Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2001), Cincinnati, Ohio, IEEE Computer Society Press, 15-18 August 2001.

Effect of Event Orderings on Memory Requirement in Parallel Simulation

Y.M. Teo, B.S.S. Onggo and S.C. Tay
Department of Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543
email: teoym@comp.nus.edu.sg

Abstract

A new formal approach based on partial order set (poset) theory is proposed to analyze the space requirement of discrete-event parallel simulation. We divide the memory required by a simulation problem into memory to model the states of the real-world system, memory to maintain a list of future event occurrences, and memory required to implement the event synchronization protocol.

We establish the relationship between poset theory and event orderings in simulation. Based on our framework, we analyze the space requirement using an open and a closed system as examples. Our analysis shows that apart from problem size and traffic intensity that affects the memory requirement, event ordering is an important factor that can be analyzed before implementation. In an open system, a weaker event ordered simulation requires more memory than strong ordering. However, the memory requirement is constant and independent of event ordering in closed systems.

1. Introduction

Parallel simulation offers the potential to speedup simulations and to increase the size and complexity of models simulatable within a given time. However, additional memory is required by these event synchronization protocols that exploit parallelism. While much effort has been devoted to improve the runtime of parallel and distributed simulation, the amount of memory required for a simulation run is also an important issue to address. In conservative protocols [3,5], events are executed in the correct event time order. To prevent deadlock, null messages are introduced and it has been shown that the memory overhead for null messages can grow at an exponential rate [18]. On the other hand, the optimistic approach [12] executes simulation events without the consideration for event ordering and a rollback mechanism is used to correct out-of-order event execution. Additional memory is required to store the simulation states in anticipation of rollbacks.

There have been many studies on the space aspect of parallel simulation but most of them concentrate on managing the memory required to implement various synchronization protocols. In particular, conservative approach focuses on reducing the number of null messages, for example, the carrier-null mechanism [4] demand-driven method [1], and flushing method [18]. For optimistic approach, the focus is on delimiting the optimism thus constraining the memory consumption, and to reclaim memory before a simulator runs out of storage, for example, the use of event horizon in Breathing Time Bucket [17], the adaptive Time Warp [2], message send-back [11,13], the artificial rollback [15], adaptive memory management [6].

There are few publications on space analysis using analytical methods [13,22], and they concentrated on the CMB protocol only. Since each execution platform contains a fixed amount of memory, a performance model that analyzes the memory requirement of a simulation problem is necessary in understanding the potentials and limits of exploiting parallelism. This paper proposes a methodology to study the memory required to implement a simulation model of a real-world problem based on event ordering. Poset theory is used to formalize event ordering in simulation. The analysis in this paper is independent of the synchronization protocol used, and we study the effect of different event orderings on memory requirements.

The rest of this paper is organized as follows. Section 2 presents an overview of simulation modeling process. Section 3 defines our memory analysis framework. Section 4 introduces the poset theory and its relationship to event orderings in simulation. We formalize the three event orderings using poset theory and derive their memory requirement. The proposed framework is validated against experimental results obtained using two benchmarks, i.e., pipeline representing an open system, and PHOLD representing a closed system [8] in section 5. Section 6 presents our concluding remarks and highlight further works.

2. Modeling and simulation process

A computer simulation is a program that emulates the behavior of another system over time. Figure 1 shows a typical modeling and simulation process. The real-world system being emulated is called the physical system. Simulation modeling is a process of producing a simulation model of a physical system. There are three main modeling world-views, i.e., event-oriented, activity scanning, and process-oriented [10]. The physical system and simulation model are independent of the synchronization protocol selected for ensuring event causality at runtime. A simulation model is a logical model of a physical system that defines the input, parameters, output, system states, and other physical system components to be simulated. The next step involves translating the simulation model into either a sequential or parallel implementation (simulator program).

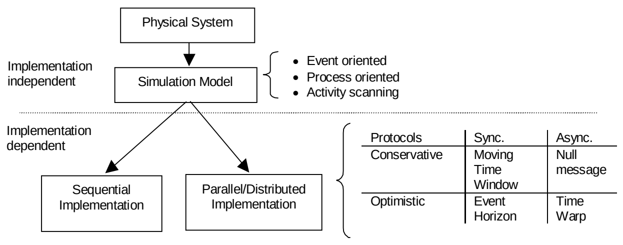

Figure 1. Typical modeling and simulation process

Most existing parallel discrete-event simulation (PDES) mandates a process-oriented world-view that forbids processes to have direct access to shared state variables [7]. In PDES, the physical system is viewed as being composed of a number of physical processes (PP) that interact at some fashion. Each physical process is modeled by a logical process (LP) with its own event list and clock, and interactions between physical processes are modeled by exchanging timestamp messages between the corresponding LPs. This LP model is independent of the protocol selected to synchronize event execution among the various local event lists.

3. Memory analysis model

We divide the memory required by a simulation into three main components. The first component, called M^{prob}, accounts for the memory required to model the states of the physical system. This is dependent on the characteristic of the system to be simulated. One of the most important system's states is the queue.

We measure M^{prob} by observing the queue size, and its upper bound is defined as the total maximum queue length, i.e., $M^{prob} \le \sum_{i=1}^{n} Q_i$, where Q_i is the maximum queue size at service center i, and n is the number of service centers. For simplicity, we count the number of entries in each queue. The actual amount of memory required depends on the data structure used in the implementation.

To ensure that all events are executed in the correct time order, future events are modeled by placing these events in an event list ordered by the event occurrence time. In parallel simulation, future events are stored in event lists that are distributed among LPs. At runtime, a synchronization protocol is required to ensure that the collection of events executed at each LP forms a correct global event sequence. These two components of memory required are called M^{ord} and M^{sync} respectively.

M^{ord} depends on the characteristic of system under study, i.e., event arrival and service rates, and the event ordering adopted. In sequential simulation, event ordering is ensured through a global future events list (FEL), and M^{ord} is defined as the maximum number of entries in the FEL. In parallel simulation, different event orderings can produce the same correct simulation results. The upper bound of M^{ord} is defined as the sum of all FEL lengths, i.e., $M^{ord} \le \sum_{i=1}^{n} FEL_i$, where FEL_i is the maximum FEL size at service center i (n is the number of service centers). The actual memory required depends on the FEL implementation.

Parallel simulation maintains several FELs. Consequently, if each processor executes events from its FEL independently, the execution order of all events across the system may not be correct. Hence, parallel implementation requires synchronization to ensure global event causality. M^{sync} accounts for the additional memory required to organize such synchronization. In optimistic protocol, memory is required for state saving in anticipation of rollbacks, message buffering to support message annihilation, GVT synchronization, etc. Null message in conservative protocol ensures event causality and requires additional memory at runtime. For sequential implementation, $M^{sync} = 0$.

It can be shown that $M^{prob} + M^{ord} + M^{sync}$ provides the memory lower bound. This paper focuses on memory analysis that is independent of the synchronization protocol implemented. A detail analysis of M^{sync} will be discussed in another paper.

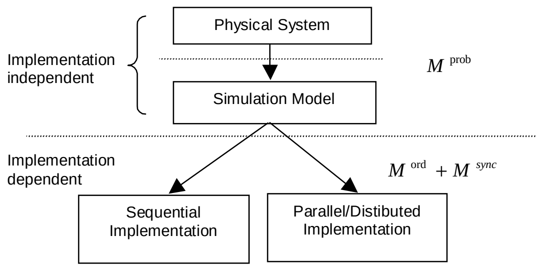
Physical System
Implementation independent
M^{prob}
Simulation Model
Implementation dependent
$M^{ord} + M^{sync}$
Sequential Implementation
Parallel/Distibuted Implementation

Figure 2. Components of memory model

Figure 2 depicts the relationship between the simulation modeling process and the memory components. Axiom 1 defines the memory required by a simulation. This axiom implies that any simulation implementation must require at least as much memory as is required by the physical system ($M > M^{prob}$).

AXIOM 1. The memory required to simulate a physical system (M) is $M = M^{prob} + M^{ord} + M^{sync}$.

The main benefit of our proposed methodology is that M^{prob} , M^{ord} , and M^{sync} can be analyzed independently:

- compares the M^{prob} between physical systems,
- understands the effect of event ordering on memory requirement (M^{ord}) for a simulation model before implementation.
- compares the synchronization costs (M^{sync}) of different parallel simulation protocols.

The following definitions of time by Fujimoto are used in this paper [10]. *Physical time* refers to time in the physical (real) system. For example, a ship arrived at 10:10 am yesterday. *Simulation time* is an abstraction used by the simulation to model physical time. In this paper, the simulation (occurrence) time of an event e is represented as ts(e). *Wall-clock time* refers to time used to execute the simulation program.

4. Types of event ordering

For a given physical system to be modeled M^{prob} must be the same regardless of how it is implemented. Thus, M^{prob} can be derived in the same way using queuing theory as proposed for parallelism [19]. On the other hand, M^{ord} is dependent on the event ordering adopted. In a simulation study, a set of events occurs in the physical system. The simulation model must execute these events in the right time order.

The ordering of elements of a set (or ordering for short) permeates our daily life. The main characteristics of ordering are that it is transitive, e.g. if 1<2 and 2<32 then 1<32, and it is anti-symmetric, e.g. if 1 is less than 2 then 2 is not less than 1. Mathematicians usually permit equality, hence the anti-symmetric property is rather weak, i.e. if x≤y and y≤x then x=y. Therefore, there are two definitions of partial order relation: *weak inclusion* and *strong inclusion*. The poset theory was built based on these properties. For ease of understanding, we reproduce the following four axioms from [16].

AXIOM 2. Weak inclusion definition: a relation R over $S \times S$ (where S is a set) is called a partial order if $\forall x \in S \bullet (x,x) \in R$ (reflexive), $(x,y) \in R$ and $(y,x) \in R$ then $x = y$ (weak anti-symmetric), $(x,y) \in R$ and $(y,z) \in R$ then $(x,z) \in R$ (transitive).

AXIOM 3. Strong inclusion definition: a relation R over $S \times S$ (where S is a set) is called a partial order if $\forall x \in S \bullet (x,x) \notin R$ (anti-reflexive), $(x,y) \in R$ then $(y,x) \notin R$, and vice versa (strong anti-symmetric), $(x,y) \in R$ and $(y,z) \in R$ then $(x,z) \in R$ (transitive).

AXIOM 4. Given a poset $\langle S, R \rangle$, two distinct element of S, i.e. x and y are comparable if either $(x,y) \in R$ or $(y,x) \in R$. Similarly, x and y are incomparable (denoted by $x \parallel y$) if neither $(x,y) \in R$ nor $(y,x) \in R$.

AXIOM 5. A relation R over $S \times S$ (where S is a set) is called a total order if for all $x, y \in S$ then x and y are comparable.

Negger and Kim [16] showed that weak and strong inclusion definitions were transformable by introducing operators $=$ and $\neq$. However, in a simulation study, the set of events is our main interest and we are not interested in comparing an event with itself. Therefore, we consider only strong inclusion. For the rest of this paper, the terms partial order refers to the strong inclusion definition.

The comparability between elements of a set is defined in axiom4, and axiom 5 defines a total order, i.e. an order in which all of its elements are comparable, for example: relation < on a set of natural numbers is a total order, however relation "descendant" on a set of people is not a total order. We explain in the next section how we formalized event ordering using poset theory.

4.1 Event ordering in simulation

Lamport introduced two event-ordering policies, i.e. happens before (partial) event ordering and total event ordering [14]. We express these two policies using poset theory in axiom 6 and 7.

AXIOM 6. Let $\langle E, <_{par} \rangle$ be a poset, where E is a set of events. Under partial event ordering, e_1 happens before e_2 (denoted by $e_1 <_{par} e_2$), if:

- $\neg(e <_{par} e)$ *, for any event $e \in E$;*
- *e_1 and e_2 are events in the same process, and e_1 comes before e_2 ;*
- *e_1 is the sending event in process P_i , and e_2 is the correspond receiving event in process P_j ;*
- *if $e_1 <_{par} e_2$ and $e_2 <_{par} e_3$ then $e_1 <_{par} e_3$.*

The first property satisfies the anti-reflexive requirement of a partial order given in axiom 3. The next two properties guarantee that if $e_1 <_{par} e_2$ is true then $e_2 <_{par} e_1$ is false, and vice versa; thus following the strong anti-symmetric requirement of a partial order. The last property shows that partial event order is transitive; therefore, partial event order matches the partial order definition in the poset theory.

AXIOM 7. Let $\langle E, <_{tot} \rangle$ be a poset, where E is a set of events. Assume that each $e \in E$ can be stamped with a simulation time (denoted by $ts(e)$). Under total event ordering, e_1 happens before e_2 (denoted by $e_1 <_{tot} e_2$), if:

- $ts(e_1) < ts(e_2)$ *, or*
- $ts(e_1) = ts(e_2) \wedge e_1$ *has higher priority than* e_2 .

Each event $e \in E$ has a timestamp, therefore for any two events $e_1, e_2 \in E$ either $ts(e_1) < ts(e_2)$, $ts(e_1) > ts(e_2)$ or $ts(e_1) = ts(e_2)$ is true. If $ts(e_1) < ts(e_2)$ or $ts(e_1) > ts(e_2)$ then both events are comparable. The second property guarantees that all events are comparable even if the timestamp is equal, so long as all events that have the same timestamp have different priorities. Therefore, total event ordering matches the total order poset theory definition in axiom 5.

AXIOM 8. $\langle E, <_1 \rangle$ and $\langle E, <_2 \rangle$ are posets where E is a set of events. $<_1$ and $<_2$ are two distinct event orders. Let $CONC(<_1)$ and $CONC(<_2)$ be a set of concurrent (incomparable) events under $<_1$ and $<_2$ respectively. Event order $<_1$ is stronger than $<_2$, if and only if $CONC(<_1) \subset CONC(<_2)$.

An event order $<_1$ is stronger than event order $<_2$ if there is a non-empty set of incomparable events under $<_2$ that is comparable under $<_1$.

AXIOM 9. Let $\langle E, <_{ts} \rangle$ be a poset, where E is a set of events. Assume that each $e \in E$ can be stamped with a simulation time (denoted by $ts(e)$). Under timestamp event ordering, e_1 happens before e_2 (denoted by $e_1 <_{ts} e_2$), iff $ts(e_1) < ts(e_2)$.

Fujimoto defined timestamp order to eliminate some temporal anomalies [10]. We formalize this as *timestamp event order* to enhance our analysis in axiom 9. Timestamp event ordering relaxes the properties of total event ordering by ignoring the second property in axiom 7. This implies that $CONC(<_{tot}) \subset CONC(<_{ts})$, and thus timestamp event order is weaker than total event order. On the other hand, partial event ordering relaxes the property of timestamp event ordering. Even if $ts(e_1) < ts(e_2)$, e_1 and e_2 may not be comparable under partial event ordering. This implies that $CONC(<_{ts}) \subset CONC(<_{par})$, and thus timestamp event order is stronger than partial event order. This has also been reported by [9] without formal proof.

In summary, we have three event orderings that differ in their strength. This provides a framework to study the effect of event ordering's strength on memory requirement. Total event ordering is synonymous to a sequential simulation with one FEL, i.e. $M^{ord}(<_{tot})$ = $\max(FEL)$. Therefore the upper bound for $M(<_{tot})$ is given in equation 1.

$$M(<_{tot}) \leq \sum_{i=1}^{n} Q_i + \max(FEL) \qquad (1)$$

Timestamp and partial event orderings can be viewed as two different orderings suitable for parallel implementations. The upper bound for timestamp and partial event orderings are $M(<_{ts})$ and $M(<_{par})$ respectively.

$$M(<_{ts}) \leq \sum_{i=1}^{n} Q_i + \sum_{i=1}^{n} FEL_i(<_{ts}) \qquad (2)$$

$$M(<_{par}) \leq \sum_{i=1}^{n} Q_i + \sum_{i=1}^{n} FEL_i(<_{par}) \qquad (3)$$

5. Experimental results

Our proposed methodology is tested using two benchmarks. Figure 3a shows a linear pipeline system. This *open system* is parameterized by the number of service centers (n) and traffic intensity (ρ). Inter-arrival and service rates are assumed to be exponentially distributed.

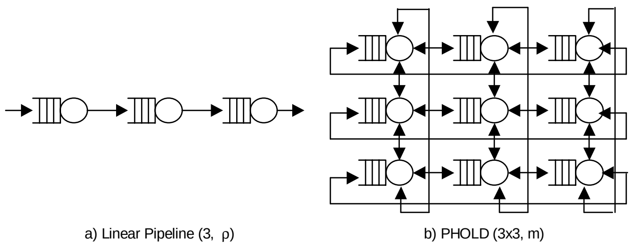

Figure 3. Open and closed system benchmarks

The second benchmark called PHOLD [8] is shown in figure 3b. This benchmark is commonly used in parallel simulation studies and represents a *closed system*. We assume (i) a timestamp increment function that is exponentially distributed with a mean of 1, (ii) a movement function that is uniformly distributed and each service center has four neighbors, and (iii) an initial configuration with messages distributed equally among the service centers. In our experiments, we vary the number of service centers ($n \times n$) and the message population per service center (message density or m).

The two benchmarks are implemented using CSIM, a simulation library in C language by Watkins [21]. Instrumentation codes were added to the CSIM library to monitor the *Conditional Event List (CEL)* and *Future Event List (FEL)*. Our experiment results are based on the average of five replications and simulation duration of 10,000 time units.

5.1 Pipeline

Table 1 shows the memory usage and parallelism for varying number of service centers (n) and traffic intensity (ρ). We observe that n has a linear effect on M^{prob}, and ρ has an exponential effect on M^{prob}, i.e., M^{prob} is dependent on the characteristic of problems.

$M^{ord}(<_{tot})$ is measured by observing the maximum FEL size during the simulation duration. $M^{ord}(<_{ts})$ and $M^{ord}(<_{par})$ are derived by summing up the maximum size for all FELs. The results show that M^{ord} differ among event orderings being used. Lemma 1, 2 and conjecture 1 explains the characteristic for each event ordering.

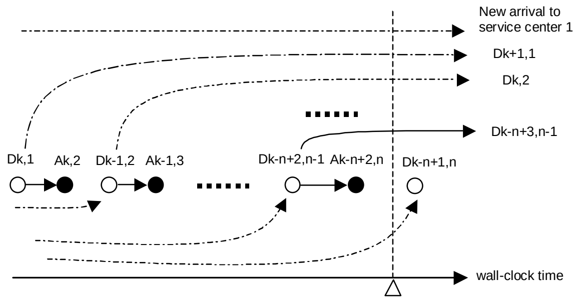

Figure 4. Example of a worst-case scenario in total event ordering

Parameters		Memory usage				Parallelism	
n	ρ	M^{prob}	M^{ord}			$<_{ts}$	$<_{par}$
			$<_{tot}$	$<_{ts}$	$<_{par}$		
4	0.2	28.2	6	8	17	1.6	4
4	0.4	51.4	6	8	27	1.6	4
4	0.6	90.6	6	8	45	1.6	4
4	0.8	195.8	6	8	90	1.7	4
8	0.2	58.8	10	16	37	1.8	8
8	0.4	107.0	10	16	62	1.9	8
8	0.6	184.6	10	16	102	1.9	8
8	0.8	379.8	10	16	210	2.0	8
12	0.2	83.4	14	24	55	1.9	12
12	0.4	158.2	14	24	93	2.0	12
12	0.6	275.0	14	24	161	2.1	12
12	0.8	574.6	14	24	325	2.1	12
16	0.2	113.4	18	32	76	2.0	16
16	0.4	211.8	18	32	130	2.1	16
16	0.6	358.8	18	32	213	2.2	16
16	0.8	769.8	18	32	456	2.3	16

Table 1. Linear Pipeline (n, ρ)

LEMMA 1. $1 \le M^{ord}(<_{tot}) \le n+2$.

PROOF. To be operational, there must be at least one arrival event in the FEL. Thus, the lower bound is proven. Figure 4 shows one possible worst-case condition ($A_{k,n}$ refers to the kth arrival on service center n, and similarly D refers to departure event). Assume that all service centers are busy and have at least one job in their queues. Further, assume $D_{k,1}$, $D_{k-1,2}$, …, $D_{k-n+2,n-1}$, and $D_{k-n+1,n}$ are the first n elements in FEL. Eventually $D_{k,1}$ is executed and schedules $A_{k,2}$ and $D_{k+1,1}$. $A_{k,2}$ has the same timestamp as $D_{k,1}$ so $A_{k,2}$ is the first element in the FEL, and $A_{k,2}$ is executed and produces no event. The next event executed, $D_{k-1,2}$ produces $A_{k-1,3}$ and $D_{k,2}$. The cycle is repeated until the simulator executes $D_{k-n+2,n-1}$ and schedules $A_{k-n+2,n}$ and $D_{k-n+3,n-1}$. At this wall-clock time (denoted by the vertical line), there are $n+2$ events in the FEL, i.e. $A_{k+1,1}$, $D_{k+1,1}$, $D_{k,2}$, …, $D_{k-n+3,n-1}$, $A_{k-n+2,n}$ and $D_{k-n+1,n}$. This is reflected from the number of cuts between the arrow lines and the vertical line. This is the worst case because all service centers can schedule at most two events. However, with zero transit delay, the arrival to the next service center will be executed immediately. This implies that all service centers can have at most one outstanding event, except for the first service center and the current service center whose event is being executed (two outstanding events each). Since total order is used, then only one event (one active service center) can be executed at any time. Therefore, the upper bound is $n+2$.

LEMMA 2. $1 \le M^{ord}(<_{ts}) \le 2n$

PROOF. There must be at least one event in the FEL. The worst-case condition happens when all jobs leave the service centers at the same time (figure 5). The arrow from a pointing to b represents that a schedules b. The dashed lines represent processes. Unshaded circles denote departure events, and the shaded ones represent arrival events. There must be one arrival event in the LP_1's FEL (the leftmost arrow). Since LP_1 is busy and its queue length is greater than zero, once LP_1 executes a departure event, the next departure event is scheduled. Hence, LP_1's FEL contains at most two events at any particular time. The execution of departure event in LP_1 schedules an arrival event in the LP_2. Assume there is no transit delay and a departure event has been scheduled at the same simulation time. Hence, there are two simultaneous events with the *same* timestamp at LP_2. Assume departure event has higher priority than arrival event, i.e., departure event will be executed first (consequently the arrival event will be put in the FEL). Since LP_2 is busy and its queue length is greater than zero, then upon executing the departure event, LP_2 schedules another departure event. Hence, the remaining LPs also contain at most two events at any time. Thus, the upper bound is $2n$.

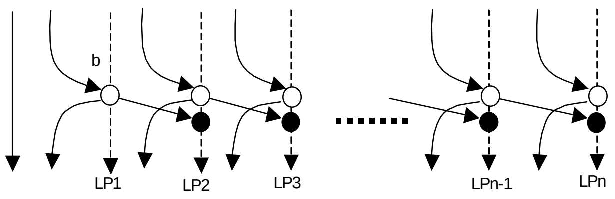

Figure 5. Example of a worst-case scenario in timestamp event order

CONJECTURE 1. $1 \le M^{ord}(<_{par}) \le 2+f(n,\rho)$, *where* $f(n,\rho)$ *is an unknown function.*

Figure 6a shows that $M^{ord}(<_{par})$ linearly correlated with n. On the other hand, a second order term of ρ is obviously present (figure 6b).

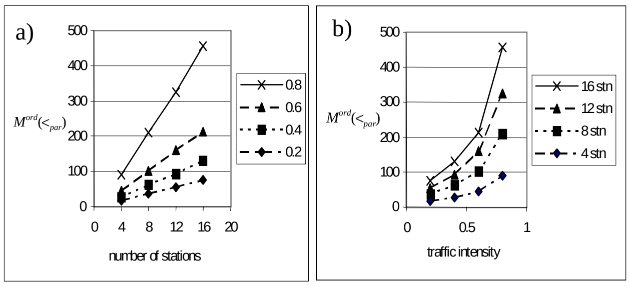

Figure 6. Pipeline (n, ρ) - $M^{ord}(<_{par})$ behavior

As discussed earlier, LP_1 has at most two events. However, the remaining LPs could have more events. Consider figure 7 where the simulation clock of LP_1 has progressed ahead of LP_2. Assume that LP_2 is still executing the departure event, and thus all arrival events from LP_1 will be put on its FEL. Therefore, the number of events in the LP_2's FEL (and also other LPs) depends on the traffic intensity (ρ). The fact that a second order term of ρ affects the $M^{ord}(<_{par})$ is similar to the relation between ρ and the average queue size. The exact model of the $M^{ord}(<_{par})$ can be found using any function approximation methods such as multiple linear regression. In addition, the pipeline is homogenous, so that the expected maximum FEL size for each LP is the same. This explains why $M^{ord}(<_{par})$ is linearly correlated with n (figure 6a).

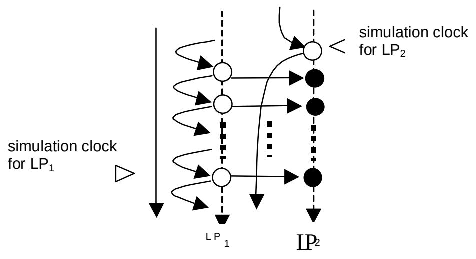

Figure 7. Snapshot of events execution using partial event ordering

THEOREM 1. For an open system, a weaker event order requires more M^{ord}.

PROOF. The number of events in the FEL depends on the rate of event arrival and the rate of event execution, and the event ordering adopted. However, event arrival rate is independent of event ordering. An event execution may schedule new events or may not schedule any events, i.e., when there is no job in the queue. When the traffic intensity is moderate to high, event execution usually schedules new events. A weaker event order has more flexibility in executing events concurrently. Hence, the rate of event execution will be higher and it needs more memory to hold the scheduled events in the event list(s).

Table 1 shows that $M^{ord}(<_{tot}) < M^{ord}(<_{ts}) < M^{ord}(<_{par})$.

Parallelism is discussed in detail in a separate paper, but is included in table 1 for validation purposes. The parallelism for partial event order is similar to that reported in [5]. We observe that the strength of an event ordering affects the parallelism. A weaker event order produces more incomparable events that can be executed concurrently. Therefore, a weaker event order has potential for more parallelism at the expense of more memory (theorem 1).

5.2 PHOLD

PHOLD is a closed system and thus the number of jobs in the system is bounded. Lemma 3 shows that without considering the overhead of synchronization, the amount of memory required is independent of event ordering.

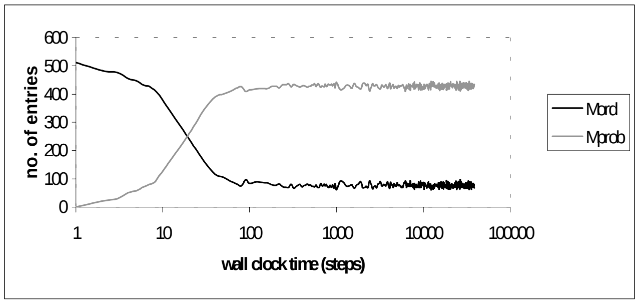

Figure 8. PHOLD (8×8, 8) - memory profile for partial event order

Parameters				Parallelism	
$n\times n$	m	$M^{prob+ord}$	$\lim_{t\to\infty} Q(t)$	$<_{ts}$	$<_{par}$
4×4	1	16	13.0	2.0	6.4
4×4	8	128	119.4	2.0	8.2
4×4	16	256	245.6	2.0	8.3
8×8	1	64	42.8	2.0	22.1
8×8	8	512	466.8	2.0	29.7
8×8	16	1024	973.3	2.0	30.5
16×16	1	256	152.2	2.1	83.2
16×16	8	2048	1841.2	2.1	114.4
16×16	16	4096	3872.0	2.1	118.1

Table 2. PHOLD (n×n, m) - experiment result

LEMMA 3. *Assuming protocol independence, an $n\times n$ PHOLD requires $n\times n\times m$ unit memory, where $n\times n$ is the node size, and m is the message population (density).*
PROOF. The maximum FEL size occurs at the initialization step where each node (LP) schedules m event arrivals. There are $n\times n$ nodes, so the total FEL size is $n\times n\times m$. On the other hand, the queue size is minimum because initially there is no job in the queues. Hence the total memory required for a closed system (M) is bounded, i.e. $n\times n\times m$. Figure 8 shows the changes of M^{prob} and M^{ord} overtime.

As shown in table 2, the queue size at steady state (i.e. $\lim_{t\to\infty} Q(t)$) depends on the characteristic of the problem and event ordering has no effect on it. The memory required for a closed system is given in theorem 2**.**

THEOREM 2. *For a closed system, the total memory required (M) is $M^{prob+ord} + M^{sync}$, where $M^{prob+ord}$ is the amount of memory required to order event occurrences and to model the queue of service centers, and M^{sync} is the amount of memory required to implement the synchronization protocol.*
PROOF. In an open system, queue size and FEL size are independent. For a closed system as shown in lemma 3, the queue and FEL sizes are dependent, i.e., $M^{prob} + M^{ord} = M^{prob+ord}$ (where $M^{prob+ord}$ is a constant). Therefore, the total memory required for a closed system is $M = M^{prob+ord} + M^{sync}$.

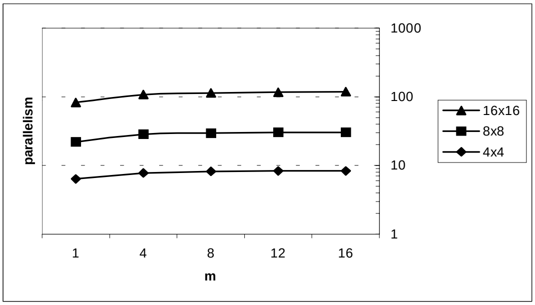

Figure 9. PHOLD (n×n, m) – parallelism under partial event order

For parallelism, the effect of event ordering on PHOLD is similar to the pipeline benchmark, i.e., parallelism is gained by relaxing the event ordering. However, in PHOLD, the gain in parallelism does not affect the total memory required. Figure 9 shows the parallelism gain based on partial event order. Our result is similar to that reported in [8]. Additional parallelism can also be exploited by increasing the message density. The asymptotic behavior of parallelism is further analyzed in [20].

6. Conclusions and further works

We propose a formal approach to analyze space requirement of a simulation based on the amount of memory required to store the states of the physical system, event ordering and to implement the event synchronization protocol. The event orderings are formalized using poset theory.

For a given simulation problem, this enables each memory components to be analyzed independently. The advantage of this approach is that space analysis can be carried out prior to implementation, and the overhead of parallel synchronization protocols implementation can be separately quantified. We apply the analysis to an open pipeline system and a closed PHOLD system. The results are summarized in table 3.

Benchmark	Total Event Order	Timestamp Event Order	Partial Event Order
Pipeline(n, ρ)			
M^{prob}	$\le g(n,\rho)$	$\le g(n,\rho)$	$\le g(n,\rho)$
M^{ord}	$1..n+2$	$1..2n$	$1..f(n,\rho)$
PHOLD(n×n, m) $M^{prob}+M^{ord}$	$n\times n\times m$	$n\times n\times m$	$n\times n\times m$

Table 3. Summary of memory utilization

We have shown that the characteristic of a problem determines M^{prob}. M^{ord} is dependent not only on the characteristic of the problem, but also the strength of the event ordering. For an open system, we provide a formal prove that a weaker event order needs more memory. However, the memory required ($M^{prob+ord}$) by a closed system is constant. We also observe that greater parallelism can be exploited by relaxing the event ordering. This phenomenon is in contrast to the open system, where greater parallelism is found to require more memory.

Acknowledgement

This research is supported by a grant from the Ministry of Education and PSA Corporation under grant RP960715.

References

[1] W.L. Bain, and D.S. Scott, "An Algorithm for Time Synchronization in Distributed Discrete Event Simulation", Proc. of the SCS Multi-conference on Distributed Simulation, 19, 3, February 1988, pp.30-33.

[2] D. Ball, and S. Hoyt, "The Adaptive Time-Warp Concurrency Control Algorithm", Proc. of the SCS Multi-conference on Distributed Simulation, 1990, pp.174-177.

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

[4] W. Cai, and S.J. Turner, "An Algorithm for Distributed Discrete Event Simulation – The Carrier Null Message Approach", Proc. of the SCS Multi-conference on Distributed Simulation, January 1990, pp. 3-8.

[5] K.M. Chandy, and J. Misra, "Distributed Simulation: a Case Study in Design and Verification of Distributed Programs", IEEE Trans. on Software Engineering, 5, 5, September 1979, pp. 440-452.

[6] S.R. Das, and R.M. Fujimoto, "An Adaptive Memory Management Protocol for Time Warp Parallel Simulation", ACM SIGMETRICS Conf. on Measurement and Modeling of Computer System, 1994, pp. 201-210.

[7] R.M. Fujimoto, "Parallel Discrete Event Simulation", Communication of the ACM, 33, 10, 1990, pp. 31-53.

[8] R.M. Fujimoto, "Performance of Time Warp under Synthetic Workload", Proc. of the SCS Multi-conference on Distributed Simulation, 22, 1, January 1990, pp. 23-28.

[9] R.M. Fujimoto, and R.M. Weatherly, "Time Management in the DoD High Level Architecture", Proc. of the 10th Workshop on Parallel and Distributed Simulation, 1996, pp. 60-67.

[10] R.M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley & sons, inc., 2000.

[11] A. Gafni, "Rollback Mechanisms for Optimistic Distributed Simulation Systems", Proc. of the SCS Multi-conference on Distributed Simulation, 19, 3, 1990, pp. 61-67.

[12] D.R. Jefferson, "Virtual Time", ACM Trans. on Programming Language System, 7, 3, July 1985, pp. 404-425.

[13] D. Jefferson, "Virtual Time II: Storage Management in Distributed Simulation", Proc. 9th annual ACM symposium on Principles of Distributed Computation, 1990, pp. 75-90.

[14] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communication ACM, 21, 7, July 1978, pp. 558-565.

[15] Y.B. Lin, and B.R. Preiss, "Optimal Memory Management for Time Warp Parallel Simulation", ACM Trans. on Modeling and Computer Simulation, 1, 4, October 1991, pp. 283-307.

[16] J. Negger, and H.S. Kim, *Basic Posets*, World Scientific Publishing, 1998.

[17] J.S. Steinmann, "SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation", International Journal on Computer Simulation, 2, 1992, pp. 251-286.

[18] S.C. Tay, *Parallel Simulation Algorithms and Performance Analysis*, PhD Thesis, National University of Singapore, 1998.

[19] Y.M. Teo, H. Wang, and S.C. Tay, "A Framework of Analyzing Parallel Simulation Performance", Proc. of the 32nd Annual Simulation Symposium, IEEE Computer Society Press, San Diego, USA, April 1999, pp. 102-109.

[20] Y.M. Teo, and B.S.S. Onggo, *A Methodology for Space Analysis of Parallel Simulation*, Technical Report, Dept. of Computer Science, National University of Singapore, TRA6/01, June, 2001.

[21] K. Watkins, *Discrete Event Simulation in C*, McGraw-Hill, 1992.

[22] Y.C. Wong, and S.Y. Hwang, "Prediction of Memory Consumption in Conservative Parallel Simulation", Proc. 9th Workshop on Parallel and Distributed Simulation, 1995, pp. 199-202.