorithm has costs associated with scanning channels to determine the time up to which it is safe to execute events and to update channel clocks, costs associated with sorting the LP scheduling queue and costs associated with sorting an LP's local event priority queue. Assuming that the queues are implemented as heaps, that events are uniformly distributed among LPs and that P, N, D and C are constant, the per event cost for the sequential execution of the CMB algorithm as derived in (Curry et al. 2005) is

$$\text{Per Event Cost} = O\left(\frac{C+\log N}{E}+\log D\right).$$

When $E > 1$ the LP scheduling queue is accessed less frequently, reducing sorting costs in comparison to the CEL algorithms. Also, better cache behavior is expected as the LP state remains in cache for the execution of E events. When $E < 1$ the LP scheduling queue is accessed more than once per event on average, giving rise to worse cache performance and greater sorting costs in comparison to CEL algorithms. High connectivity can also lead to poor performance for the CMB algorithm.

It was also shown in (Curry et al. 2005) that the expected minimum time advance per LP execution session will be L on average giving an expected minimum value of $E_{min} = LD/\mu$ events per LP execution. The per event cost of a CMB algorithm can now be expressed as

$$\text{Per Event Cost} = O\left(\frac{\mu(C+\log N)}{LD}+\log D\right).$$

The above cost expression is influenced by the number of LPs, the event density, the connectivity, the minimum lookahead and the average lifetime of an event. Table 3 summarizes the expected behavior when modifying a given parameter and keeping the other parameters constant. Empirical results in (Curry et al. 2005) confirmed the expected

behavior of CMB. Excellent performance has been achieved for high event density and high lookahead with poor performance achieved for low event density, low lookahead or high connectivity.

Table 3: Expected Behavior of CMB when Modifying Model Parameters

N	D	C	L	μ
$O(\log N)$	$O(\frac{1}{D} + \log D)$	$O(C)$	$O(\frac{1}{L})$	$O(\mu)$

3 CHASE

The *CHAnnel based SEquential* (CHASE) algorithm aims at taking advantage of the cache benefits of the CMB algorithm while still achieving the same order of complexity of CEL algorithms in the worst case. For example, compared to a heap based CEL algorithm, CHASE must have a per event cost no greater than $O(\log(P))$. This can be achieved by avoiding problems associated with low lookahead cycles to ensure that there is at least one event per LP execution and by eliminating channel scanning costs.

Approaches that address low lookahead cycles for CMB in a parallel environment such as *Carrier NULL Messages* (Cai and Turner 1990) and *cooperative acceleration* (Blanchard, Lake, and Turner 1994) pass extra information between LPs in an attempt to advance to the next event more quickly. CHASE employs a much simpler mechanism since the processor has knowledge of all events in a sequential environment.

This section proceeds by describing the CHASE algorithm, giving an informal proof of correctness and then by deriving the complexity of the algorithm.

3.1 Algorithm

Up until the simulation end time is reached, the simulation proceeds by repeatedly removing the first LP in the LP scheduling queue, executing the LP and inserting the LP back into the scheduling queue if it has at least one event. An LP that has no events in its local priority queue at the end of an execution session will be inserted into the LP scheduling queue when the next event is sent to the LP. The LP scheduling queue is sorted by the lowest timestamped event at each LP and consists of only those LPs that have at least one event in its local priority queue. Therefore, the clock of the LP scheduling queue is that of the lowest timestamped event in the system. This allows the simulation to directly advance to the lowest timestamped event in the system after each LP execution.

Pseudocode for an LP execution session is shown in Figure 1. The LP execution proceeds by executing events in the LP's local priority queue in timestamp order while at least one of the following two conditions is met: 1) none of the LP's input channels are empty (i.e., contain no events) or 2) the timestamp of the next event to process is less than that of the *safetime* (i.e., lower bound on the timestamp of future events to be received by the LP) (lines 1-4). A counter is maintained indicating the number of empty input channels. When there are no empty input channels it is guaranteed that no event will arrive with a timestamp less than that of the next event to be processed. If there are one or more empty input channels then it is still safe to process the next event if its timestamp is less than or equal to the safetime.

```
1.  While no channels are empty or timestamp
    of next event is less than safetime:
2.      Set LP clock to timestamp of next event.
3.      Process next event.
4.      Sample channel for event if necessary.
5.  Set the LP clock to timestamp of next event
    or ∞ if no events.
```

Figure 1: Pseudocode for LP Execution Session

Rather then scanning each input channel to determine the safetime, a simpler approach that eliminates channel scanning costs is employed. The safetime is taken to be the clock of the LP scheduling queue (which is the lowest timestamped event at any other LP in the model) plus the minimum lookahead of all input channels to the LP currently being executed. If the LP scheduling queue is empty the clock of the LP scheduling queue is taken to be infinity. No event from any other LP will arrive with a timestamp less than this time. Since it is possible for the clock of the LP scheduling queue to change when events are sent to other LPs, the safetime must be kept up to date.

Note that there might not be any input channels from the first LP in the LP scheduling queue to the LP currently being executed. Also, it could be possible that such an input channel does exist, but with a lookahead that is greater than the minimum lookahead of all input channels to the LP. As such, this approach for determining safetime could result in a more conservative estimate than that obtained using the CMB algorithm. However, as the lowest timestamped event at any other LP in the system is being used in the safetime calculation, it is possible that the safetime estimate could be far greater than that obtained using the CMB algorithm.

Before processing an event, the clock of the LP is set to the timestamp of the event (line 2). After processing an event (line 3), if the event came from a channel, the channel is sampled for another event (line 4). Sampling involves receiving the next event from the channel and inserting it in the LP's local priority queue, or if there is no event, marking the channel not sampled and incrementing the counter of empty input channels. Only one event from each channel is kept in the local priority queue as is done for the CMB based algorithm described in (Curry et al. 2005). This

reduces sorting costs by keeping as many events in FIFO queues as possible.

After all safe events have been processed the clock of the LP is set to the timestamp of the next event in the LP's local priority queue or to infinity if there are no events in the LP's local priority queue (line 5). This is done so that the LP scheduling queue is sorted according to the lowest timestamped event at each LP. Note that it is still possible for an LP to receive an event with a timestamp lower than this clock value.

To ensure that the clock of the LP scheduling queue is always accurate the process of sending events to channels was changed from that used in the CMB algorithm. For the CMB algorithm, when an event is sent to a channel it is simply appended to the FIFO queue of events, and the receiving LP will eventually remove it from the channel. For CHASE, when an event is sent to the channel, if the channel is sampled (i.e., not empty) it is appended to the end of the FIFO queue. If the channel is not sampled (i.e., empty), the channel is marked sampled, the counter of empty input channels at the destination LP is decremented and the event is directly inserted into the destination LP's local priority queue. The destination LP clock is updated to the time of the lowest timestamped event in the LP's local priority queue. If the destination LP is not in the LP scheduling queue it is inserted into the LP scheduling queue. If the destination LP is already in the LP scheduling queue and the value of the LP clock is lower than it was before the event was inserted, a decrease key operation is performed on this LP to keep the LP scheduling queue sorted.

3.2 Proof of Correctness

This section contains an informal proof of the CHASE algorithm. To prove that CHASE is correct it must be shown that a simulation will successfully terminate by processing all events with a timestamp less than or equal to the simulation end time without any causality errors occurring.

To prove that no causality errors will occur it suffices to show that each LP obeys the *local causality constraint* as defined in (Fujimoto 2000), i.e., each LP must process events in nondecreasing timestamp order. During an LP execution an LP processes events in nondecreasing timestamp order while there are no empty channels. Since events are sent on channels in nondecreasing timestamp order and received in the same order, this guarantees that no event with a timestamp less than that of the event being processed will be received in the future. When one or more empty channels exist only events with timestamps that are less than the safetime (i.e., clock of the LP scheduling queue plus the minimum lookahead of all input channels to the LP currently being executed) may be processed. Since the clock of the LP scheduling queue is that of the lowest timestamped event at any other LP, no event can arrive with a timestamp less than the calculated safetime in the future. Therefore, the local causality constraint is obeyed.

To prove that the simulation will terminate it suffices to show that deadlock is avoided and that progress is made each LP execution session. Deadlock is avoided as an LP is always placed in the LP scheduling queue if it has an event with a timestamp less than or equal to the simulation end time (i.e., there is always an LP that can be executed if the simulation end time has not been reached). An LP is only inserted into the LP scheduling queue if it has at least one event. Also, the LP scheduling queue is sorted by the lowest timestamped event at each LP. Therefore, when an LP executes, it contains the lowest timestamped event in the system which is safe to process according to the safetime calculation. This guarantees that at least one event is processed each LP execution session. Therefore, progress is made each LP execution session and the simulation end time will eventually be reached (assuming that only a finite number of events with a timestamp less than or equal to the simulation end time are generated).

Since the local causality constraint is obeyed, deadlock is avoided and progress is made each LP execution session, the simulation will successfully terminate proving the correctness of the CHASE algorithm.

3.3 Cost Analysis

As channel scanning has been eliminated the simulation overhead per event (assuming that heaps are used for both the LP scheduling queue and the LP's local priority queue) can be divided up into two parts as follows:

1. **LP Scheduling Queue Cost:** $O(\log N)$ if $D >= 1$, $O(\log P)$ if $D < 1$ - For the CHASE algorithm an LP is removed from and inserted into the LP scheduling queue once per LP execution session as is the case for the CMB algorithm. This gives a per event cost of $O(\log N/E)$ as per the CMB algorithm. An event may also have an additional LP scheduling queue sorting cost if the is event sent on an empty channel and causes the destination LP to lower its clock. This is $O(\log N)$ complexity. Therefore the per event LP scheduling queue cost is $O(\log N/E) + O(\log N)$ which works out to be $O(\log N)$ in the worst case since $E >= 1$ for CHASE. Sorting due to events sent on empty channels may be infrequent and thus CHASE could achieve the same best case performance as CMB. For CMB there are always N LPs in the LP scheduling queue. However, for CHASE an LP is in the LP scheduling queue only if it has an event. Therefore, if $P < N$, which is the case for $D < 1$, then the cost is $O(\log P)$.

2. **Local Event Priority Queue Cost:** $O(\log D)$ - An event is inserted into and removed from an LP's local event priority queue once per event execution, so the per event cost is $O(\log D)$. Like the CMB algorithm, this reduces to $O(\log C)$ if $C <= D$ and no local events are generated (i.e., all events are sent on channels to other LPs). In a system where events are not uniformly distributed among LPs then the worst case scenario is actually $O(\log P)$ as it could be possible that all events are located on the same LP.

Combining the two costs together for the case where $D >= 1$ gives a per event cost of $O(\log N + \log D) = O(\log(ND)) = O(\log P)$. For $D < 1$, the local event priority queue cost can be ignored also giving $O(\log P)$. Therefore the per event cost of CHASE using heaps for both the LP scheduling queue and an LP's local priority queue is:

$$\text{Per Event Cost} = O(\log P)$$

This is the same per event cost as the heap CEL based algorithm. Similar analysis could be performed for other types of priority queues. Although CHASE has the same complexity of CEL algorithms in the worst case, it still has the potential to gain the advantages of the CMB algorithm if $E > 1$ and sorting of the LP scheduling queue due to events sent on empty channels is infrequent.

4 EXPERIMENTAL METHODOLOGY

This section describes the experimental methodology used to evaluate the sequential performance of the CHASE algorithm with respect to CEL and CMB algorithms. Included are descriptions of the experimental environment, simulation model and performance metrics.

4.1 Experimental Environment

The CEL, CMB and CHASE algorithms that are examined in experiments are implemented as part of the same simulation kernel and make use of the same model code. The CMB and CHASE implementations use a heap for the LP scheduling queue and a linked list for the local event queue at each LP. A linked list is used for the local event queue as this queue usually does not contain a large number of events and linked lists perform well for small queue sizes.

Experiments were run on an IBM eServer BladeCenter HS20 with 2 GB RAM and two 3.0 GHz Intel Xeon processors. Each processor has a 12 KB micro-op instruction trace cache, an 8 KB, 4-way associative first level (L1) data cache with a 64 byte line size, and a 512KB, 8-way associative second level (L2) cache with a 64 byte line size. The BladeCenter was running Red Hat Linux 9.0 with the v2.4.20-24.9gpfs kernel. The GNU g++ V3.2.2 compiler was used with the "-O2" optimization flag.

Cachegrind, which is part of *Valgrind 2.2.0* and is available at `<http://valgrind.kde.org>`, was used to determine instruction counts and analyze cache behavior. It was configured to simulate an L2 cache with the same specifications as that for the computer that the experiments were run on, as described above.

4.2 Simulation Model

A ring model, also used in Curry et al. (2005), was used for the experiments. The ring model does not implement any real system, but it allows the effects of model size, event density per LP, connectivity and lookahead to be examined. An example ring model can be seen in Figure 2. The model is parameterized with $N = 8$ LPs, an average event density of $D = 4$ events per LP, a connection radius $R = 2$, and a minimum channel lookahead $L = 1$ simulation time unit. Each LP is connected to R LPs ahead in the ring and R LPs behind in the ring with channel lookahead L.

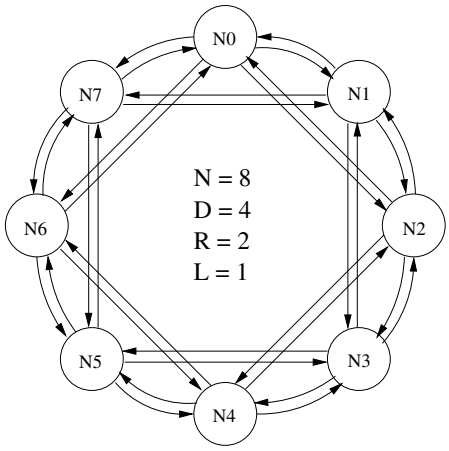

Figure 2: Example Ring Model

Before the simulation is started, the system is populated with $N \times D$ events with timestamps selected independently from an exponential distribution with a mean of 1 simulation time unit. The events are uniformly distributed among LPs such that on average, each LP is populated with D events. The initial events are considered to be local events (i.e., events generated locally at the same LP).

Upon processing a local event, an output channel is selected randomly with uniform probability. A new external event is generated with a timestamp equal to the timestamp of the current event plus the minimum lookahead L assigned to the selected output channel. Upon receiving an event from a neighboring LP a local event is generated with a timestamp equal to the timestamp of the current event plus an increment drawn from an exponential distribution with a

mean of 1 simulation time unit. The total event population P in the system remains constant at $P = N \times D$ throughout the simulation.

4.3 Performance Metrics

Four metrics are used to analyze the performance of the algorithms, namely the *instructions per event*, *cache misses per event*, *events per LP execution* and *event rate*. The first three metrics are obtained while running the simulator with Cachegrind, whereas the event rate metric is obtained while running the simulator without Cachegrind.

Instructions per event is the total number of instructions divided by the total number of events executed. Only instructions after initialization are taken into account.

Cache misses per event is the total number of L2 data cache misses divided by the total number of events executed. Only cache misses after initialization are taken into account. L1 cache misses are not examined as the cost of an L2 cache miss is much larger.

Events per LP execution is the total number of events divided by the total number of LP execution sessions. This is easily calculated for the CMB algorithm. For the CEL algorithms this is taken to be the average number of events executed consecutively at the same LP.

Event rate is the total number of events divided by the wallclock time (i.e., execution time) taken to run the simulation. Only wallclock time after initialization is taken into account. This metric is taken from runs where the simulator is run without Cachegrind so that the performance overhead of Cachegrind does not affect the results.

5 EXPERIMENTAL RESULTS

This section compares the performance of three different CEL algorithms, the CMB algorithm and the CHASE algorithm. The discussion of results in this section primarily focus on the CHASE algorithm. For more details pertaining to the behavior of the CEL and CMB algorithms see Curry et al. (2005).

Each test was run twice with Cachegrind and once without it. One of the Cachegrind runs used a simulation end time of 100 simulation units and the other Cachegrind run was terminated after initialization so that the effects of initialization could be eliminated from the results. The run without Cachegrind lasted for 60 seconds of wallclock time, excluding initialization. The three runs were repeated 5 times using different random number seeds. The metrics were averaged over the 5 runs and corresponding 95% confidence intervals calculated. The half-width of the confidence interval was less than 5% of the sample mean for all metrics in all cases.

5.1 Event Density Results

The first set of experiments examine the effects of varying the event density with number of LPs $N = 8192$, connection radius $R = 1$ and minimum channel lookahead $L = 1$. Figure 3(a) shows a plot of the instructions per event versus the event density. The plot clearly shows the poor behavior of CMB at low event density. The CHASE algorithm does not exhibit this behavior at low event density but rather exhibits the same behavior as the CEL algorithms as is expected. At higher event densities the behavior of the CHASE algorithm follows that of the CMB algorithm.

Figure 3(b) shows a plot of the L2 cache misses per event versus the event density. At low event density the CMB algorithm exhibits a very high cache miss rate due to the higher number of instructions per event. With increasing event density the cache miss rate for the CMB algorithm decreases resulting in a much lower cache miss rate than CEL algorithms for higher event densities. The CHASE algorithm exhibits a similar cache miss rate to the CEL algorithms at low event density and a similar cache miss rate to the CMB algorithm at high event density.

A plot of the events per LP execution versus the event density is shown in Figure 3(c). A single line is plotted for all of the CEL algorithms as the events per LP execution values are the same for all of these algorithms. The number of events executed consecutively at the same LP is usually one. This means that the next event in the central event list most often occurs at a different LP. The LP might not be in the cache so the cache miss rate increases. Due to the nature of the ring model, the CEL algorithms achieve one event per LP execution on average for all of the test sets in this paper. It should be noted that there are cases where CEL algorithms would achieve greater than one event per LP execution which are not captured by this model.

The expected minimum number of events per LP execution is also plotted for the CMB algorithm. This was derived to be $E_{min} = \frac{LD}{\mu} = \frac{2LD}{1+L}$ in Curry et al. (2005). For the CHASE algorithm the same derivation applies except that $E >= 1$. Therefore for the CHASE algorithm $E_{min} = max(1, \frac{2LD}{1+L})$.

The expected maximum number of events per LP execution is also plotted for the CMB algorithm. This was derived to be $E_{max} = \frac{4LD}{1+L} = 2E_{min}$ in Curry et al. (2005). This maximum does not apply to the CHASE algorithm. The E_{min} and E_{max} curves for the CMB algorithm are also plotted on the events per LP execution graphs for the remaining test sets in this paper.

The plot shows that the number of events per LP execution for the CHASE algorithm stays above one at low event density. In contrast, the number of events per LP execution for the CMB algorithm becomes significantly less than one at lower event densities. At low event density the CMB algorithm suffers the problem of executing many LPs before

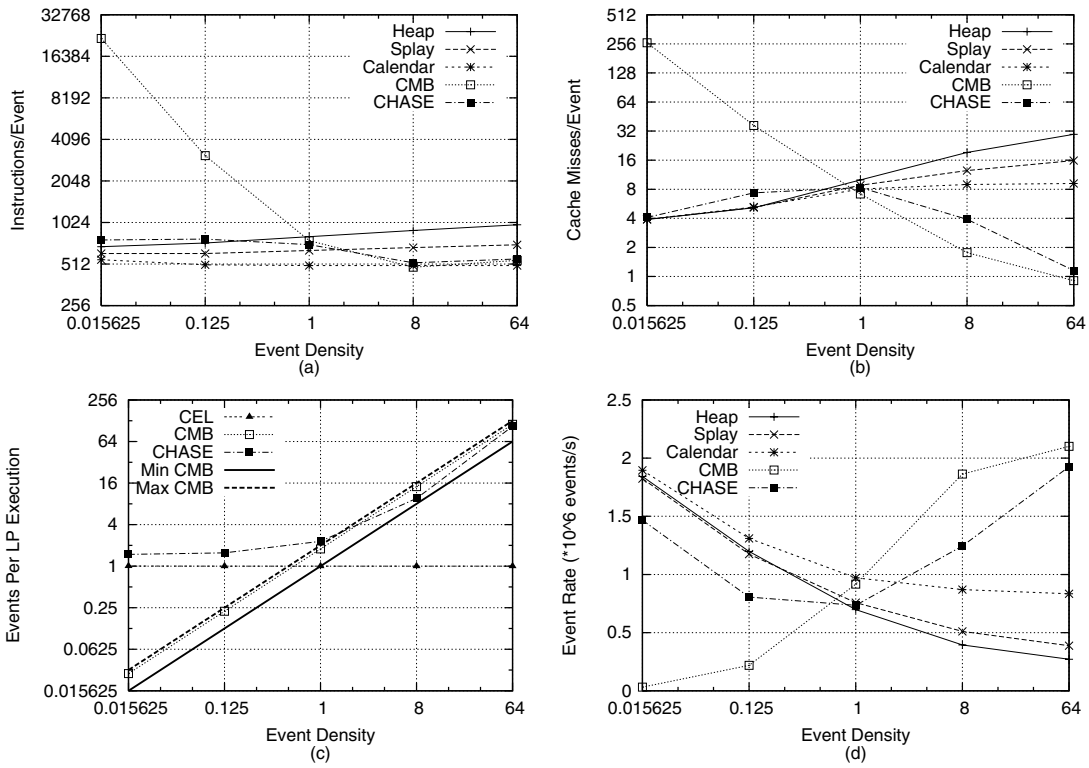

Figure 3: Plots of (a) Instructions Per Event, (b) Cache Misses Per Event, (c) Events Per LP Execution and (d) Event Rate vs. the Event Density for N=8192, R=1 and L=1

executing an LP that contains an event. After each LP execution session, the CHASE algorithm jumps directly to the LP containing the lowest timestamped event in the model. The events per LP execution increases with event density for the CMB algorithm and CHASE algorithm. At higher event densities the number of events per LP execution for both the CHASE and CMB algorithms are similar. This explains why CHASE has a similar cache performance to the CEL algorithms at low event density and to the CMB algorithm at higher event densities.

Figure 3(d) shows a plot of the event rate versus the event density. The performance of the CHASE algorithm is neither the best nor the worst. It typically lies in between the performance of the CEL algorithms and the CMB algorithm and is most similar to the performance the algorithm that performs the best for a particular event density range. At low event density, the CEL algorithms perform best and the CHASE algorithm achieves similar, but somewhat lower performance due to extra sorting costs. At high event density, the CMB algorithm performs best and the CHASE algorithm achieves similar, but somewhat lower performance due to a more conservative safetime estimate that results in fewer events per LP execution and hence a higher cache miss rate.

5.2 Lookahead Results

The second set of experiments examine the effects of varying the minimum channel lookahead on the CEL, CMB and CHASE algorithms with number of LPs $N = 8192$, event density $D = 8$ and connection radius $R = 1$. Figure 4(a) shows a plot of the instructions per event versus the minimum channel lookahead. Nearly constant behavior is observed for the CEL algorithms as expected. While the instruction cost for CMB is somewhat less than that for the CEL algorithms at high lookahead, it becomes significantly greater at low lookahead. This is due to the temporal separation of events being much greater than the minimum lookahead resulting in low lookahead cycles. The instruction cost for CHASE is similar to that for CMB at high lookahead, increases somewhat as lookahead starts to decrease, but approaches constant behavior like the CEL algorithms rather than continually increasing like CMB. This is due to the ability of CHASE to avoid low lookahead cycles by immediately jumping to the timestamp of the next event after an LP execution session.

Figure 4(b) shows a plot of the L2 cache misses per event versus the minimum channel lookahead. Overall, the cache miss rate for the CEL algorithms is close to constant. The observed variation could be due to the distribution of event timestamps being dependent on the lookahead since

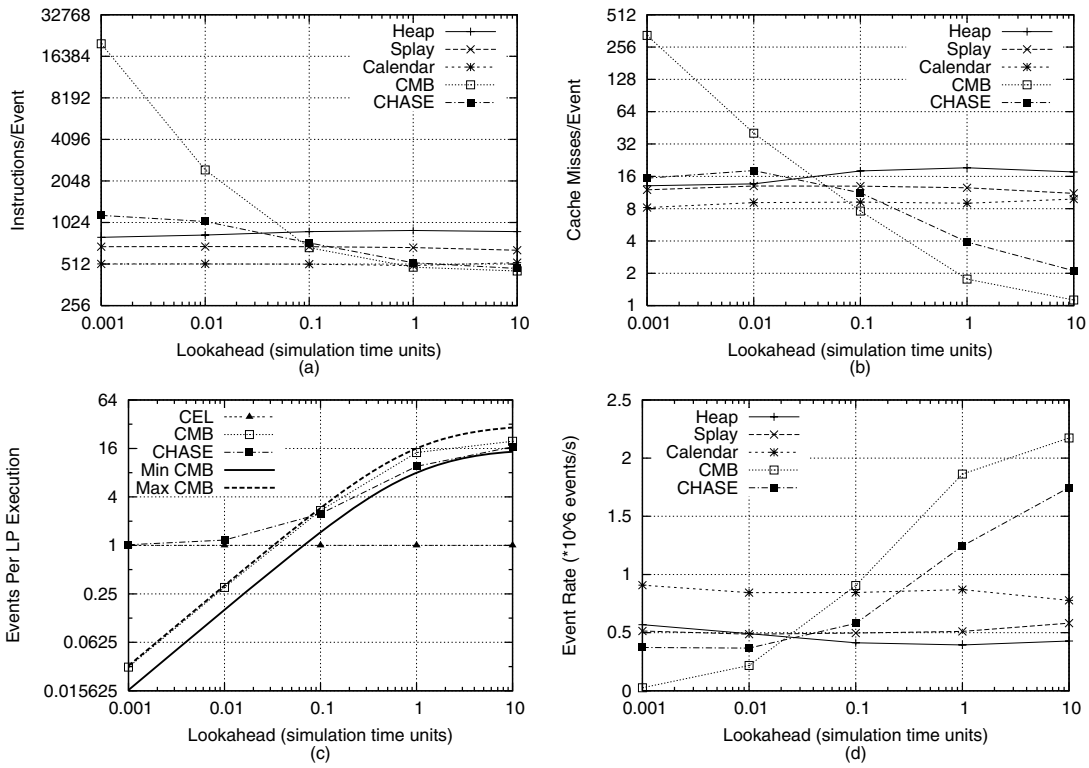

Figure 4: Plots of (a) Instructions Per Event, (b) Cache Misses Per Event, (c) Events Per LP Execution and (d) Event Rate vs. the Lookahead for N=8192, D=8 and R=1

half of the events are generated with a lifetime equal to the minimum lookahead. The cache miss rate for the CMB based algorithm starts out very high (due to the state of many LPs being accessed in low lookahead cycles) and decreases as lookahead increases, eventually exhibiting much better cache performance than the CEL algorithms. The cache miss rate for CHASE is similar to the CEL algorithms at low lookahead as low lookahead cycles are avoided. At higher lookahead values, the cache performance of CHASE follows that of the CMB algorithm.

A plot of the events per LP execution versus the minimum channel lookahead is shown in Figure 4(c). At low lookahead, the CHASE algorithm is able to achieve at least one event per LP execution while the CMB algorithm does very poorly. As lookahead increases, the number of events per LP execution for the CHASE algorithm increases and eventually exhibits similar behavior to the CMB algorithm.

Figure 4(d) shows a plot of the event rate versus the minimum channel lookahead. At low lookahead the CHASE algorithm achieves near constant event rates as do the CEL algorithms. As the lookahead increases, the event rate for the CHASE algorithm begins to increase along with the event rate for the CMB algorithm. Once again the performance of the CHASE algorithm is neither the best or the worst, but exhibits similar behavior to the algorithm that performs best over a certain lookahead range.

5.3 Connectivity Results

The third set of experiments examines the effects of varying connectivity on the CEL, CMB and CHASE algorithms with number of LPs $N = 8192$, event density $D = 8$ and minimum channel lookahead $L = 1$. Figure 5(a) shows a plot of the instructions per event versus the connection radius. The instruction cost for the CMB algorithm increases with increasing connection radius due to greater channel scanning costs. The instruction cost for the CHASE algorithm remains constant like the CEL algorithms. This is expected as the CHASE algorithm does not have channel scanning costs.

Figure 5(b) shows a plot of the L2 cache misses per event versus the connection radius. The CEL algorithms do not need to access additional state as the connection radius increases and therefore the cache miss rate remains constant. The cache miss rate for the CMB algorithm increases with increasing connectivity as a greater number of channels must be scanned during each LP execution. For the CHASE algorithm, the cache miss rate starts low, increases and eventually levels off at a cache miss rate close to that of the CEL algorithms as the connectivity increases. The cache miss rate starts out low for the CHASE algorithm as at lower connectivity multiple events may be sent to or received from the same channels, thus accessing the same channel state more than once in a given LP execution session.

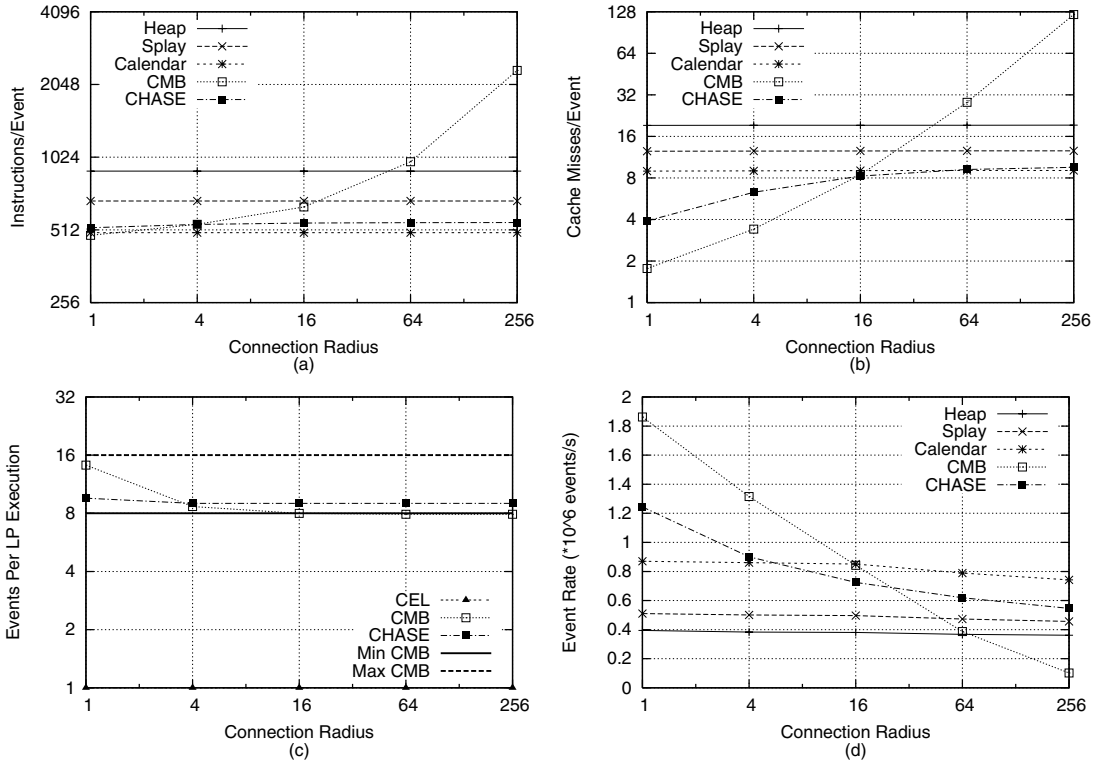

Figure 5: Plots of (a) Instructions Per Event, (b) Cache Misses Per Event, (c) Events Per LP Execution and (d) Event Rate vs. the Connection Radius for N=8192, D=8 and L=1

As the connectivity increases there are a greater number of different channels that events are sent to and received from, increasing the cache miss rate. The cache miss rate eventually levels off, rather then continually increasing as for the CMB algorithm, as the state for only those channels on which events are sent to or received from need to be accessed (i.e., the same number of channels are accessed each LP execution session on average regardless of the connection radius). For the CMB algorithm the state of each channel must be accessed each LP execution session.

A plot of the events per LP execution versus the connection radius is shown in Figure 5(c). The events per LP execution for the CMB algorithm starts out near the expected maximum value but approaches the expected minimum value as the connection radius increases. As the connection radius increases, an LP has more neighbors so it is unlikely that all neighbors are L simulation time units ahead at the start of each LP execution session. It is more likely that there will be one or more neighbors that are close in time allowing the LP to advance only L simulation time units instead of $2L$ simulation time units. The events per LP execution for the CHASE algorithm is constant and close to the minimum expected value. At higher connectivity CHASE actually achieves a slightly higher value for events per LP execution than the CMB algorithm. This is likely due to a greater safetime estimate that can be achieved by taking into account the lowest timestamped event at another LP, rather than by scanning channels.

Figure 5(d) shows a plot of the event rate versus the connection radius. The event rate for the CMB algorithm starts out high due to good cache performance and decreases with increasing connection radius due to more instructions and cache misses. Eventually, it exhibits the worst performance of all of the algorithms. The CHASE algorithm starts out with a higher event rate than the CEL algorithms, due to better cache performance initially, but eventually exhibits similar performance levels as the CEL algorithms due to a constant number of instructions per event and a cache miss rate that approaches that of the CEL algorithms. Since the number of instructions per event is constant and the cache miss rate levels off, one would expect that the event rate for the CHASE algorithm should level off. However it still tends to decrease somewhat with increasing connection radius. This performance degradation is observed for the calendar queue CEL algorithm and also for the heap and splay tree CEL algorithms to a lesser extent. Further exploration is needed to determine the cause of this behavior. Once again, the performance of the CHASE algorithm is neither the best nor the worst.

6 CONCLUSIONS AND FUTURE WORK

This paper explored the range of performance that can be achieved by a the new CHASE sequential discrete event simulation algorithm in comparison to the performance of several CEL algorithms and the sequential performance of the CMB algorithm. In cases where the CMB algorithm exhibits better performance than CEL algorithms, such as high event density and high lookahead, empirical results demonstrated that the CHASE algorithm can also exhibit better performance than CEL algorithms. Empirical results also confirmed a theoretical analysis that the CHASE algorithm has the same complexity as CEL algorithms in the worst case. Thus the CHASE algorithm avoids the performance pitfalls of the CMB algorithm when event density is low, lookahead is low or connectivity is high.

Among the different algorithms tested, CHASE never achieved the best performance or the worst performance. However it did follow the behavior of the algorithm with the best performance over a given model parameter range. This suggests that the CHASE algorithm may serve as a better general purpose algorithm. If the model characteristics such as event density and lookahead are well known then it would be more appropriate to choose an algorithm that performs best for the model characteristics in question. However if the model characteristics are unknown, or if the model covers a wide range of characteristics then it may be more appropriate to employ the CHASE algorithm.

Studies involving different models, parameters and event distributions would be useful. Particularly, it would be interesting to explore if the CHASE algorithm exhibits the best performance for models that have a wide range of event densities or lookahead values.

ACKNOWLEDGMENTS

The authors would like to thank Roger Curry for the initial development of the simulation testing environment, done as part of his M.Sc. thesis work, that was used and expanded upon for experiments in this paper. Financial support for this research was provided by ASRA (Alberta Science and Research Authority) and NSERC (Natural Sciences and Engineering Research Council of Canada). Experiments were run on computer equipment purchased through the WestGrid project <www.westgrid.ca>, a collaborative high performance computing project among research institutions in Western Canada.

REFERENCES

Blanchard, T. D., T. W. Lake, and S. J. Turner. 1994. Cooperative acceleration: Robust conservative distributed discrete event simulation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, 58–64.

Brown, R. 1988. Calendar queues: A fast $O(1)$ priority queue implementation for the simulation event set problem. *Communications of the ACM* 31 (10): 1220–1227.

Bryant, R. E. 1977. Simulation of packet communication architecture computer systems. Technical Report MIT-LCS-TR-188, Laboratory for Computer Science, Massachusetts Institute of Technology.

Cai, W., and S. J. Turner. 1990. An algorithm for distributed discrete-event simulation – the "carrier null message" approach. In *Proceedings of the SCS Multiconference on Distributed Simulation*, Volume 22 of *SCS Simulation Series*, 3–8.

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

Curry, R., C. Kiddle, R. Simmonds, and B. Unger. 2005. Sequential performance of asynchronous conservative PDES algorithms. In *Proceedings of the 2005 Workshop on Principles of Advanced and Distributed Simulation*, 217–226.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. New York: John Wiley & Sons, Inc.

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

Jones, D. W. 1986. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM* 29 (4): 300–311.

McCormack, W. M., and R. G. Sargent. 1981. Analysis of future event set algorithms for discrete event simulation. *Communications of the ACM* 24 (12): 801–812.

Sleator, D. D., and R. E. Tarjan. 1985. Self-adjusting binary search trees. *Journal of the ACM* 32 (3): 652–686.

Vaucher, J. G., and P. Duval. 1975. A comparison of simulation event list algorithms. *Communications of the ACM* 18 (4): 223–230.

AUTHOR BIOGRAPHIES

CAMERON KIDDLE is a Postdoctoral Fellow in the Grid Research Centre at the University of Calgary. Dr. Kiddle received B.Sc. and Ph.D. degrees in Computer Science and a B.Sc. degree in Physics from the University of Calgary. His e-mail address is <kiddlec@cpsc.ucalgary.ca>.

ROB SIMMONDS is the Research Director of the Grid Research Centre and Adjunct Assistant Professor of Computer Science at the University of Calgary. He is also the Chief Technology Officer of WestGrid. Dr. Simmonds received B.Sc. and Ph.D. degrees from the School of Mathematical

Sciences at the University of Bath. His e-mail address is <simmonds@cpsc.ucalgary.ca>.

BRIAN UNGER is the Executive Director of the Grid Research Centre and Professor of Computer Science at the University of Calgary. He was the founding President and CEO of iCORE, the "informatics Circle of Research Excellence", from 1999 though 2004. iCORE (<www.icore.ca>) is a not-for-profit corporation that funds research in the information and communication sciences at Alberta universities. In 2004, he received the CANARIE IWAY award for his outstanding contributions to Canada's information society. Dr. Unger received a Ph.D. in Information and Computer Science from the University of California, San Diego, an M.Sc. from the University of Southern California, and a B.Sc. from Loyola University, Los Angeles. His e-mail address is <unger@cpsc.ucalgary.ca>.