ORSA Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Commentary—Will Parallel Simulation Research Survive?

Yi-Bing Lin,

To cite this article:
Yi-Bing Lin, (1993) Commentary—Will Parallel Simulation Research Survive?. ORSA Journal on Computing 5(3):236-238. https://doi.org/10.1287/ijoc.5.3.236

ORSA Journal on Computing
Vol. 5, No. 3, Summer 1993

0899-1499/93/0502-0238 $01.25

COMMENTARY

Will Parallel Simulation Research Survive?

YI-BING LIN / *Bellcore, MRE 2D297, 445 South Street, Morristown, NJ 07962; Email: liny@mookie.bellcore.com*

In his feature article, Fujimoto pointed out that the success of parallel simulation depends not only on the speedup that can be achieved through parallel simulation but also on the effort required to develop the parallel code. The general simulation community will only recognize parallel simulation technology if several practical simulation applications are found such that the total cost of the code development and code execution for parallel simulation is much less than the cost of sequential simulation. The sequential execution times of most examples listed in Table II in the feature article were less than two hours. It is difficult to justify parallel simulation as the right approach to follow if one needs to spend one additional day (or more) to develop parallel code in order to reduce a two hour execution time to several minutes.

Most parallel simulation research focuses on reducing execution time of parallel simulation. Fujimoto proposed several approaches to address the parallel code development issue. I will consider this issue in a different aspect as a supplement. In addition, I will discuss how to predict the performance of parallel simulation, an issue not explicitly discussed in the feature article.

Another issue not addressed in the feature article is the distributed computing platform for parallel simulation. Many examples listed in Table II in the feature article were run on shared memory architectures, and none of the examples were run on a network of workstations, a widely used hardware platform which cannot be ignored by parallel simulation researchers. It is difficult to gain acceptable speedup for parallel simulation in a network of workstations due to long communication delays. One possible solution is to connect the workstations using a high speed network. Another solution is to carefully map the processes to the processors such that inter-processor communications are reduced. How to do the latter efficiently is still an open issue.

1. Predicting the Performance of Parallel Simulation

It is important to predict the performance of parallel simulation before a simulationist invests a substantial effort in developing parallel codes. For example, if the inherent parallelism for the simulation application is not sufficiently large, it is not worthwhile to take the parallel simulation approach. The inherent parallelism can be analyzed by a technique called *critical path analysis*. The critical path analysis algorithms for the case when every process is executed by a dedicated processor were proposed by Berry and Jefferson[2] and Livny.[9] Lin[6] generalized the algorithms for different process-to-processor mappings under different process scheduling policies. Wong, Hwang, and Lin[13] developed a critical path analyzer for Chandy-Misra simulation with different process mapping and scheduling policies.

The critical path analyzers are small segments of code inserted into the sequential simulation programs. The parallelism is computed in real time along with the execution of the sequential simulation. A good example of using the critical path analysis is given below. In the past, sequential simulations of asynchronous transfer mode networks and signaling networks were built to investigate the performance of telecommunications networks. The sequential simulations worked fine if the network sizes were small. When the network size increased (e.g., up to hundreds of nodes), the sequential simulation could not produce stable statistics before the computing resources were used up. In this case, critical path analysis could be used to predict whether parallel simulation is the right approach to follow.

Future research directions for parallel simulation performance prediction include the determination of event execution times (a parameter used in the algorithm), determination of the execution time for the critical path analysis before reliable performance prediction can be achieved, and development of critical path analysis for different parallel simulation protocols.

2. Designing a Future Simulation Environment

Regarding the design of a parallel simulation language, Fujimoto suggested that, ideally, one should be able to develop a simulation model without being overly concerned with the execution environment, and achieve acceptable performance should it be executed on a parallel machine. I agree. Simulation environments in the future should be designed for different levels of users. A naive user should not see the primitives for a parallel simulation engine (such as "sim_write_lock" in Sim++). An experienced user may set the parameters for the execution engine (such as the frequency of garbage collection in Time Warp) to improve the performance. A sophisticated user may modify the existing simulation engine or add a new parallel simulation engine without changing the other parts of the simulation environment (thus, the environment should be able to accommodate different simulation engines).

To achieve this goal, the object-oriented model is an appropriate paradigm to follow. Many modern sequential simulation languages are object-oriented[1] because, most importantly, implementing simulation entities by objects closely matches the way simulationists think of the problem. Several advantages of object-oriented software

design[3] also apply to the simulation environment design, including encouraging the reuse of simulation code, producing simulation code that is more resilient to change, and allowing an intelligent separation of concerns in a simulation design, which reduces the possibility of development mistakes.

As pointed out in the feature article, for a parallel simulation researcher, the beauty of the object-oriented model is its similarity to the logical process model of parallel simulation. It is natural to implement an object-oriented simulation model on a parallel simulation engine by translating the objects into logical processes synchronized by the parallel simulation protocol. In modern parallel simulation languages such as Sim++, users are allowed to build the simulation model without considering the underlying simulation engine. In other words, the effort to program a simulation model for parallel simulation is the same as for sequential simulation.

Another important aspect is that the object-oriented model makes it easier to accommodate a new parallel simulation engine (assuming that the engine follows the logical process model). This aspect is not well addressed in the parallel simulation research field. Experience suggests that to achieve impressive speedup, the parallel simulation engine must be tailored to fit the specific simulation application. A good example was given in the feature article. Reed, Malony, and McCredie[11] reported discouraging results for queueing network simulations using the Chandy-Misra simulation engine. However, Fujimoto[4] showed that by exploiting lookahead information, the speedup can be significantly improved. For different types of applications, the methods of exploiting lookahead are different. For example, the lookahead technique for a first-come-first-serve queue[10] is different from the techniques for round-robin queues[12] and priority queues[8]. Thus, lookahead generation in the simulation engine must be carefully tailored for different applications.

Another example is given by Lin.[7] In existing Time Warp implementations, all processes use the same cancellation strategy. In a queueing network simulation, the performance of the cancellation strategies (either *lazy* or *aggressive*) for a Time Warp process (which simulates a queue element) depends on the simulated traffic ρ to the queue element. For this type of simulation application, the rollback mechanism should be modified so that the cancellation strategy can be switched (in a Time Warp process) based on the measured ρ value.

A third example is the parallel simulation of blocking networks in tandem configurations.[5] For such an application, it was shown that there is no need to propagate a rollback. Also, the local clock of the process at the last stage of the tandem network is equal to the *global virtual time* (*GVT*) or the minimum timestamp of all unprocessed events in the system. Thus, the rollback mechanism and the GVT computation can be simplified. If the general Time Warp is used to run the simulation, the best speedup cannot be achieved because parallel simulation pays the overhead for features it never uses. However, it is too expensive to implement a new parallel simulation engine for this application. Thus, a better solution is to tailor the general Time Warp engine such that the GVT computation routine and the rollback routine are simplified.

In future simulation environments, methods must be provided for the sophisticated user to tailor the existing parallel simulation engine for a specific application. It is important to identify the components of a general parallel simulation engine (e.g., the rollback mechanism, GVT computation mechanism, checkpoint interval selection mechanism, distributed termination mechanism, memory management mechanisms, and so on) and the interfaces among the components so that a user can modify a component without affecting the others. After the components are well defined, the simulation engine can be conveniently implemented using the object-oriented paradigm. Adding features or modifying the existing components can be easily done by *polymorphism* (the ability to call a variety of functions using exactly the same interface).

3. Conclusion

This commentary discussed two issues that supplement Fujimoto's article. First, performance prediction methods for parallel simulation must be developed. It is important to predict the performance of parallel simulation before effort is invested in building the parallel code for a specific simulation application. Second, information hiding must be supported in future simulation environments so that a user who builds the simulation models does not see the primitives for the underlying simulation engine (although a user may need to have knowledge of the simulation engine). To improve performance, methods must be provided for a user to tailor the parallel simulation engine without affecting other parts of the simulation environment. Such an environment can significantly reduce the parallel code development time.

References

1. T.G. Beaumariage and R. Ege (eds.), 1992. *Proceedings of Object-Oriented Simulation*, Society of Computer Simulation, Newport Beach, CA.
2. O. Berry and D. Jefferson, 1985. Critical Path Analysis of Distributed Simulation, *Proceedings of the 1985 SCS Multiconference on Distributed Simulation*, pp. 57–60.
3. G. Booch, 1991. *Object Oriented Design with Applications*, Benjamin Cummings, Redwood City, CA.
4. R.M. Fujimoto, 1988. Lookahead in Parallel Discrete Event Simulation, *Proceedings of the 1988 International Conference on Parallel Processing*, Vol. III, pp. 34–41.
5. Y.-B. Lin, 1993. Parallel Simulation of Blocking Networks with Tandem Configuration, Technical Report TM-ARH-028745, Bellcore, Morristown, NJ.
6. Y.-B. Lin, 1993. Parallelism Analyzers for Parallel Discrete Event Simulation (to appear in *ACM Transactions on Modeling and Computer Simulation*).
7. Y.-B. Lin, 1993. Selecting the Cancellation Strategy for Optimistic Parallel Simulation, Technical Report TM-ARH-022588, Bellcore, Morristown, NJ.

8. Y.-B. LIN and E.D. LAZOWSKA, 1990. Exploiting Lookahead in Parallel Simulation, *IEEE Transactions on Parallel and Distributed Systems* 1:4, 457–469.
9. M. LIVNY, 1985. A Study of Parallelism in Distributed Simulation, *Proceedings of the 1985 SCS Multiconference on Distributed Simulation*, pp. 61–64.
10. D.M. NICOL, 1988. Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks, *Proceedings of the ACM SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages, and Systems*, pp. 124–137.
11. D.A. REED, A.D. MALONY and B.D. MCCREDIE, 1988. Parallel Discrete Event Simulation Using Shared Memory, *IEEE Transactions on Computers* 14:4, 541–553.
12. D.B. WAGNER and E.D. LAZOWSKA, 1989. Parallel Simulation of Queueing Networks: Limitations and Potentials, *Proceedings of 1989 ACM SIGMETRICS and Performance '89 Conference*, pp. 146–155.
13. Y.-C. WONG, S.-Y. HWANG and Y.-B. LIN, 1993. A Parallelism Analyzer for Conservative Parallel Simulation, Technical Report TM-ARH-022727, Bellcore, Morristown, NJ.