Experiments in Automated Load Balancing*

Linda F. Wilson
Institute for Computer Applications in Science and Engineering
Mail Stop 132C
NASA Langley Research Center
Hampton, Virginia 23681

David M. Nicol
Department of Computer Science
The College of William and Mary
P. O. Box 8795
Williamsburg, Virginia 23187

Abstract

One of the promises of parallelized discrete-event simulation is that it might provide significant speedups over sequential simulation. In reality, high performance cannot be achieved unless the system is fine-tuned to balance computation, communication, and synchronization requirements. As a result, parallel discrete-event simulation needs tools to automate the tuning process with little or no modification to the user's simulation code.

In this paper, we discuss our experiments in automated load balancing using the SPEEDES simulation framework. Specifically, we examine three mapping algorithms that use run-time measurements. Using simulation models of queuing networks and the National Airspace System, we investigate (i) the use of run-time data to guide mapping, (ii) the utility of considering communication costs in a mapping algorithm, (iii) the degree to which computational "hot-spots" ought to be broken up in the linearization, and (iv) the relative execution costs of the different algorithms.

*This work was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001. Nicol was also supported by NSF grant CCR-9201195.

1 Introduction

Discrete-event simulation can be used to examine a variety of performance-related issues in complex systems. Parallel discrete-event simulation (PDES) offers the potential for significant acceleration of solution time over sequential simulation. Unfortunately, high performance is often achieved only after rigorous fine-tuning is used to obtain an efficient mapping of tasks to processors. In practice, good performance with minimal effort is often preferable to high performance with excessive effort.

In a typical PDES, components of the system under examination are mapped into logical processes (LPs) that can execute in parallel. The LPs are distributed among the physical processors, and communication between LPs is accomplished by passing messages. There is a tension between distribution for load balance and distribution for low communication costs. If LPs are distributed among the processors such that interprocessor communication is kept low, some processors may sit idly waiting for something to do while others are overloaded with work. At the other extreme, a "perfectly-balanced" workload may induce high communication costs. Thus, load-balancing strategies must find a compromise between distributing work evenly and minimizing communication costs.

In earlier work [8], we described our early experiences in developing an automated load-balancing strategy for the SPEEDES simulation environment. In this paper, we discuss experiments with three different mapping algorithms used in conjunction with our automated scheme. Section 2 presents background material on SPEEDES while Section 3 describes the load-balancing methodology and the three mapping algorithms. Section 4 presents results from two models simulated on the Intel Paragon and compares the execution times obtained by the models' default partitionings with those obtained by the different mapping algorithms. Section 5 discusses the issue of mapping algorithms' execution times. Section 6 presents our conclusions.

2 SPEEDES

SPEEDES (Synchronous Parallel Environment for Emulation and Discrete-Event Simulation) is an object-oriented simulation environment that was developed at the Jet Propulsion Laboratory [5]. Designed for distributed simulation, SPEEDES supports multiple synchronization strategies (including Time Warp, Breathing Time Buckets, and Breathing Time Warp) that can be selected by the user at run time. In addition, SPEEDES provides a sequential simulation mode (with most of the parallel overhead removed) so that a particular simulation model can be executed serially or in parallel.

Developed using C++, SPEEDES uses an object-oriented computational model. The user builds a simulation program by defining simulation objects, object managers for those objects, and events. An object manager class must be defined for each type of corresponding simulation object. For example, in the SPEEDES model of the National Airspace System, an airport object class (AIRPORT_OBJ) must have a corresponding airport manager class (AIRPORT_MGR). The object managers are responsible for creating and managing the set of simulation objects. Thus, the user (through the object managers) is *completely responsible* for the mapping of the simulation objects to the processors.

While SPEEDES gives the user freedom to choose an appropriate mapping, it is quite likely that the user does not know *a priori* how to choose a good allocation of objects to processors. Most variations of the mapping problem are computationally intractable, so optimal mappings are extremely difficult to obtain. Furthermore, many users and potential users of PDES would prefer to let "the system" make such decisions, especially if the resulting performance is "good enough". In the next section, we present our methodology for automated load balancing in SPEEDES.

3 Automated Load Balancing

As discussed in [8], we modified SPEEDES to collect data on the workload characteristics of a simulation. Since each event is connected to exactly one simulation object, simulation objects determine the resolution of the partitioning. Thus, we collect computation data (number of events processed) for each object and communication data (number of messages sent) for each pair of simulation objects. The event and message counts are collected and saved in data files during a single run of the simulation. Ultimately, we will use this data to govern dynamic load balancing based on run-time information. In this report, we investigate the suitability of different algorithms for balancing computation and communication, with the expectation of embedding suitable candidates into a dynamic load-management system. Thus, at present, SPEEDES is instructed to collect the data (for one run) and use the data (in another run) through the use of run-time flags. The data is analyzed by a mapping algorithm that determines the load-balancing allocation of simulation objects to processors. In this paper, we experiment with three different mapping algorithms to examine the effectiveness of using run-time measurements in our automated load-balancing scheme. We also consider the execution costs of the algorithms themselves.

The first algorithm (LBalloc1) determines object placement based only on the computation weight of each object. Specifically, we use the longest-processing-time-first list scheduling algorithm in which the objects are ordered by decreasing computation weight and the heaviest unplaced object is placed on the processor with the lightest cumulative weight [1]. Notice that this approach concentrates on balancing the workload on the processors but ignores communication costs.

The second algorithm (LBalloc2, from [3]) arranges the LPs in a linear chain and then partitions the chain into as many contiguous subchains as there are processors, mapping one subchain per processor. The partition chosen minimizes the amount of work assigned to the most heavily-loaded processor, where the computation weight of a processor is the sum of measured computation weights of its LPs, and the computation weight of an LP is the number of committed events it executed. Given a linear ordering, this optimization problem can be solved very quickly, e.g. in $O(PM \log M)$ time, where P is the number of processors and M the number of LPs.

However, choice of an optimal linearization of LPs is computationally intractable; choice of a "good" linearization remains an open research problem. One idea (used in LBalloc2) is to linearize so as to keep heavily communicating LPs close to each other in the chain, thereby increasing the chance that they will be assigned to the same subchain. This intuition is realized by a recursive heuristic which at the first step "pairs" LPs using a stable matching

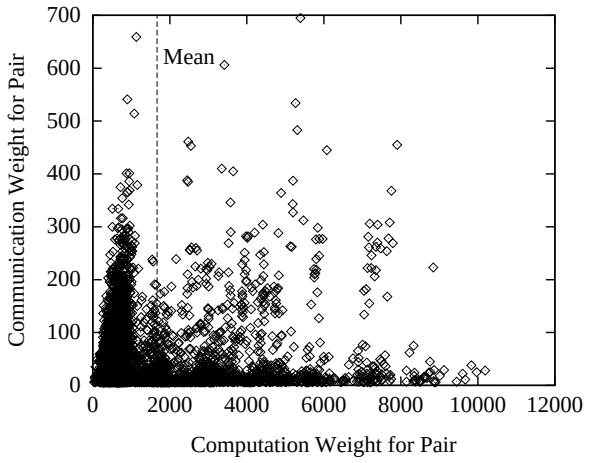
algorithm. Here the communication weight between two LPs is a measure of their attraction; a stable matching is one where if A and B are matched, and C and D are matched, it is not possible to break the matches and reassemble (e.g. A and C, B and D) and have higher attraction values for both matchings. Two LPs that are matched will be adjacent to each other in the linear ordering. We then merge matched LPs into super-LPs and compute the attraction between two super-LPs as the sum of the attractions between LPs in the two super-LPs. Matching these, the sets of LPs represented in two matched super-LPs will be adjacent to each other in the linear ordering. This process continues until there is a single super-LP.

Using the communication weight between LPs as the base attraction function is suitable as long as there is only weak correlation between the communication between two LPs and their computation weights. However, many simulations have "hot spot" simulation objects that perform most of the work. Typically, the hot spots have large amounts of communication with other hot spots. Use of the communication weight as the attraction function can cluster the hot spots together, which results in poor load balancing.

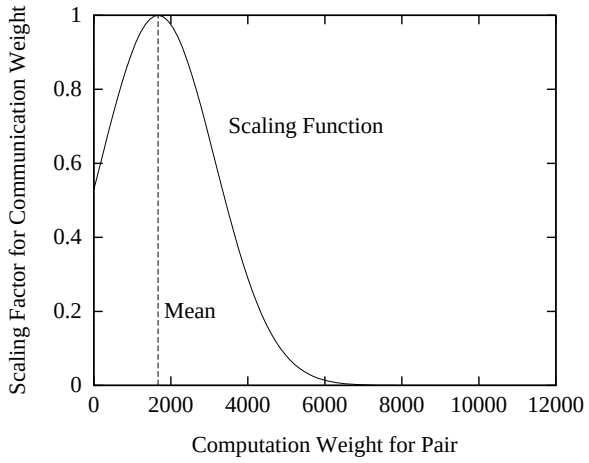
For the third mapping algorithm (LBalloc3), we modified the linear chaining algorithm to discourage clustering of extremes: pairs of very heavy objects or pairs of very light objects. Specifically, the attraction (communication weight) between each pair of objects is scaled by a Gaussian function based on the pair's computation weight, relative to the weights of the other pairs (with non-zero attractions). The scaling function has a value of one corresponding to the mean computation weight for all pairs with non-zero communication weights. The spread of the Gaussian corresponds to the variance of the paired computation weights. For example, Figure 1(a) shows a plot of first-round communication and computation weights for all pairs with non-zero communication weights.* Figure 1(b) shows the corresponding scaling function, and Figure 1(c) shows the resulting scaled communication weights. Notice that this scaling will encourage pairings in which the resulting paired computation weight will be close to the mean while it discourages pairings far from the mean.

While the automated load-balancing scheme can use any of the mapping algorithms to determine the allocation of simulation objects to processors, the SPEEDES user must choose to use that allocation when objects are created by the object managers. To assist the user, we added three functions to SPEEDES: `LB_is_avail()`, `is_local_object(objnum)`, and `is_local_object(objname)`. The `LB_is_avail()` While the automated load-balancing scheme can use any of the mapping algorithms to determine the allocation of simulation objects to processors, the SPEEDES user must choose to use that allocation when objects are created by the object managers. To assist the user, we added three functions to SPEEDES: `LB_is_avail()`, `is_local_object(objnum)`, and `is_local_object(objname)`. The `LB_is_avail()` function is used to determine if the load-balancing data is available for use during this run (i.e. it was collected during a previous run). Thus, the user can write an object manager that uses a default mapping if data is not available and the load-balanced mapping if it is. The `is_local_object(objnum)` function is used to determine if the simulation object with global ID number `objnum` should be created on this node while `is_local_object(objname)`

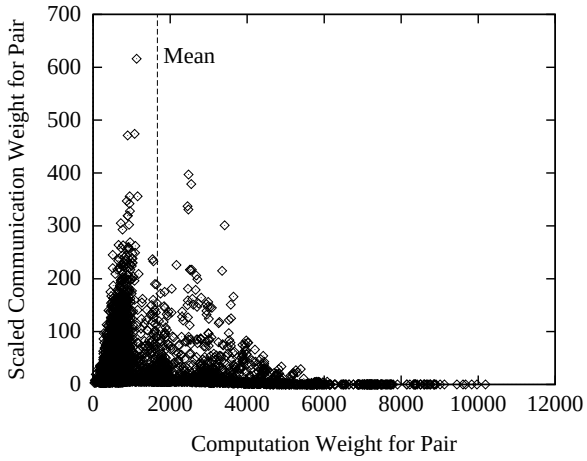
*This data comes from the DPAT algorithm, discussed in Section 4.



(a) Original Data



(b) Gaussian Scaling Function



(c) Scaled Data

Figure 1: Scaling Communication Weights for the LBalloc3 Algorithm

provides the same information based on a user-defined object name. Notice that the automated mapping will be inappropriate unless the global ID numbers and object names are consistent from one run to the next.

4 Results

We applied the three mappings in our automated load-balancing system to two simulation models: a fully-connected queuing network and the DPAT model of the National Airspace System [7]. In this section, we discuss the two models and present results from execution of the simulations on a 72-node Intel Paragon. Furthermore, we compare the results obtained from the three automated mappings with those obtained from the default partitioning.

4.1 Qnet Simulation

Queuing networks are often used as PDES benchmarks because they are commonly used in simulation studies, are relatively easy to program, and yet exhibit many of the difficulties experienced by more complex models[2, 6]. Thus, we are in good company using a fully-connected queuing network (Qnet) as the first test of our automated load-balancing system.

The Qnet simulation contains 1600 fully-connected servers. Since a homogeneous network of queues is balanced naturally without sophisticated algorithms, we study a deliberately unbalanced network. We define 50 servers to be "hot spots", where the probability that a customer exiting a server goes next to a hot spot is 0.15. Furthermore, we give each server a neighborhood of up to 30 neighbors, where the probability that a customer exiting one server goes to a neighbor is 0.65.[†] Finally, a customer goes to a random queue with probability 0.10 and back to the same queue with probability 0.10.

For the Qnet simulation, the default mapping of simulation objects to processors is so-called *block* partitioning. Specifically, n objects are allocated to p processors by placing the first (by global id) n/p objects on the first processor, the next n/p objects on the second processor, and so forth.

As we conducted our investigation, we were surprised to discover that in the SPEEDES implementation on the Intel Paragon, there is very little difference between the overall communication cost of placing two objects on the same processor or separating them. As a consequence, any method that balances the workload well will perform well in this context. It is unrealistic to expect this to be true either on other architectures or with different simulation testbeds. For the purposes of exploring the effectiveness of the algorithms in such situations, we parametrically inflate the interprocessor communication costs by adding a timed delay to the posting of each interprocessor event. For the Qnet simulation, we examined three different cases:

- Qnet #1: delay = 0 msec, 50 initial customers per server, simulated execution time = 1500 seconds
- Qnet #2: delay = 10 msec, 25 initial customers per server, simulated execution time = 250 seconds

[†]Note that if Server A is a neighbor of Server B, Server B is not necessarily a neighbor of Server A.

- Qnet #3: delay = 15 msec, 25 initial customers per server, simulated execution time = 250 seconds.

To put these delays in perspective, the time required to execute a SPEEDES event on the Paragon is approximately 1 msec. Thus, the 10 and 15 msec delays represent the sorts of relative network delays one might see with medium-grained workloads running on a workstation cluster or fine-grained workloads on a large-scale parallel architecture.

For the simulations discussed in this paper, we used the optimistic Breathing Time Warp (BTW) synchronization protocol [4] that combines the Time Warp and Breathing Time Buckets protocols. At the beginning of each global virtual time (GVT) cycle, messages are sent aggressively using Time Warp. Later in the cycle, all messages are sent risk-free using Breathing Time Buckets. Two run-time parameters in SPEEDES determine the amount of risk in the BTW protocol: Nrisk and Nopt. For the first Nrisk events processed after the last GVT computation, messages are released immediately to the receiver (Time Warp). For the events from Nrisk to Nopt (where Nrisk $<$ Nopt), event messages are cached locally and the event horizon is computed (Breathing Time Buckets).

The Breathing Time Warp parameters used for Qnet Simulation #1 were Nrisk = 1500 and Nopt = 3000. For Qnet Simulations #2 and #3, the parameters were Nrisk = 75 and Nopt = 150. These parameters were determined by conducting various runs of the Qnet simulation (using the default partitioning) on different numbers of processors. Overall, these parameters gave the shortest execution times.

Figure 2 presents results for Qnet #1. Notice that when eight or more processors are used, the automated load-balancing schemes give results that are noticeably better than the default partitioning, while the two linear-ordering mappings (LBalloc2 and LBalloc3) give almost identical results. When four processors are used, LBalloc1 and the default partitioning yield the fastest execution time and yield significantly better performance than LBalloc2 and LBalloc3. This occurs because LBalloc2 and LBalloc3 are constrained somewhat from balancing load, based on their linear orders. Yet, with few processors the communication load is inconsequential compared to the computation load. The situation changes though as the number of processors is increased. The mappings that explicitly balance load are nearly identical in performance and leave the default performing rather poorly. Also, notice the relatively large difference between LBalloc2 and LBalloc3 at four processors. Here we see the effects of LBalloc2 naively clumping hot spots together in the linear order, making it harder to distribute the workload evenly. Surprisingly, the effects of this mishap are mitigated though by increasing the number of processors. However, for the case of four processors, the Gaussian-based weighting of communication costs in LBalloc3 significantly reduces the hot-spot clumping.

Figure 3 presents results for Qnet #2, where the sending of interprocessor messages is delayed by 10 msec. When four processors are used, the default and LBalloc1 yield mappings that are noticeably worse than those of the communication-sensitive mappers LBalloc2 and LBalloc3. With larger numbers of processors, the results for LBalloc2 and LBalloc3 are very similar and, owing to their sensitivity to communication costs, are noticeably better than those for LBalloc1.

Figure 4 demonstrates the effect of even larger communication costs when large numbers of processors are used. When 24 processors are used, LBalloc3 is 10 seconds faster—about

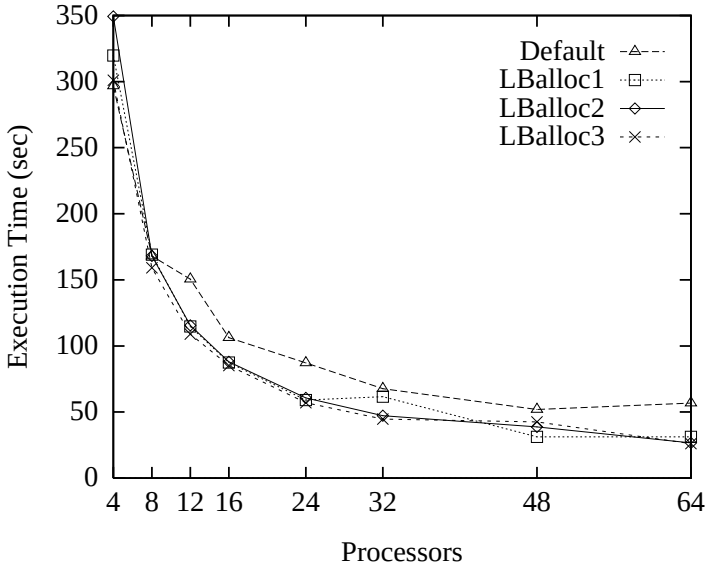


Figure 2: Results for Qnet #1 (delay = 0 msec)

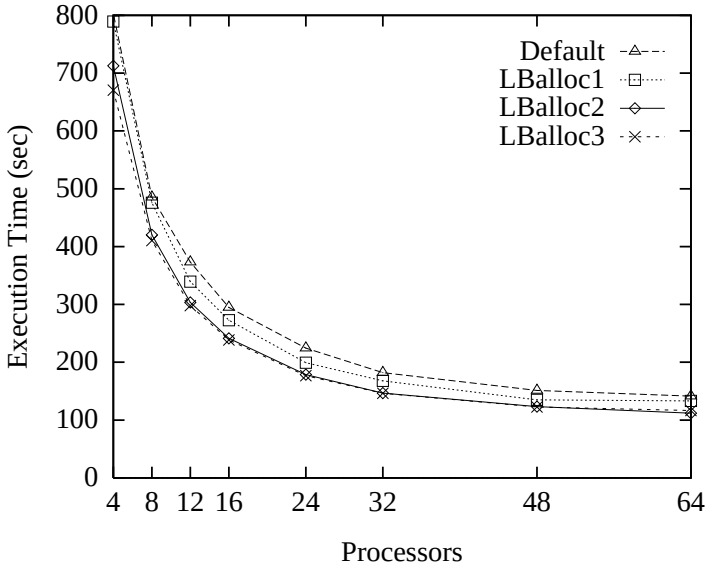


Figure 3: Results for Qnet #2 (delay = 10 msec)

5%—than LBalloc2, which indicates that some improvement was made by breaking up some of the hot spots. In general, there is little difference between LBalloc2 and LBalloc3. Both provide markedly better performance than either of the communication-cost insensitive mappings.

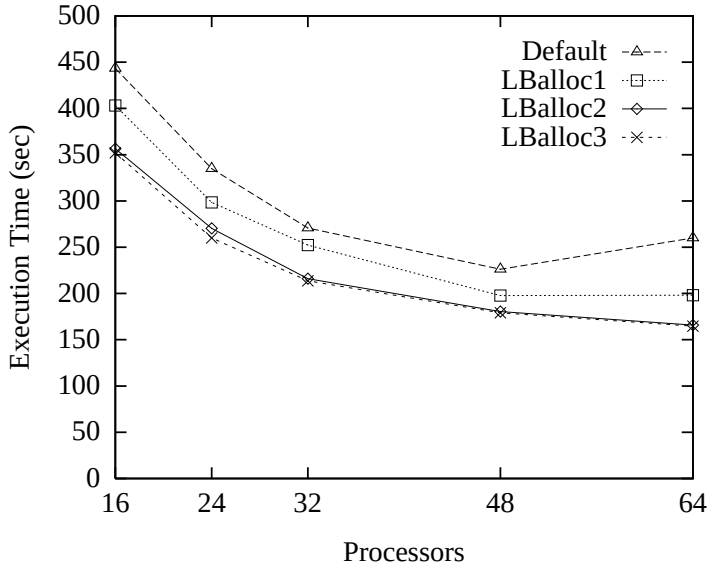


Figure 4: Results for Qnet #3 (delay = 15 msec)

4.2 DPAT: A Model of the National Airspace System

For the last several years, the MITRE Corporation has been studying the National Airspace System (NAS), which encompasses all commercial and general aviation air traffic in the United States [7]. On a typical day, the NAS consists of 45,000 to 50,000 flights from approximately 16,000 airfields. The commercial air traffic is handled by roughly 1000 airports while 80% of the general aviation traffic is handled by the top 500 airports. In addition to the airfields, the NAS contains 701 three-dimensional regions called sectors that cover the airspace between airports.

MITRE recently developed a PDES model of the NAS called DPAT (Detailed Policy Assessment Tool) that is used to examine the average delay encountered by aircraft under various weather and traffic conditions. As discussed in [7], the physical NAS system is a good candidate for PDES because the aircraft, air traffic controllers, and airports operate naturally in parallel.

The DPAT model contains SPEEDES simulation objects for 520 airports and 701 sectors. Events in the system include takeoffs, landings, and transfers of aircraft between sectors. Scheduling data from the Official Airlines Guide (OAG) is used to schedule commercial flights while general aviation flights are scheduled stochastically. Details of this model can be found in [7].

The DPAT simulation begins by reading in large files of flight and airplane data to initialize system parameters and schedule initial events. When we first executed DPAT on the Intel Paragon, we encountered severe performance problems due to memory paging. In particular,

the aggregate size of the executable and data exceeded the roughly 23 MBytes per node of user-available memory. After discussing the problem with MITRE, we modified the program to use a subset of the aircraft data. This modification eliminated the memory problems (when multiple processors were used) without reducing the amount of computational work required.

The DPAT simulation organizes the airports and airspace sectors into geographic groupings called centers. For example, the La Guardia, Kennedy, and Newark airports in New York and New Jersey belong to a center that contains the airspace sectors around those airports. The Los Angeles and San Francisco airports belong to different centers because of the distance (and number of other airports) between them. These geographic centers form the basis for DPAT's default partitioning of simulation objects to processors.

Given that a flight must travel through contiguous sectors between airports, it is logical to assume that a geographical partitioning of the airports and sectors will reduce communication costs. The problem, however, is that the geographic distribution may not result in an even distribution of work. With DPAT, the airports and sectors are divided among 22 centers, where the "laziest" center is associated with 421 events and the busiest center has 20963. Even if more than 22 processors are used, only 22 will receive work. This center-based approach serves as the default mapping for DPAT.

We executed DPAT on the Intel Paragon using the default partitioning and the three automated algorithms. To determine the effect of interprocessor communication, we evaluated two scenarios:

- DPAT #1: delay = 0 msec
- DPAT #2: delay = 10 msec.

For consistency, all of the runs were initially taken with Nrisk = 250 and Nopt = 500, which were determined from the best timings of the default partitioning.

Figure 5 presents results for DPAT #1 in which no additional communication delays were added. Notice that LBalloc1 gives the best overall results while LBalloc2 and LBalloc3 give very similar results. Table 1 presents a comparison of LBalloc1 and LBalloc2 data for DPAT #1. Notice that the difference in their respective bottleneck weights (measured as the largest number of committed events executed on any processor) gradually increases with the number of processors. Furthermore, notice the dramatic difference between LBalloc1 and LBalloc2 in the total number of off-processor messages. In this case, the execution time corresponds directly with the maximum computation weight of a processor, and reducing the number of off-node messages has little or no effect. This data indicates that there is little difference in SPEEDES (on the Paragon) between the costs for on-processor and off-processor communication.

As mentioned earlier, we initially ran DPAT using Nrisk = 250 and Nopt = 500. When we added a communication delay of 10 msec, we had trouble running DPAT on large numbers of processors because the number of optimistically processed events grew much faster than the number of committed events, resulting in memory problems on the Paragon. Figure 6 presents results for the combinations that were able to run to completion using the initial parameters. In Figure 7, we present additional results that were obtained by modifying the parameters for runs with 32 or more processors. Specifically, we reduced Nrisk and Nopt

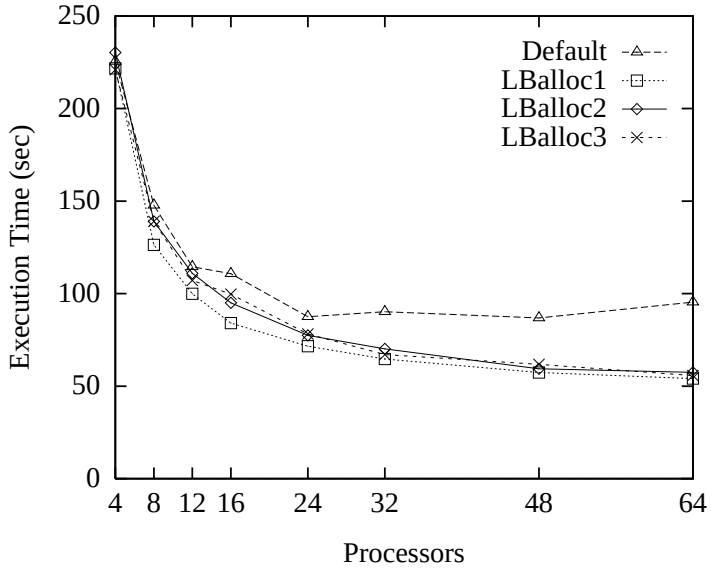


Figure 5: Results for DPAT #1 (delay = 0 msec)

Table 1: Comparison of LBalloc1 and LBalloc2 for DPAT #1

#	Max. Comp. Weight[a]		Off-Node Messages		Exec. Time (sec)	
Proc.	LBalloc1	LBalloc2	LBalloc1	LBalloc2	LBalloc1	LBalloc2
4	121,474	121,538	179,222	54,510	221.8	230.3
8	60,739	61,241	205,711	74,075	126.3	139.0
12	40,496	40,818	228,322	85,096	99.9	110.8
16	30,374	30,787	235,065	102,734	84.1	95.0
24	20,253	20,682	261,144	113,984	71.6	77.4
32	15,194	16,009	285,942	141,032	64.7	70.1
48	10,142	10,732	334,786	175,977	57.4	59.4
64	7,611	8,446	388,998	218,582	54.1	57.5

[a]Computational workload is measured as the maximum number of committed events executed by any one processor.

until we could get all four versions to run for a particular number of processors. For 32 and 48 processors, we used Nrisk = 250 and Nopt = 400. For 64 processors, we used Nrisk = 150 and Nopt = 200.

In Figures 6 and 7, the results obtained by LBalloc2 and LBalloc3 are significantly better than those obtained by LBalloc1 or the default (center-based) partitioning. These results confirm the significance of minimizing communication when communication costs are high. Finally, notice that LBalloc3 has a slight advantage over LBalloc2 when large numbers of processors are used, probably because the hot spots are more evenly distributed among the processors.

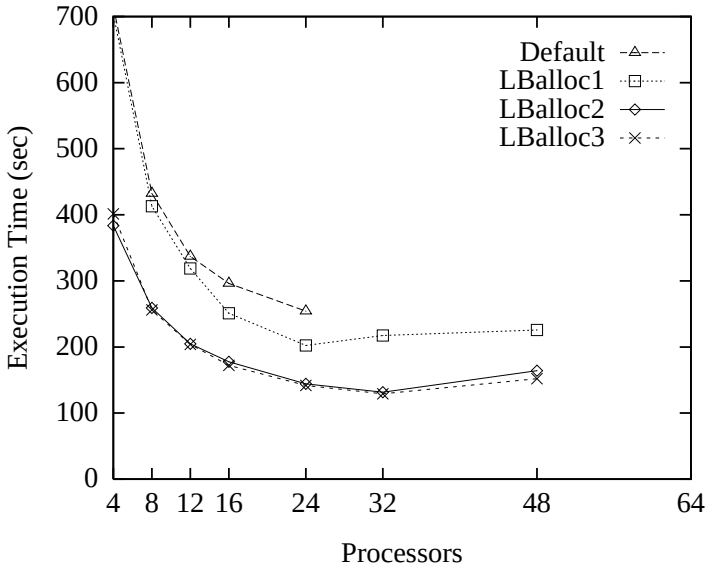


Figure 6: Results for DPAT #2 (delay = 10 msec)

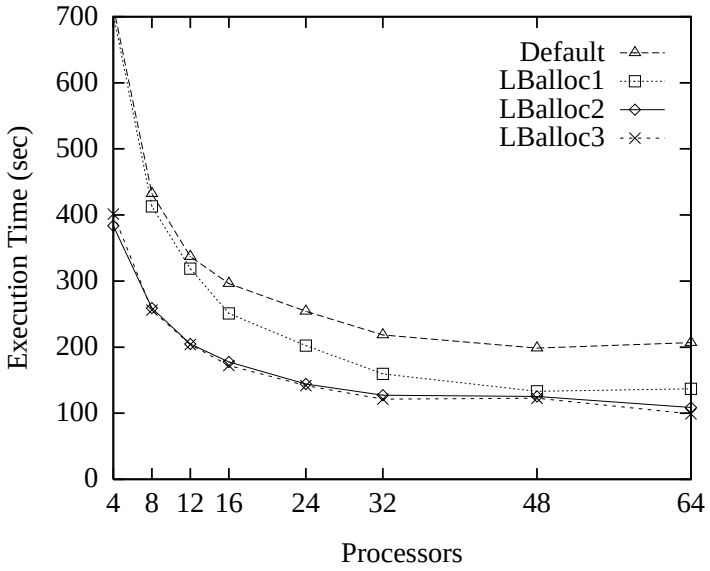


Figure 7: Results for Modified DPAT #2 (delay = 10 msec)

5 Mapping Algorithm Execution Time

If we are to use any of these mapping algorithms at run-time, we must consider the costs of doing so. Towards this end, we considered the execution time required to produce the mappings for LBalloc1, LBalloc2, and LBalloc3 on the DPAT benchmark (where approximately 1200 objects are mapped). More or less regardless of the number of processors targeted, LBalloc1 required $1/10th$ of a second, LBalloc2 required 33 seconds, and LBalloc3 required 39 seconds. Measuring the various contributions to this cost, we discovered that the dominant cost is due to pairings between simulation objects. If either method is to be implemented at run time clearly more work is needed to intelligently cut down on the number of pairings considered, both from the point of view of computation time, and from the point of view of transporting the communication measurements to the processor responsible for executing the mapping routine. On the other hand, LBalloc1 is sufficiently fast to use at run time, and requires much less information. At least for SPEEDES on the Paragon, LBalloc1 is clearly the mapping algorithm of choice. However, since the dominant cost of LBalloc2 and LBalloc3 is in the linearization (the workload balancing part is very fast), one might only consider the linearization part very infrequently, but balance workload more frequently (this is the approach used in [3]). It is also important to note that half a minute execution time is notable only in the context of repeated application at run time. When used as we have—once, at initialization—half a minute is a minor cost.

6 Conclusions

We have studied the suitability of various mapping algorithms for automated mapping of SPEEDES models. One algorithm, LBalloc1, is a well-known multiprocessor scheduling algorithm and ignores any information concerning communication costs between objects. LBalloc2 explicitly considers communication costs by constructing a linear ordering of objects wherein objects that communicate heavily tend to be close to each other in the ordering. The linear ordering is then partitioned subject to a constraint that only contiguous subchains be mapped to processors; subject to this constraint, we efficiently find the mapping that minimizes the computational workload of the most heavily-loaded processor. LBalloc3 corrects the possibility of "hot spots" producing bad linearizations in LBalloc2.

Our experiences on a large queuing network model and on a realistic model of the National Airspace System show that for SPEEDES on the Paragon, LBalloc1 is clearly the mapping algorithm of choice. While it does not explicitly consider communication costs, there evidently is very little difference in SPEEDES on the Paragon between the cost of interprocessor and intraprocessor message passing. Furthermore, LBalloc1 is fast enough to be repeatedly called in a dynamic remapping context. However, whatever the cause of this lack of distinction between on-processor and off-processor messaging, one cannot expect it to hold true for other tools and/or other architectures. By artificially increasing the communication cost for interprocessor messages, we studied the benefit of LBalloc2 and LBalloc3 in communication-sensitive contexts. While we did find instances where LBalloc3 improved upon LBalloc2, for the most part the differences were slight. However, in the presence of significant communication delays, they do both provide substantially better mappings than

communication-blind mappings. However, in their present implementation, the execution time used to produce a linear ordering is too high to be considered except for very long running simulations.

Future work will involve coming to understand the reason for SPEEDES communication-cost insensitivity on the Paragon, and implementing dynamic object migration in SPEEDES.

References

[1] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies", *SIAM Journal of Applied Mathematics*, Vol. 17, No. 2, pp. 416–419, March 1969.

[2] D. M. Nicol, "Parallel Discrete-Event Simulation of FCFS Stochastic Queuing Networks", *Proceedings ACM/SIGPLAN PPEALS 1988: Experiences with Applications, Languages and Systems*, pp. 124–137.

[3] D. M. Nicol and W. Mao, "Automated Parallelization of Timed Petri-Net Simulations", *Journal of Parallel and Distributed Computing*, Vol. 29, No. 1, pp. 60–74, August, 1995.

[4] J. Steinman, "Breathing Time Warp", *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation (PADS '93)*, pp. 109–118, July 1993.

[5] J. Steinman, "SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation", *International Journal in Computer Simulation*, Vol. 2, No. 3, pp. 251–286, 1992.

[6] J. Steinman, "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation", *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, SCS Simulation Series, Vol. 23, No. 1, pp. 95–103, January, 1991.

[7] F. Wieland, E. Blair, and T. Zukas, "Parallel Discrete-Event Simulation (PDES): A Case Study in Design, Development, and Performance Using SPEEDES", *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pp. 103–110, June 1995.

[8] L. F. Wilson and D. M. Nicol, "Automated Load Balancing in SPEEDES", *Proceedings of the 1995 Winter Simulation Conference*, December, 1995.