Predicting the performance of synchronous discrete event simulation systems

Jinsheng Xu and Moon Jung Chung
Department of Computer Science
Michigan State University
{xujinshe,chung}@cse.msu.edu

ABSTRACT

In this paper we propose a model to predict the performance of synchronous discrete event simulation. The model considers parameters including the number of active objects per cycle, event execution granularity and communication cost. We derive a single formula that predicts the performance of synchronous simulation.

We have benchmarked several VHDL circuits on SGI Origin 2000. The benchmark results show that the prediction model explains more than 90% of parallel simulation execution time. We also measure the effect of computation granularity over performance. The benchmark results show that although higher granularity can have better speedup because of dominance of computation over communication, the computational granularity cannot overshadow the inherent synchronization cost. This model can be used to predict the speed-up expected for synchronous simulation, and to decide whether it is worthwhile to use synchronous simulation before actually implementing it.

Keywords
Parallel Discrete Event Simulation, Synchronous Simulation, Synchronization Cost, Communication Cost, Granularity, Performance.

1. Introduction

Synchronous simulation is one of the simplest and easiest parallel simulation protocols. It, however, may suffer from poor performance because only events with the smallest timestamp can be executed. For every cycle, processors are synchronized, making processors wait until all other processors finish their event execution. Frequent synchronization makes synchronous simulation more prone to the situation when the load is unbalanced in certain time stamps [9]. This does not mean that synchronous simulation should not be used. Except from additional communication cost, synchronous simulation has little overhead compared to a sequential simulation. Although conservative and optimistic simulations can exploit more parallelism, they have much more overhead than synchronous simulation, which can make the overall performance worse than synchronous simulation. Soule and Gupta evaluated the Chandy-Misra algorithm [3] for digital logic simulation. They found that overhead of Chandy-Misra algorithm overwhelm its advantage and the performance is about three times slower than traditional parallel event-driven algorithm [17].

In this paper we propose a model to predict the performance of synchronous simulation. Parameters of the performance model can be obtained from the circuit characteristics and parallel computer systems. From this model and parameters, we can decide whether it is worthwhile to use synchronous simulation before actually implementing it. Synchronous simulation cost includes computation cost, communication cost and idle time waiting for others to finish event execution. This waiting is caused by load unbalancing in some time steps. The cost of longest executing processor is considered as the computation cost of each time step. For other processors that have shorter execution time will have to wait for the longest executing processor before all of them can proceed to next time stamp. After each time step, all the processors exchange message. This parallel programming model can be categorized as a BSP programming model [18].

Agrawai and Chakradhar [1] considered only load unbalancing factor for synchronous simulation. They gave a statistical model for synchronous simulation. Given a system with uniform granularity of event execution and with random partitioning, an object is active with a probability of *activity rate*. The number of independent random variables is the same as the number of objects in the system. At each cycle, different processors have different number of active objects while the cost of the cycle is determined by the maximum order statistics of binomial random variables. They proposed a performance model based on the number of objects together with the *activity rate.* They compared the theoretical prediction with the benchmark for several circuits. However, they did not consider the communication cost. In many cases, especially when the computational granularity is small, communication cost should be considered.

In this paper, we first propose a different statistical model to predict the distribution of load and the effect of load balancing factor. Our model is based on multinomial distribution, and uses *average number of active object per cycle*. To predict the performance of synchronous simulation, we have developed a model based on the multinomial distribution model and communication model.

The communication overhead of modern message passing parallel computers includes the operating system overhead of dividing the messages into packets, adding header information to the packets and then putting the packets into hardware. Modern parallel computers with advanced routing technique and high bandwidth, communication overhead takes most part of total communication cost.

We have benchmarked several VHDL circuits on SGI Origin 2000. The prediction result is close to benchmark results. Circuits with large average active number of object per simulation cycle have better performance because it has larger multiplication factor from the maximum order statistics of multinomial random variables. We also measured the effect of computation granularity over the performance. The benchmark results show that although higher granularity can have better speedup because of dominance of computation over communication, the deciding factor over performance is still active number which can both reduce the load unbalancing factor and the communication cost. This can be seen from the result that when the granularity is higher than some value

This work was supported in part by HPCMP program F33615-96-C-1913 and NSF DMI-0075396.

there is little gain in speedup. The computational granularity can overshadow the communication cost but not the inherent synchronization cost.

The paper is organized as follows: In section two we proposed multinomial distribution model. In section three we proposed prediction model due to load unbalancing factor. In section four we discussed the communication model. In section five we derived our final performance formula for synchronous simulation based on our multinomial distribution model and communication model. We also compared the performance model with empirical result.

2. The Model

The discrete event system we are interested in is all objects in the system have the same granularity. This is applicable to the circuit simulation where each object is a gate and there is not significant difference between execution time of two gates. The other assumption of the model is random partitioning. The objects in the system are randomly partitioned to different processors. This allows us to apply statistical methods to estimate the cost of the simulation. Other partitioning algorithms can reduce the cost of message passing by making as many events internal to its own partition as possible. This will reduce the randomness of object distribution and will potentially cause less accurate estimations.

Figure 1 shows the synchronous algorithm our simulation is based on. The algorithm runs with parallel message passing system. GVT is computed by piggybacking the next smallest timestamp in the event message sent per cycle. This algorithm has the advantage of removing the cost of broadcasting and reducing cost, but it needs to send a null message to other processors when there is no event for some cycles.

```
Each processor executes the following code:

while (GVT < MAXGVT) {
  Execute all the events with GVT;
  Send events to other processors with LVT attached;
  Receive events from other processors;
  Compute the GVT with piggybacked LVT;
}
```

Fig. 1. Synchronous algorithm our model based on

This is a very simplified synchronous simulation model. There are optimizations to this model. We are interested in this model because it enables us to derive a nice performance prediction formula and we believe other complex models can be reduced to this simple model without significant loss of performance.

2.1 Multinomial Distribution

In this paper, we proposed a different statistical model for synchronous simulation. We emphasized the importance of the average number of active objects per simulation step rather than the activity rate. A system with a large number of objects with a low activity rate may perform better than a system with a small number of objects but with a high activity rate because the system with a large number of objects with a low activity rate has a greater average number of active objects per cycle. For simplicity, we will call the average number of active objects per cycle, the *active number*. Based on the *average number of active objects per cycle,* we proposed a multinomial distribution model to model the distribution of load and the effect of the load-balancing factor. An active object has the same probability to be at any one of the partitions. There are an *active number* of active objects per cycle and they are independent random variables. The cost of each cycle is determined by the maximum order statistics of the counting variables that are the number of active objects distributed to different processors. The multinomial distribution model reduces the number of parameters from two in the binomial distribution model to just one variable – the *active number*. The prediction is computationally easier for the multinomial distribution model because the number of active objects is usually much smaller than the total number of objects in the system. We compared the two models and the results were close. In part three of this paper, we discuss the multinomial distribution model and compare simulation results with the benchmark results for several VHDL circuits.

The *active number* for one cycle is different than that of another cycle. The benchmark shows that the *active number* varies from cycle to cycle significantly in circuit simulation. For the simplicity of prediction formula, we are not considering the distribution of the *active number* over the cycles. We use the *average active number* to predict performance. The accuracy of the prediction shows that the distribution factor of the *active number* in circuit simulation does not have significant contribution to the final performance.

The *active number* is an inherent property of a circuit and other discrete event systems. It is the most important factor for our performance prediction model. There is question about how we can acquire this number. If we already know the *activity rate* of the circuit, we can estimate the *active number* by multiplying the total number of objects with the *activity rate*. If we do not have the *activity rate*, we can do sequential simulation with large enough cycles to estimate the value of the *active number*.

The general idea that allows us to predict the performance of synchronous simulation is based on the following simple observation. If the objects are partitioned randomly enough and the behavior of the discrete event system is random enough, we can assume that the active objects for each cycle have multinomial distribution over partitions. An active object has the same probability to be at any of the partitions. There are *active number* of independent random variables each cycle. The cost of each cycle is determined by maximum order statistics of the counting variables that are number of active objects distributed to different processors. If the number of cycles is large enough, the average cost of each cycle will be close to its expected value.

These are the terms we will use in this paper.
Notations and terms
$o_1,o_2,...,o_n$: a list of n objects in the discrete event system
$t_1,t_2,...,t_{numcycle}$: a series of discrete time and *numcycle* is the number of cycles.
$A[t_i]$: Number of objects that is active at time t_i
A : The average of $A[t_i]$, $i=1,2,...,max$
$S = A[t_1] + A[t_2] + ... + A[t_{max}]$: Total number of activity
p : Number of partitions
cycle : a cycle is the duration of simulation for each discrete time t_i

We assume the following three conditions.
Condition 1:
$A[t_i] > 0$, for all i
$A[t_i]=0$ means there is no object active at time t_i. It is not necessary to have discrete time t_i.
$S >> A[t_i]$
This means that simulation has a wide range of discrete time and does not finish in a few cycles.

Condition 2: Randomness

At each cycle, an object has equal probability of being active with all other objects.

Condition3: Uniform granularity

Each object takes a unit time to execute an event.

3. Prediction Model due to Load Balancing factor

In this section we will derive the computation cost under the multinomial distribution model and compare it with the benchmark result. We also propose a multiplication factor, which describes the degree of load unbalancing.

3.1 Total computation cost

The cost of each cycle is the maximum number of active objects among all partitions. We denote the cost of each cycle by $C[t_i] = max(A_k[t_i])$, where $A_k[t_i]$ is number of objects active at partition k and $A_1[ti] + ... + A_p[t_i] = A[t_i]$. p is the number of processors the simulation is running on. The cost of the whole simulation is $C[t_1] + C[t_2] + ... + C[t_{max}]$. Since we have assumed that each active object has equal probability of being present at any of the partitions, $A[t_i]$ objects that are active at time t_i, are randomly distributed to p partitions. Therefore the cost of each cycle can be estimated by the expected value of the maximum order statistics of counting variables that are the number of active objects distributed to p partitions. The cost of each cycle is a function of $A[t_i]$ and p which denote as $E(A[t_i] ,p)$. The closed formula for function $E(A[t_i] ,p)$ may not be easy to obtain. Instead of trying to find the closed formula for the function, we can use simulation to get the result of the function with good accuracy. Total simulation costs due to execution of events should be $Numcycles*E(A,p)$, where A is the average of $A[t_i]$.

3.2 Multiplication Factor

We want to measure the degree of load unbalancing with different average number of active objects per cycle. We now define the multiplication factor as follows. Suppose that the average activity number of objects per cycle of a circuit is A and the circuit is partitioned to P parts. Suppose $E(A,p)$ is expected value of maximum order statistics of counting variables that are number of objects distributed to P partitions. We define multiplication factor M as:

$$M(A,P) = \frac{1}{E(A,P) * P / A}$$

M is a function of A and P. Table 1 shows the value of the multiplication factor over different values of A and P. The range of multiplication factor is from 0 to 1 exclusive. The multiplication factor shows the efficiency of parallel simulation. Larger multiplication factor means lower degree of load unbalancing. The multiplication factor increases as A increases and P decreases. Therefore, the synchronous parallel simulation will be more efficient with a larger A and smaller P.

Multiplication factor					
A/P	2	4	8	16	32
1	0.50	0.25	0.13	0.06	0.03
10	0.80	0.60	0.41	0.27	0.17
100	0.93	0.83	0.70	0.57	0.43
1000	0.98	0.94	0.87	0.81	0.72
10000	0.99	0.98	0.96	0.93	0.89

Table 1. Multiplication factor for different *A* and *P*

The speedup can be defined using multiplication factor as:

$$S(A,P) = P*M(A,P)$$

Table 2 shows the speedup for the A and P of above table.

Speedup					
A/P	2	4	8	16	32
1	1.00	1.00	1.00	1.00	1.00
10	1.61	2.38	3.31	4.29	5.29
100	1.85	3.31	5.63	9.07	13.73
1000	1.95	3.76	6.94	13.02	23.06
10000	1.98	3.92	7.69	14.95	28.57

Table 2. Speedup table for different *A* and *P*

Table 1 and 2 show that for large A there is little loss of efficiency, but for small A the loss of efficiency is significant. For example when A is 100 and P is 32, the load unbalancing will cost 57% of the ideal speedup.

Comparisons with Benchmark

Theoretical vs. Empirical

We ran several VHDL circuits sequentially up to some large enough time limits. We are able to get the total number of executions and number of cycles. The benchmark circuits used are based on VHDL Description of ISCAS circuit. The activity number is derived from the quotient of number of executions divided by number of cycles. Table 3 shows the results for s35932_structural, s15850_structural, s9234_structural and multi32_arraymulti32.

Number of Executions, Cycles and Activity Number				
Circuit	S35932	S15850	S9234	Multi32
Number of Executions	3294920	257457	143965	2303858
Cycles	3499	4415	3258	1998
Avg. Activity Number	942	58	44	1153

Table 3. Number of Executions, cycles and Average Activity Number

When these circuits are partitioned to a number of processors, the cost of each cycle is the maximum number of executions among these processors. The sum of these maximum numbers of executions for all cycles is the total amount of the computation. We can estimate the total amount using our multinomial distribution model with the average activity number.

To see how accurate the estimation is, we randomly partitioned the objects into a different number of parts where each part had the same number of objects. We then recorded the number of executions for each of these parts and recorded the sum of then maximum number of executions of all parts for all cycles. This experiment was done sequentially and there was no communication cost involved. Table 4 shows comparisons of the benchmark computation cost and the estimation with the different number of partitions. The estimation is quite close to the benchmark. Most of the estimations have errors less than 2%. This shows that many circuits have random behaviors that fit well to our multinomial distribution model. We compared the result of the multinomial distribution with binomial distribution. They both have good estimations of the benchmark. The number of independent trials for multinomial distribution model is much less than that of binomial distribution model because the number of independent variables of the multinomial distribution model is much less than that of the binomial distribution model. A more important advantage of the multinomial distribution model is it only depends on one parameter – the *average number of active object per cycle*. The binomial distribution model needs the *activity rate* and the total number of objects.

s35932					
PE	Empirical	Multinomial	Binomial	Multinomial Error	Binomial Error
2	1701621	1690454	1687933	0.66%	0.80%
4	877571	879266	877099	0.19%	0.05%
8	463184	466671	465514	0.75%	0.50%
16	254433	254690	253294	0.10%	0.45%

s15850					
PE	Empirical	Multinomial	Binomial	Multinomial Error	Binomial Error
2	138725	142112	142378	2.44%	2.63%
4	77702	82084	82048	5.64%	5.59%
8	48511	50101	49892	3.28%	2.85%
16	29089	32552	32526	11.90%	11.82%

s9234					
PE	Empirical	Multinomial	Binomial	Multinomial Error	Binomial Error
2	81135	80534	80564	0.74%	0.70%
4	45804	47469	47519	3.64%	3.74%
8	29635	29596	29515	0.13%	0.40%
16	19591	19670	19598	0.40%	0.04%

multi32					
PE	Empirical	Multinomial	Binomial	Multinomial Error	Binomial Error
2	1172691	1178893	1176680	0.53%	0.34%
4	605081	611254	608103	1.02%	0.50%
8	317674	322804	319431	1.61%	0.55%
16	168821	174766	171944	3.52%	1.85%

Table 4. Comparison of theoretical and empirical results

3.3 Speedup and Multiplication factor

From results of the above experiment we obtained the speedup for those four VHDL circuits. Therefore, the speedup was calculated from pure computation with no consideration of communication cost. Figure 2 shows the speedup graph for the VHDL circuits we used above. The multi32 has the best performance while s9234 has the worst. From Table 3 we can see that multi32 has the highest average activity number and s9234 has the lowest average activity number. The speedup is directly related to the average activity number of a circuit.

From the multinomial distribution model, we can see that the speedup is a non-decreasing function of the number of processors. We ignored the communication cost for all previous analysis. If the communication cost is considered, the speedup will not be a non-decreasing function. Indeed if the number of partitions increases over a certain threshold, the performance will be suffered because some processors are just waiting for other processors to finish their work.

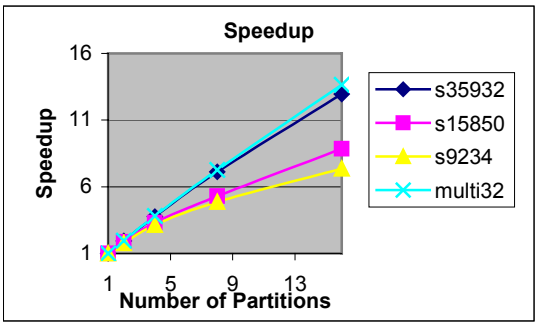


Fig. 2. Speedup graph

4. Communication Model

In this section, we propose a prediction model based on the communication cost and the synchronization cost. The communication overhead of modern message passing parallel computers includes the operating system overhead of dividing the messages into packets, adding header information to the packets and then putting the packets into hardware. For modern parallel computers with advanced routing technique and high bandwidths, communication overhead takes up most of the total communication cost. Because of this we are only interested in the communication overhead and will incorporate it into our prediction model.

There are several parallel models proposed. The LogP model is good when the message is short [6]. The LogGP model is proposed to incorporate long messages [2]. In the LogGP model the message with size k is sent into network at time $O_{null}+m(k-1)$, where O_{null} is overhead and m is time per byte for a message. In this paper, we considered $O_{null}+m(k-1)$ as the communication overhead and, O_{null} as the overhead for sending the null message. We ran the experimental program on SGI Origin 2000. Figure 3 shows the change of communication overhead of sending and receiving as size of message increases.

Figure 4 shows the least square fit of communication overhead to a linear formula. We assumed that the communication overhead O follows following linear formula:

$$O = O_{null} + m*MsgSize$$

Where O_{null} is the overhead of sending a null message and m is the increasing factor as size of the message increases. We used least square method to estimate m and O_{null}. The value of m and O_{null} are $12.9*10^{-3}$ microsecond and 37.3 microsecond separately on SGI origin 2000.

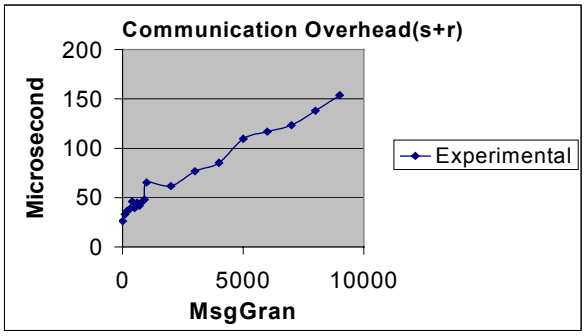


Fig. 3. Message overhead

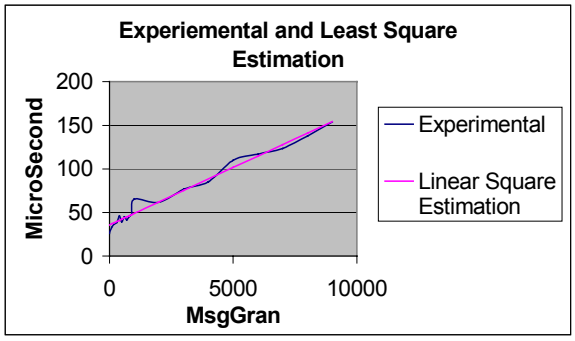


Fig. 4. Least Square fit to a linear formula

5. Performance Model

Total execution time is the sum of computation time and communication time. Our multinomial distribution model can estimate the computation cost. From the activity number and the number of processors, we can get the multiplication factor. The total computation cost is *Total Computation = G*N/(P*M),* where *P* is the number of processors; *M* is multiplication factor; G is the granularity of computation for each execution of an event and *N* is the total number of event executions.

The communication time depends on number of events generated. Suppose that an average of *E* events are generated per cycle and *C* is the total number of cycles. The total number of cycles is total number of event executions divided by the *active number*. When the system is partitioned into P parts, the number of events generated also follows multinomial distribution. The cost of communication per cycle is decided by the maximum order statistics of these multinomial random variables. Therefore, the number of events generated per cycle per processor is *(E/P)*1/M*. These events will be sent to *P* processors. Therefore the event sending and receiving cost of a cycle is $P*((E/P)/(P*M)*m+O_{null})$. O_{null} and *m* are the communication parameters derived from part four of this paper. The total communication cost is:

$$C*P*((E/P)/(P*M)*m+O_{null})=E_{total}/(P*M)*m+C*P*O_{null}$$

Where E_{total} is the total number of events generated. Not all event executions generate new events. Depending on the circuit, we have $E_{total}=N*k$, *where k* is the probability that an event execution generates a new event. The number of cycles *C* can be represented as *N/A*. The refined total communication cost is $N/(P*M)*(m/k)+N/A*P*O_{null}$.

5.1 Predicted Speedup

The speedup is the ratio of sequential simulation time and parallel simulation time. The following gives the predicted speedup of synchronous simulation under our model.

$$Speedup=\frac{T_{seq}}{T_{comp}+T_{comm}}=\frac{N*G}{N*G/(P*M)+N/(P*M)*(m/k)+N/A*P*O_{null}}$$

The above formula can be simplified to:

$$\frac{1}{\frac{1+(m/k)/G}{P*M}+\frac{O_{null}}{G}*\frac{P}{A}}$$

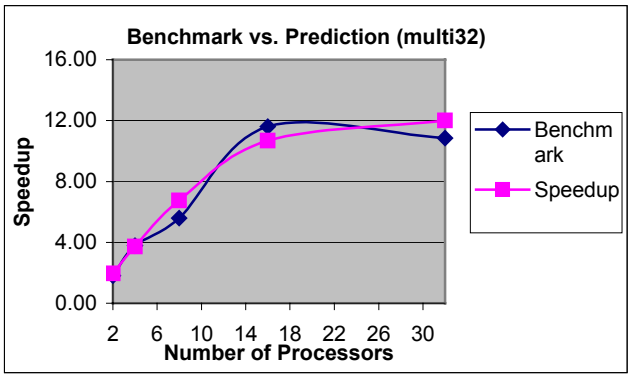
G includes the event queue handling and event execution that can be considered to be much larger than *m* and *k* is within a practical range. The value of *m*, the unit cost of message per byte, is around $12.9*10^{-3}$ microsecond for SGI Origin 2000 and value of *k*, the event generation rate, is dependent on the circuit and the values are around 0.1 for the circuit we have tested. The value of *G* is dependent on the circuit too and it is in the order of microseconds. Therefore, *(m/k)/G* is close to 0. This can be explained intuitively. The queue overhead plus the object execution cost for generating one event is much larger than the cost of sending a longer message with additional size of an event. We removed *(m/k)/G* from the formula to get the simpler formula:

$$\frac{1}{\frac{1}{P*M}+\frac{O_{null}}{G}*\frac{P}{A}}$$

From this formula we can see that if A is fixed, the upper limit of speedup is *P*M* when $G >> O_{null}$. When *G* is fixed the upper limit of speedup is also *P*M* when A >> P. The large *active number* A has the effect of reducing both computation and communication cost. According to the formula, synchronous simulation performs best when A is large. Although granularity G helps to overshadow the communication cost, it cannot overcome the upper bound speedup barrier of *M*P*.

5.2 Benchmark Results

Figure 5 shows the comparison of empirical speedup and theoretical speedup for s5932 and multi32. The prediction formula predicts the benchmark reasonably well. Figure 6 shows the effect of granularity. We put some loops to the object execution to study the effect of granularity over speedup on s35932 and multi32. Each unit of extra granularity is around 20us. There is improvement of speedup when extra G goes up from 0 to 1. But when extra G is over 1, there is no significant improvement on speedup. This can be explained by our prediction formula: when $G*A >> O_{null}*P$, $\frac{1}{P*M}$ dominates $\frac{O_{null}}{G}*\frac{P}{A}$. For s15850, it has much smaller A than s35932, although *G* is large $\frac{1}{P*M}$ cannot overwhelm $\frac{O_{null}}{G}*\frac{P}{A}$. This caused the better speedup for larger *G*.



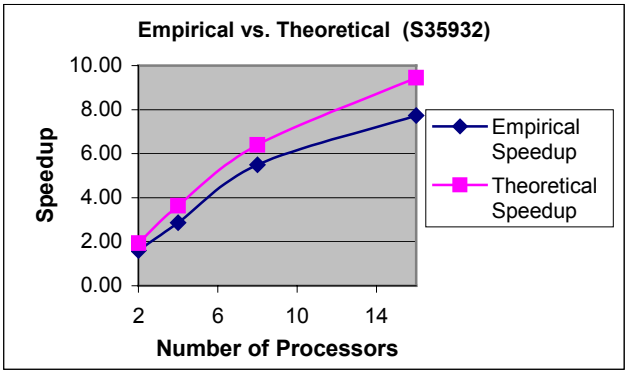
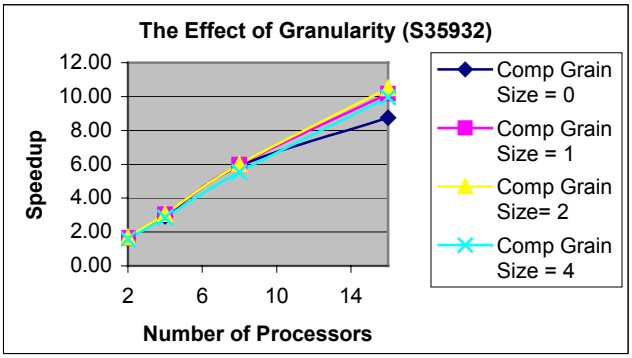


Fig. 5. The empirical speedup vs. Theoretical speedup.



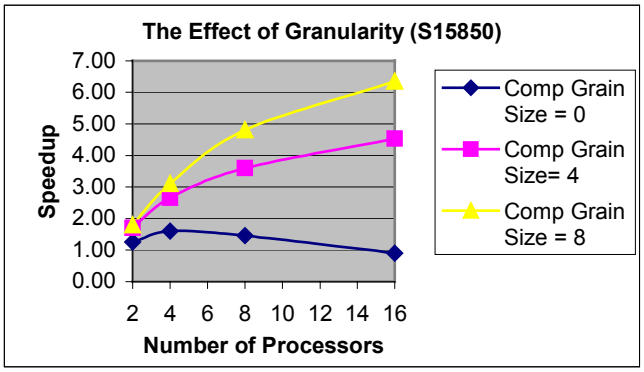


Fig. 6. The effect of granularity.

5.3 Observation

The effect of granularity
Higher granularity can improve the performance because it can hide the increasing communication cost as the number of processors increases. This can be seen best in the speedup graph for s15850 with varying granularity.

When the granularity is large enough that communication takes a very small part of the total execution time, increasing the granularity will not improve the performance. Inherent load unbalancing as demonstrated by our multinomial distribution model restricts speedup. The speedup of s35932 did not increase with the granularity starting from two. The predicted communication cost of s35932 at granularity two takes less than 12% of the total cost.

The cost computation vs. communication
The cost of computation decreases as number of processors increases, while the cost of communication increases as the number of processors increases. If the activity number is small or granularity is small, then the computation takes a small portion of the total cost and performance will suffer.

6. Conclusion

In this paper we proposed a performance model for synchronous parallel discrete event simulation. From the speed-up formula derived from our performance model, we can predict that synchronous simulation performs better with a higher *average active number* of objects per cycle. It can both reduce the load unbalancing factor and communication cost. The large computation granularity can hide communication cost but it cannot hide the effect of load unbalancing caused the random distribution of active objects through a number of processors. P*M determines the upper limit of speedup of synchronous simulation, which is a function of the number of processors and the average number of active objects per cycle.

Increasing the average active number of object per cycle is the key for improving the performance of synchronous simulation. In the forthcoming paper, we are planning to propose a relaxed synchronous algorithm that can increase this number and improve the performance. The error of our prediction result compared with the benchmark result is less than twenty percent. This result can give some hint on whether it is worthwhile to use synchronous simulation before actually implementing it.

7. References

[1] Vusgwani D. Agrawai and Srimat Chakradhar, 1992. Performance Analysis of Synchronized Iterative Algorithms on Multiprocessors Systems. In IEEE Transactions on Parallel and Distributed Systems Vol. 3 No. 6, pp. 739-749.

[2] A. Alexandrov, M. Ionescu, K. E. Schauser, C. Scheiman, 1995. "LogGP: Incorporating Long Messages into the LogP model - One step closer towards a realistic model for parallel computation", 7th Annual Symposium on Parallel Algorithms and Architecture (SPAA'95)

[3] K. M. Chandy and J. Misra, 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Transaction on Software Engineering.

[4] Eunmi Choi and Moon Jung Chung, 1995. An Important Factor for Optimistic Protocol on Distributed Systems: Granularity. In Proceeding of Winter Simulation Conference.

[5] Moon Jung Chung, Jinsheng Xu and Hee Chul Kim, 1998. Parallel VHDL Simulation Engine. Department of Defense High Performance Computing Modernization Program, UGC98

[6] David Culler, et. al. 1993. LogP: Towards a Realistic Model of Parallel Computation. ICSA 93.

[7] P. M. Dickens, D. M. Nicol, P. F. Reynolds, JR. and J. M. Duva. 1997. Analysis of Bounded Time Warp and Comparison with YAWNS. ACM Transaction on Modeling and Computer Simulations, Vol. 6, No. 4.

[8] R. E. Felderman and L. Kleinrock, 1990. An Upper Bound on the Improvement of Asynchronous versus Synchronous Distributed Processing. Proc. of the SCS Multiconf. on Distributed Simulation, vol. 22.

[9] Alois Ferscha, 1995. Parallel and Distributed Simulation of Discrete Event Systems. Handbook of Parallel and Distributed Computing.

[10] D. R. Jefferson, 1985. Virtual Time. ACM Transaction of Programming Languages and Systems, 7(3).

[11] Chu-Cheow Lim et. al. Performance Prediction Tools for Parallel Discrete-Event SimulationGintic Institues of Manufacturing Technology.

[12] Yi-Bing Lin, 1992. Parallelism Analyzers for Parallel Discrete Event Simulation. ACM Transaction of Modeling and Computer Simulation.

[13] B.D. Lubachevsky, A. Weiss and A. Shwartz, 1991. An Analysis of Rollback-Based Simulation. ACM Transaction of Modeling and Computer Simulation.

[14] D. M. Nicol, 1991. Performance Bounds on Parallel Self-Initiating Discrete Event Simulations. ACM Transaction of Modeling and Computer Simulation.

[15] J. K. Peacock, J. W. Wong and E. C. Manning. 1979. Distributed Simulation using a Network of Processors. Computer Networks: 3(1): 44-56

[16] Christian Schaubschlaeger, 1999. Measurements of SKaMPI, Version 2.2 of SGI Origin 2000 at GUP/ZID, University of Linz, Austria.

[17] L. Soule and A. Gupta, 1991. An Evaluation of Chandy-Misra-Bryant Algorithm for Digital Logic Simulation. ACM Transaction of Modeling and Computer Simulation.

[18] L.G. Valiant, A Bridging Model For Parallel Computation. 1990. Communications of ACM.