Performance and Granularity Control in the SPaDES Parallel Simulation System

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

Parallel simulation has the potential to accelerate the execution of simulation applications. However, developing a parallel discrete-event simulation from scratch requires an in-depth knowledge of the mapping process from the physical model to the simulation model, and a substantial effort in optimising performance. This paper presents an overview of the SPaDES (Structured Parallel Discrete-Event Simulation) 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. Our empirical results show that the SPaDES framework can deliver good speedup if the process granularity is properly optimised.

1 Introduction

Parallel discrete-event simulation (PDES), also called distributed simulation, refers to the execution of a *single* discrete-event simulation program on a parallel computer [3, 7]. In recent years, there has been a growing interest in PDES due to the bottleneck in the conventional approach where large sequential simulation in various fields of computer science and engineering consume enormous amount of time. Furthermore, with the increasing availability of low-cost parallel machines, parallel simulation has become a viable 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 [10] 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 of SPaDES. Section 3 discusses the performance analysis for an implementation of SPaDES called SPaDES/C++ using three benchmark programs. Performance optimization and speedup analysis are discussed to ascertain the usability of the simulation environment. Section 4 contains our concluding remarks.

2 SPaDES Framework

SPaDES places great emphasis on structured modeling and programming in the development of parallel simulators by providing the users with a set of well-defined modeling primitives, object classes and a modeling template. Figure 1 shows the modular architecture of SPaDES framework. The process-oriented modeling methodology adopted allows the

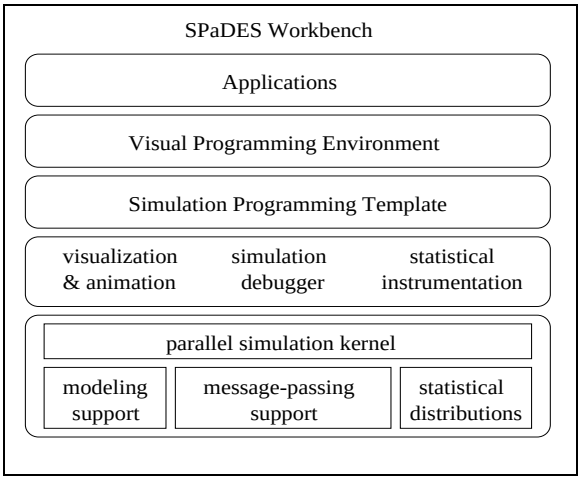

Figure 1. Architecture of SPaDES

abstraction and mapping of *entities* and *servers* in a real-world problem as *processes* and *resources* in the concep-

tual 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.

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 [5], and an event parallelism throttle [9] 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.

A set of simulation primitives to support simulation modeling can be found in [10].

Frequently, the granularity of an LP is small and thus incurred a high synchronization and communication overheads if the LP is mapped onto one processor. To increase computational granularity, several LPs can be clustered to form a larger run-time process *(called physical process (PP))* such as threads or PVM process. PP(s) are then mapped onto physical processors and many PPs can be mapped onto one processor. LPs residing on a PP shares the same physical memory. Message-passing is necessary for PPs that reside on different processors. The clustering of LPs that communicate frequently in the same PP permits one to control the computation granularity of parallel simulations for better execution performance. Similarly, the ability to spawn PPs on different processor permits one to customize the locations of the PPs so as to take advantage of the communication topology of the execution platform. For instance, PPs that interact frequently can be assigned to the same processor or to adjacent processors and PPs can be mapped to the processors to achieve an even distribution of workload. As shown in table 1, SPaDES provides users the primitives to control the computational granularity of process for better execution performance.

3 Performance Evaluation

An implementation of SPaDES called SPaDES/C++ supports parallel event execution. Simulation and parallelism capabilities are provided through a set of primitives encapsulated in a library of classes. SPaDES/C++ adopts the PVM message-passing library to support message passing. An implement of SPaDES/C++ using MPI for message passing is also available.

The current implementation of SPaDES/C++ runs on a number of platforms including a network of PCs running Window NT. The experiments reported in this paper were carried out on a Fujitsu AP3000 parallel computer with 32 143MHz Sun UltraSparc processors. Each processor has 256MB of memory and is connected by a high-speed 2-D torus communication network. The three simulation benchmarks are:

1. A super-ping model, super-ping(n), consists of *n* objects connected in a ring with a fixed number of messages circulating the model [1]. As shown in figure 2, each object is mapped onto one LP.

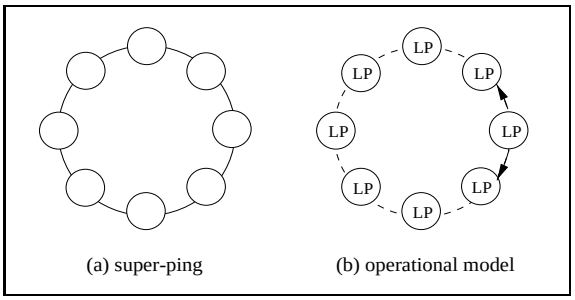

Figure 2. A Super-Ping Simulation Model

2. A $n \times n$ torus network as shown in figure 3 where messages in the switch node are routed based on a uniform probability. Each network is model using n^2 LPs.

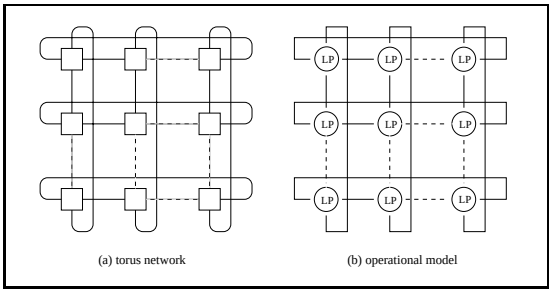

Figure 3. A $n \times n$ Torus Network

3. An open model simulating a $n \times n$ multistage omega interconnection network (MIN). As shown in figure 4, each 2×2 switch is modeled using one LP. Thus, a $n \times n$ MIN is mapped onto $\frac{n}{2} log_2 n + n$ LPs.

3.1 Elapsed Time

Table 2 depicts the simulation run-time on the Fujitsu AP3000. The sequential simulators are executed on one

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

Table 1. Mapping Primitives

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 2. Comparison of Simulation Elapsed Time

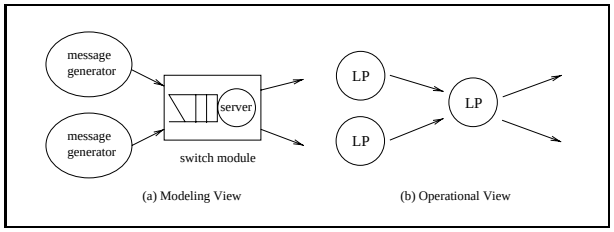

Figure 4. A Model for a 2×2 Switch

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).

3.2 Optimization

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

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)*
Output messages are 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)*
The default value is s=1.

Table 3 shows the performance of the three benchmarks 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:

$$efficiency = \frac{events\ committed}{events\ executed}$$

3. *event execution rate* in terms of the number of events executed per second (events per sec.)
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.

We observe in table 3 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 5, 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 6 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 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.

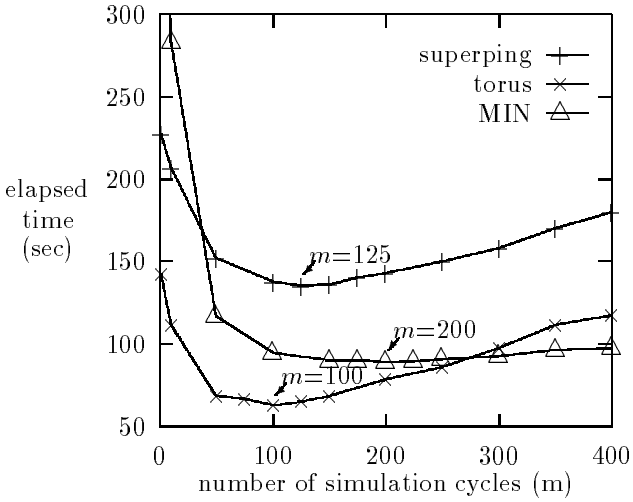

Figure 5. Effect of Delayed Message Transmission

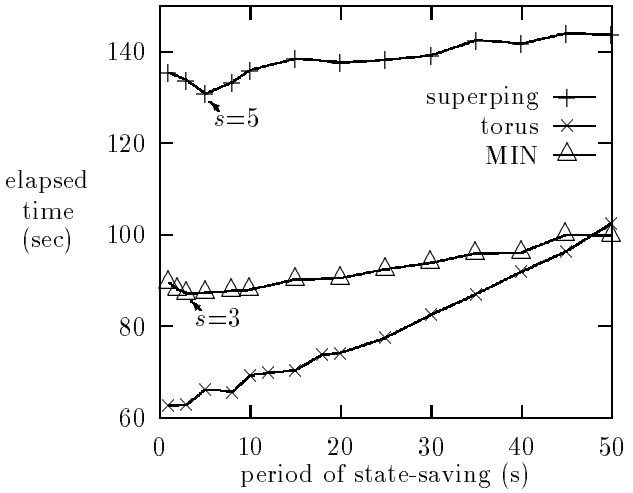

Figure 6. Effect of Periodic Checkpointing

Using the values for m and s and the optimization discussed, we re-run the simulation benchmarks. The bracketed values in table 3 shows an improvement (for superping 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.

program	duration	elapsed time (sec)	efficiency	events per sec. ($\times 10^3$)	rollback per event	no. of sends ($\times 10^4$)	message size (bytes)
Super-ping (256)	10000	138 (130)	0.46 (0.38)	23.5 (27.8)	1.18 (1.64)	19.7 (18.4)	236 (266)
	20000	272 (267)	0.46 (0.37)	23.8 (24.1)	1.16 (1.70)	34.0 (37.6)	235 (268)
	30000	403 (397)	0.46 (0.37)	24.0 (24.4)	1.15 (1.67)	57.8 (60.0)	236 (266)
Torus (16×16)	100000	63 (62)	0.29 (0.29)	10.6 (10.6)	2.47 (2.48)	10.0 (10.0)	708 (705)
	200000	130 (130)	0.28 (0.28)	10.2 (10.2)	2.62 (2.62)	20.7 (20.7)	713 (713)
	300000	191 (191)	0.28 (0.28)	10.5 (10.4)	2.55 (2.53)	30.5 (30.3)	710 (709)
MIN (128×128)	100000	93 (86)	0.54 (0.40)	15.8 (17.0)	0.84 (1.52)	13.8 (85.2)	1121 (2040)
	200000	190 (174)	0.54 (0.40)	15.7 (17.1)	0.86 (1.52)	28.1 (17.2)	1123 (2030)
	300000	288 (264)	0.54 (0.40)	15.7 (17.1)	0.86 (1.51)	42.3 (26.1)	1123 (2019)

Table 3. Benchmark Performance (numbers in brackets denote after optimisation)

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 4 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 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 Speedup

Table 5 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 speed-up, the benchmark must

- contain several hundreds of simulation processes or LPs (over 500)
- contain sufficient computation (parallelism) to amortize the communication overhead by mapping more LPs to a PP (around 64-72)

4 Conclusions

Many simulationists are reluctant to adopt PDES due to the complexities of the parallel mechanisms involved. SPaDES is a parallel simulation framework designed for simulating large systems and experimental research in PDES. 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.

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.

program	mappings LP to PP	No. of PPs	Number of Processors				
			1	2	4	8	16
Super-ping (256)	16-1	16	3473.90	1935.33	854.95	474.23	**131.57**
	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 (16×16)	16-1	16	1850.57	936.40	441.32	261.86	**62.55**
	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 (128×128)	36-1	16	1524.29	1040.47	452.15	215.12	**87.29**
	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 4. Elapsed Time for Different Mapping Schemes

program	prob. size	CSim	Simscript	SPaDES (seq.)	SPaDES (parallel)	
					p=8	p=16
Super-ping	16	1.44	9.30	1.71	149.58	277.31
	64	8.10	38.20	9.25	101.06	148.13
	512	280.10	464.72	466.21	283.50	166.41
	1024	1187.00	1317.57	1885.56	683.06	306.48
	2048	4114.78	4331.63	7705.76	1839.04	835.47
Torus	8×8	1.94	10.14	2.40	44.06	67.35
	16×16	16.88	49.73	31.15	51.19	62.55
	32×32	315.12	395.10	505.73	221.45	110.28
MIN	64×64	25.04	-	33.62	72.05	40.40
	128×128	110.06	-	207.43	155.78	87.29
	256×256	582.28	-	1228.27	447.65	190.90
	512×512	2663.13	-	6474.05	1177.40	538.31

Table 5. Comparison with Sequential Simulators

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

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.