Performance Evaluation of a Parallel Simulation Environment

Yong-Meng Teo and Seng-Chuan Tay
School of Computing
National University of Singapore
Lower Kent Ridge Road
Singapore 119260
email: teoym@comp.nus.edu.sg

Abstract

Developing a parallel discrete-event simulation from scratch requires an indepth knowledge of the mapping process from the physical model to the simulation model, and a substantial effort in coping with numerous parallelism issues in the underlying synchronization protocols adopted. The lack of software tools and environments to reduce the development effort significantly is a major hindrance in adopting parallel simulation technology. This paper presents an overview of the SPaDES (Structured Parallel Discrete-Event Simulation) scalable parallel simulation framework. We focus on the performance analysis of SPaDES/C++, an implementation of SPaDES on a distributed-memory Fujitsu AP3000 parallel computer. SPaDES/C++ hides the underlying complex parallel simulation synchronization and parallel programming details from the simulationist. We study various ways of improving SPaDES execution performance including periodic checkpointing of simulation states, aggregation of messages for logical processes that reside on the same physical processors, and increasing the computational granularity of run-time processes to reduce the costs of synchronization and communication. Our empirical results show that the SPaDES framework can deliver good speedup for applications with large problem size and is scalable.

1 Introduction

There is growing interest in parallel discrete-event simulation (PDES) especially in areas such as computer science and engineering where sequential simulation can consume enormous amount of computation time [2, 3]. With the increasing availability of low-cost parallel machines, PDES offers an alternative to help speed up the execution of these simulations. However, PDES has not achieved widespread use and remains an active area of research. A major reason for this is that efficient parallel simulation is difficult to implement due to its complexity, and the lack of appropriate tools to support the use of PDES technology. SPaDES [14] is such a tool that we have developed to provide a high-level, portable, and scalable parallel simulation environment that facilitates simulation modeling and programming.

The rest of the paper is divided into four sections. Section 2 presents an overview on the design and specification of SPaDES. Section 3 discusses the performance analysis for an implementation of SPaDES called SPaDES/C++ using three benchmark programs. Performance optimization and scalability analysis are discussed to ascertain the usability of the simulation environment for large applications. Section 4 contains our concluding remarks.

2 Overview of SPaDES

SPaDES adopts a modular architecture as shown in figure 1 [14, 15]. The process-oriented modeling methodology adopted allows the abstraction and mapping of *entities*

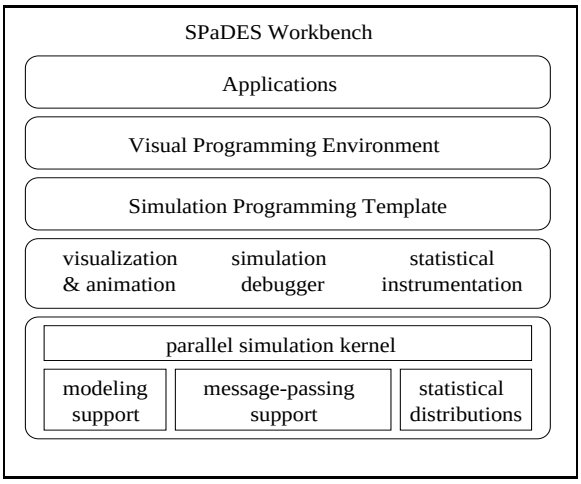

Figure 1. Architecture of SPaDES

and *servers* in a real-world problem as *processes* and *resources* in the conceptual model. Processes and resources are mapped onto *messages* and *logical processes* in the operational model (parallel simulator) respectively. The conceptual model can be implemented using a visual programming environment that allows the user to build the model interactively using an iconic representation. Alternatively, the operational model can be programmed using the parallel simulation programming template provided. The parallel simulation kernel implements the parallel simulation protocol.

An operational model of a physical system consists of a set of logical processes (LPs) each corresponding to a physical process. Each LP is responsible for simulating local events. All interactions between physical processes are modeled by timestamped event messages sent between the LPs. The current implementation adopts the optimistic protocol [6], and an event parallelism throttle [13] to better match simulation parallelism to available machine parallelism. The mapping of real-word entities onto LPs is as follows:

- permanent entities such as resources are mapped to LPs;
- temporary entities such as processes are represented as time-stamped messages.

For example, consider a queuing system shown in figure 2. The servers S1, S2 and S3 are modeled as LPs in the operational model and the jobs are modeled as messages.

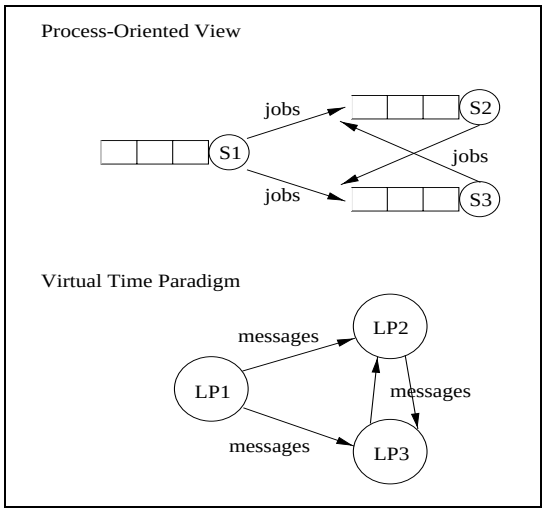

Figure 2. Process-Oriented Modeling and Virtual Time Paradigm

Table 1 provides a summary of the simulation primitives that support simulation modeling. SPaDES also permits users to control the mapping of a parallel simulator onto physical processors (PP) in order to vary the process granularity for better execution performance. The primitives to support process-to-processor mappings are listed in table 2.

3 Performance Evaluation

SPaDES/C++ is an implementation of the SPaDES specification to support parallel simulation modeling through library calls. The simulation and parallelism capabilities of SPaDES/C++ are provided in the form of primitives encapsulated in a library of classes. Both PVM [5] and MPI [12] are supported to handle message-passing in parallel event execution.

The current implementation of SPaDES/C++ runs on a Fujitsu AP3000 parallel computer with 32 143MHz Sun UltraSparc processors. Each processor has a 256MB of memory and is connected by a high-speed 2-D torus communication network. Three simulation benchmarks are used in this study, namely (i) a super-ping model that consists of a number of objects (n) connected in a ring with a fixed number of messages circulating the objects [1], (ii) a $n \times n$ torus network where messages in the model are routed based on an uniform probability, and (iii) an open model simulating a $n \times n$ multistage omega interconnection network (MIN).

3.1 Elapsed Time

For purpose of comparison, we wrote the simulation benchmarks using the Simscript [11] simulation language and in the C-based simulation library by Watkins [17]. Table 3 compares these implementations - C library (denoted by CSim), Simscript[1] and SPaDES/C++. The program code and executable file size for SPaDES/C++ is the smallest among all the three implementations.

Table 4 depicts the simulation run-time on the Fujitsu AP3000. The sequential simulators are executed on one processor while the parallel runs are based on 1, 2, 4, 8 and 16 processors. The LP to PP mappings are such that the resulting number of PPs is always 16, i.e. a mapping of 16-1 for both the super-ping and torus models (with every 16 LPs mapped to one PP), and a 36-1 for MIN (horizontal partitioning with every eight rows of switches mapped onto one PP). The poor performance is due mainly to the high communication overhead (for $p > 1$) and the fine granularity of LPs.

3.2 Optimization

We improve the performance of SPaDES/C++ in two main areas: time-warp simulation protocol and computational granularity.

[1]The Simscript implementation of the MIN model was not used in the comparison because Simscript does not support bitwise operations.

simulation primitive	argument	type	meaning	description
activate	process time	(handle) (float)	process to be activated activation time of process	creates a new process and schedules it immediately for execution
reactivate	process time	(handle) (float)	process to be reactivated the time at which the process is reactivated	reactivates a process that has been suspended by a previous call to suspend
work	resource time units	(handle) (float) (integer)	the resource requested for time required to service request number of service units requested	a process request for service at a resource
wait	time	(float)	amount of time to delay	a process delays itself for a known period of time
suspend	None			a process suspends itself during execution
terminate	None			a process terminates itself
startSimulation	duration	(float)	duration of simulation	starts a simulation
resetSimulation	None			resets a simulation

Table 1. Simulation Primitives

mapping primitive	argument	type	meaning	description
mapProcess	event LP	(handle) (handle)	event to be scheduled LP whose event list will be initialized	schedules an event in the event list of a specific LP
mapNode	numLPs LParray PP	(integer) (array) (integer)	number of LPs LPs to be clustered PP identifier	clusters a number of LPs on a physical process
mapNode	numLPs LP_0 LP_1 ... PP	(integer) (handle) (integer)	number of LPs individual LPs PP identifier	clusters a number of LPs on a physical process
mapHost	numPPs PP_1 PP_2 ... hostname	(integer) (integer) (string)	number of PPs PP identifiers name of processor	assigns several PPs to a specific processor

Table 2. Mapping Primitives

3.2.1 Time-warp Simulation Protocol

We study the performance effect of varying the following simulation protocol parameters provided by the SPaDES implementation:

(i) *the number of simulation cycles before messages are sent (m)*
A typical simulation cycle consists of the following activities:

- receive all external messages
- process a message, and
- send message(s) to another LP.

Output messages can be aggregated to increase the message size and to reduce the message-passing overhead. The default value for m is 100 cycles.

(ii) *the frequency of state-saving (s)*
To support rollback and recovery in optimistic simulation, the simulator states are saved every cycle by default (s=1). Adaptive checkpointing is discussed in [10]. We study the effect of periodic state-saving to reduce the overhead of state-saving.

Table 5 shows the performance of the three benchmarks *before* optimization using the following performance metrics:

1. *elapsed time in seconds* that is defined as the total wall clock time taken to execute the simulation
2. *efficiency* which is defined as:

$$eff. = \frac{events\ committed}{events\ executed}$$

This measures the efficiency of the time-warp protocol where a high efficiency denotes less rollbacks.

program	implementation	measure		
		line of code	exec. size	comp/link time
super-ping (256)	CSim	166	48,396	1.8
	Simscript	93	106,568	5.3
	SPaDES/C++	73	11,876	3.8
torus (16×16)	CSim	197	49,248	1.9
	Simscript	137	108,164	5.1
	SPaDES/C++	109	13,456	3.7
MIN (128×128)	CSim	222	49,372	2.0
	SPaDES/C++	138	12,476	3.5

Table 3. Comparison of Implementations

implementation	program		
	super-ping (256)	torus (16×16)	MIN (128×128)
CSim			
exec. time in seconds	70.40	16.88	110.06
Simscript			
exec. time in seconds	183.80	49.73	–
SPaDES/C++			
number of LPs used	256	256	576
exec. time in sec. (sequential)	107.68	31.15	207.43
exec. time in sec. (parallel)			
p=1	3680.43	1850.57	3783.68
p=2	1990.74	936.40	2555.73
p=4	837.64	441.32	1152.80
p=8	485.03	261.86	513.77
p=16	137.75	62.55	93.16

Table 4. Comparison of Simulation Elapsed Time

3. *event execution rate* in terms of the number of events executed per second (events per sec.)
   A high event rate implies better processor utilization. However, it could also mean that the processor may be busy executing overhead events that include: correct events that were not saved due to periodic state-saving (forward computation) and premature events which are generated based on an erroneous state.
4. *number of rollbacks per event* which is defined as:

$$\frac{events\ executed - events\ committed}{events\ committed}$$

   This metric computes the ratio of the number of rollbacked events to the number of committed events.
5. *number of sends*
   The number of sends refers to the number `pvm_send()` executed to send messages. This decreases as more messages are aggregated.
6. *message size*
   Message size refers to the average size of a aggregated message. Thus, an aggregated message may consist of several event messages to be sent to different LPs residing in the same physical process.

We observe in table 5 that for a given problem size and as the simulation duration is increased, the event execution rate, efficiency, and the rollbacks per event are fairly constant. This shows that the simulations are executing at the steady state.

As shown in figure 3, the elapsed time reduces as the parameter m varies from 1 to 400. The initial drop in elapsed time is due to the aggregation of messages which reduces the communication overhead. For larger values of m, the delay in sending out messages result in event starvation and thus increases the elapsed time.

Selecting the smallest values of m for each benchmark, we repeat the experiments by varying s, the period of checkpointing from 1 to 50 simulation cycles. Figure 4 shows that the elapsed time generally increases as s is increased. In both the super-ping and MIN benchmarks, the smallest elapsed times are recorded at s=5 and 3 respectively. This

program	duration	elapsed time (sec)	eff.	events per sec.	rollback per event	no. of sends	message size (bytes)
Super-ping	10000	137.75	0.46	23455.31	1.18	197047	236
(256)	20000	271.63	0.46	23776.40	1.16	389571	235
	30000	403.35	0.46	24011.69	1.15	578044	236
Torus	100000	62.62	0.29	10613.87	2.47	99679	708
(16×16)	200000	130.05	0.28	10217.24	2.62	206941	713
	300000	191.01	0.28	10448.85	2.55	304583	710
MIN	100000	93.16	0.54	15789.29	0.84	137745	1121
(128×128)	200000	190.27	0.54	15708.06	0.86	280499	1123
	300000	287.81	0.54	15689.20	0.86	423214	1123

Table 5. Initial Performance of the Three Benchmarks

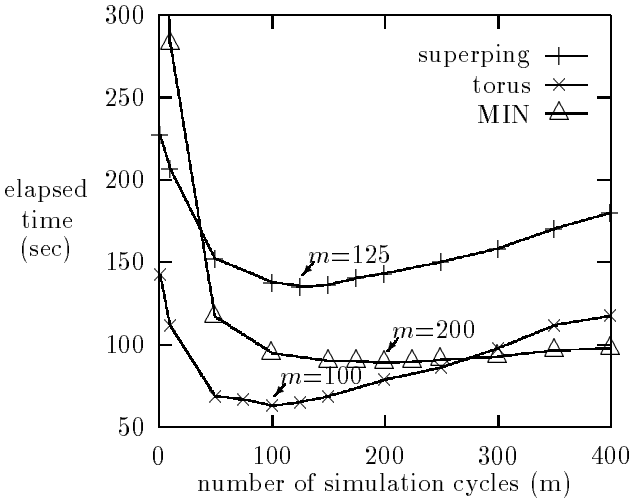

Figure 3. Effect of Delayed Message Transmission

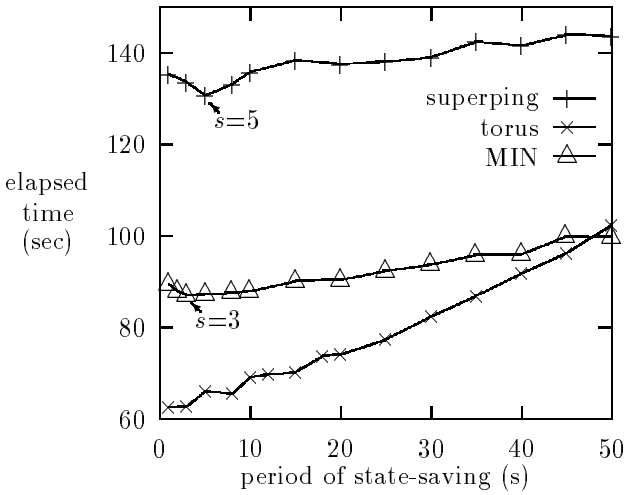

Figure 4. Effect of Periodic Checkpointing

shows that the reduction in total state-saving time results in savings in the overall simulation elapsed time at the specified points. However, as s is increased further so is the computation time required to recompute the correct events. For the torus model, the elapsed time of the simulation does not show any distinct savings as s is varied. A plausible reason for this is that unlike the other two models, messages within the torus model are routed based on a uniform probability. As a result, fewer messages can be aggregated and at the same time this also increases the number of rollbacks in the system. Hence, it is difficult to obtain any savings with such a model using periodic state-saving since the rollback costs are now much higher. This is further exemplified by the fact that the graph of the elapsed time of torus model rises at a faster rate than the other two models.

Using the values for m and s and the optimization discussed, we re-run the simulation benchmarks. Table 6 shows an improvement (for super-ping and MIN) in the simulation run-time as well as the processor utilization as reflected by the higher event execution rate. The drop in the efficiency figures (or the increase in the number of rollbacks) can be attributed to the delayed sending of the event messages. The number of `pvm_send()` required in the simulation decreases as a result of message aggregation while the average message size increases. Thus, the time required to execute overhead events is much smaller than that caused by the communication overhead.

3.2.2 Varying Computation Granularity

While mapping many LPs to a single PP increases the computation granularity, it decreases the parallelism in the simulation. On the other hand, having fewer LPs on a PP increases parallelism but introduces communication overhead. Therefore, the LP-to-PP mapping is a tradeoff between simulation parallelism and the message communication overhead.

To study the mapping effect on simulation performance, we re-run the simulation benchmarks on 1, 2, 4, 8 and 16 processors using different mappings. Table 7 shows the elapsed times with the best elapsed times for a fixed number of processors highlighted in bold. We observe that for a a fixed number of processors, there exists a LP-to-PP mapping that gives the best run-time, i.e. 64 for super-ping

program	duration	elapsed time (sec)	eff.	events per sec.	rollback per event	no. of sends	message size (bytes)
Super-ping	10000	130.32	0.38	24792.41	1.64	184294	266
(256)	20000	267.48	0.37	24145.02	1.70	376064	268
	30000	396.52	0.37	24425.67	1.67	559373	266
Torus	100000	62.64	0.29	10611.08	2.48	100267	705
(16×16)	200000	130.15	0.28	10215.14	2.62	206939	713
	300000	191.95	0.28	10397.89	2.53	303452	709
MIN	100000	86.45	0.40	17015.84	1.52	85246	2040
(128×128)	200000	174.54	0.40	17124.01	1.52	172480	2030
	300000	264.07	0.40	17099.79	1.51	261116	2019

Table 6. Performance After Optimization

program	mappings LP to PP	No. of PPs	Number of Processors				
			1	2	4	8	16
Super-ping	16-1	16	3473.90	1935.33	854.95	474.23	**131.57**
(256)	32-1	8	1412.59	791.94	454.93	**151.18**	
	64-1	4	**1214.40**	**614.60**	**284.65**		
	128-1	2	1636.73	771.46			
	256-1	1	2631.09				
Torus	16-1	16	1850.57	936.40	441.32	261.86	**62.55**
(16×16)	32-1	8	549.71	317.74	185.78	**51.19**	
	64-1	4	**418.74**	218.14	**84.26**		
	128-1	2	464.06	**198.71**			
	256-1	1	611.05				
MIN	36-1	16	1524.29	1040.47	452.15	215.12	**87.29**
(128×128)	72-1	8	**1325.81**	**748.26**	390.78	**155.78**	
	144-1	4	1582.02	803.16	**360.31**		
	288-1	2	2266.25	1064.25			
	576-1	1	3851.52				

Table 7. Elapsed Time for Different Mapping Schemes

and torus, and 72 for MIN. Considering only the elapsed times of the mappings that yield the optimal performance, we generally observe a decrease in the elapsed time as more processors are used. However, in the case of the torus, the run-time increases slightly with 16 processors because the torus does not have sufficient computation to amortize the communication overhead, i.e. when the mapping of LPs to PPs is reduced to 16-1.

3.3 Scalability

To evaluate the scalability of the SPaDES/C++ environment, we run the simulation benchmarks by varying the number of LPs present in the models. Figure 5 shows the elapsed times for the three simulation benchmarks running on 16 processors using 16 PPs. Both the super-ping and torus benchmarks show an initial drop in elapsed time due to its smaller problem sizes. However, the elapsed times improve significantly as the problem size increases. Generally, for these benchmarks as the problem size is increased, the elapsed times grow almost linearly. This indicates the scalability of SPaDES/C++.

3.4 Speedup

Figure 6 shows the speedup against the number of processors for the three benchmarks. We observe that SPaDES/C++ is able to obtain better speedups for simulation with larger problem sizes. In fact, it achieves a speedup of around 4 and 9 (with 2048 LPs) running on 8 and 16 processors respectively. The best speedup is obtained with the MIN simulation which produces a speedup of around 12. Further investigation is needed to improve the speedup. In summary, the experiments show that to obtain good speedup, the benchmark must

- contain several hundreds of simulation processes or LPs (over 500)

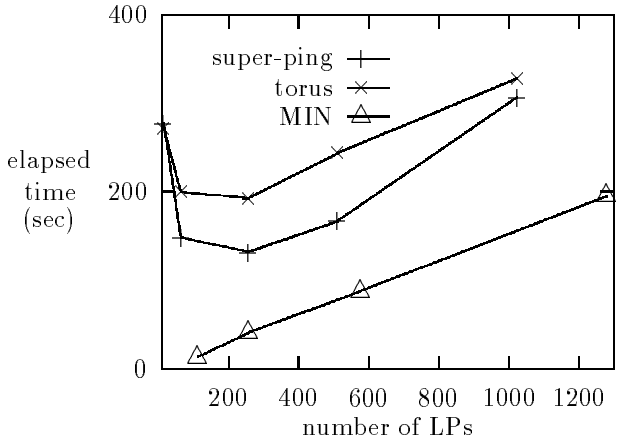

Figure 5. Scalability of SPaDES/C++

- contain sufficient computation (parallelism) to amortize the communication overhead by mapping more LPs to a PP (around 64-72)

4 Conclusions

The main design objective of the SPaDES workbench is to permit a simulationist to develop a parallel simulator without being overly concerned with the execution environment. It aims to provide modeling and simulation support that is comparable with sequential simulation languages, but exploit parallelism using PDES techniques. As SPaDES does not provide any explicit message-passing primitives nor it requires users to specify the connectivity information between the parallel processes, a higher degree of transparency and portability can be achieved. We observe that the performance of SPaDES/C++ is highly influenced by the underlying *parallel simulation synchronization implementation*, the granularity of LPs and its mapping onto physical processors, and the cost of *communication* in the execution platform. However, the effects of these overheads can be reduced through periodic checkpointing of simulation states, aggregating messages sent to LPs, and increasing the computation granularity of processes. Empirical results show that the SPaDES is scalable and can deliver good speedup for large PDES applications.

Understanding the performance of parallel simulation is a complex issue and is essential for this technology to be widely accepted in industry [4]. A framework to analyze parallel simulation performance is discussed in [16].

Acknowledgements
This research has been supported by a grant from the Ministry of Education and the Port of Singapore Authority under grant RP960715. The authors like to thank Siew Theng Kong for implementing the SPaDES/C++ library.

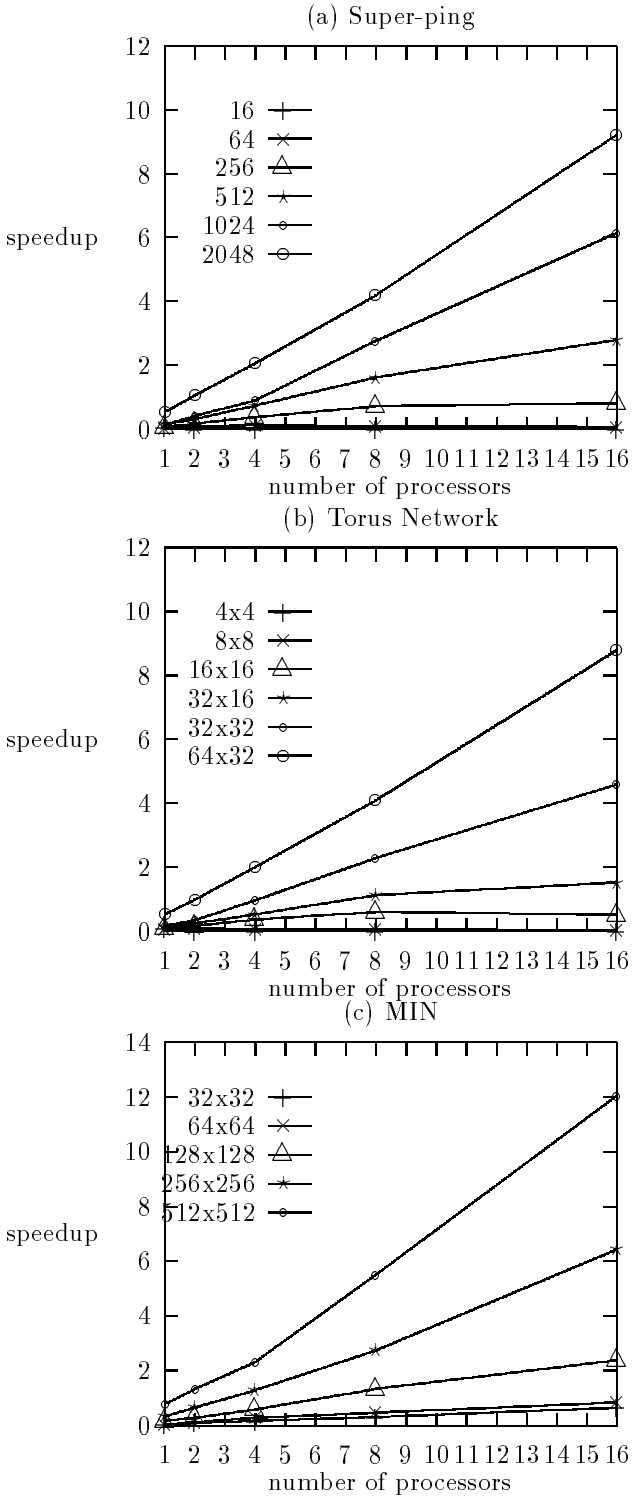

Figure 6. Speedup Analysis

References

[1] L. Barriga and R. Ronngren and R. Ayani, *"Benchmarking Parallel Simulation Algorithms,"* in Proceedings of the IEEE International Conference on Algorithms and Architectures for Parallel Processing, pp. 611-620, April, 1995.

[2] A. Ferscha, *"Parallel and Distributed Simulation of Discrete Event Systems,"* in Handbook of Parallel and Distributed Computing, McGraw-Hill, 1995.

[3] R. M. Fujimoto, *"Parallel Discrete Event Simulation, Communications of the ACM,"* Vol 23, No. 10, pp. 31-53, October 1990.

[4] R. M. Fujimoto, *"Parallel Discrete Event Simulation: Will the Field Survive?,"* ORSA Journal of Computing, Vol. 5, No. 3, pp. 213-230, Summer 1993.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, *"PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing,"* The MIT Press, 1994.

[6] D. R. Jefferson, *"Virtual Time,"* ACM Transactions on Programming Languages and Systems, Vol. 7, No. 3, pp. 404-425, July 1985.

[7] R. E. Nance, *"A History of Discrete Event Simulation Programming Languages,"* Proceedings of the Second ACM SIGPLAN History of Programming Languages Conference, ACM SIGPLAN Notices, Vol. 28(3), pp. 149-175, March 1993.

[8] D. Nicol and R. M. Fujimoto, *"Parallel Simulation Today,"* Annals of Operations Research, Vol. 53, pp. 249-286, December 1994.

[9] H. Rajaei and R. Ayani, *"Design Issues in Parallel Simulation Languages,"* IEEE Design & Test of Computers, pp. 52-63, December 1993.

[10] R. Rönngren and R. Ayani, *"Adaptive Checkpointing in Time Warp,"* Proc. 8th Workshop on Parallel and Distributed Simulations, pp. 110-117, IEEE Computer Society Press, 1994.

[11] E.C. Russell, *"SIMSCRIPT II.5 and SIMGRAPHICS Tutorial,"* Proc. of Winter Simulation Conference, pp. 223-227, 1993.

[12] M. Snir et al. *"MPI: The Complete Reference"*, MIT Press, 1996.

[13] S.C. Tay, Y.M. Teo and S.T. Kong *"Speculative Parallel Simulation with an Adaptive Throttle Scheme,"* Proceedings of the 11th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation, pp. 116-123, Austria, June 1997.

[14] Y.M. Teo, S.C. Tay and K.T. Kong, *"Structured Parallel Simulation Modeling and Programming"*, Proceedings of the 31st Annual Simulation Symposium, Boston, Massachusetts, USA, IEEE Computer Society Press, pp. 135-142, April 5-9, 1998.

[15] Y.M. Teo, S.C. Tay, *"Parallel Simulation: Programmability, Performance and Scalability"*, Proceedings of the 5th Australasian Conference on Parallel and Real-time Systems, Adelaide, Australia, Springer-Verlag, pp. 273-284, September 1998.

[16] Y.M. Teo, H. Wang and S.C. Tay, *"A Framework for Analyzing Parallel Simulation Performance"*, Proceedings of the 32st Annual Simulation Symposium, San Diego, USA, IEEE Computer Society Press, April 1999.

[17] K. Watkins, *"Discrete Event Simulation in C,"* McGraw-Hill Book Company, 1993.