ORSA Journal on Computing

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

Commentary—Practical Parallel Discrete Event Simulation

Brian W. Unger, John G. Cleary,

To cite this article:
Brian W. Unger, John G. Cleary, (1993) Commentary—Practical Parallel Discrete Event Simulation. ORSA Journal on Computing 5(3):242-244. https://doi.org/10.1287/ijoc.5.3.242

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

0899-1499/93/0502-0242 $01.25

COMMENTARY

Practical Parallel Discrete Event Simulation

BRIAN W. UNGER and JOHN G. CLEARY / *Jade Simulations International Corporation and University of Calgary, Calgary, Alberta, Canada; Email: unger@jade.ab.ca and cleary@cpsc.u calgary. ca*

The feature article addresses issues that are now of crucial importance to this field. Model development for efficient parallel execution *is* too difficult. Our initial goal at Jade was to create a parallel simulation environment that could be used to solve a wide variety of practical problems. Fujimoto's paper outlines many of the reasons why we have not yet achieved this goal.

Our experience at Jade supports Fujimoto's statement that it typically takes "highly skilled experts" to develop models for real applications that can achieve significant speedup due to parallelism. Most of the reasons he presents for this unfortunate situation also agree with our experience. However, we believe there are some important differences in how these difficulties can be addressed.

An approach to the design of efficient parallel models is sketched below. This discussion identifies four key indicators of performance and suggests adding one more silver bullet to Fujimoto's list: *tools* that support the incremental development of efficient models.

1. Developing Efficient Parallel Models

The following sketch assumes an optimistic synchronization mechanism such as the Jade *Sim++* and Time Warp development environment.[1,5] Our early experience[3] agrees with Fujimoto's claim that the model development problem is generally harder for conservative methods.

Given a simulation problem, the first step in model design is to *partition* the target model into logical processes based on heuristic design principles (e.g., maximize the number of processes while minimizing process interactions, schedule events as early as possible, maximize the time-advance per event, avoid centralized data access, push versus pull data). The resulting model, i.e., main program and processes and objects, are then implemented. Simulation experiments can then be conducted on a multiprocessor platform. This involves *mapping* processes to processors at load time in a way that attempts to balance the computation across the processors. If performance is an issue then *analyses* of the parallel execution can be conducted to determine bottlenecks and model design problems.

One practical difficulty not mentioned in Fujimoto's paper is that one must be able to partition and map a model sufficiently well so that the first run can be completed. Once the first run is completed, either normally or as a graceful shutdown, the statistics gathered can be used to guide the direction of improvements to partitioning and mapping. There is no guidance for the first run and we have many examples where heuristics and intuition do not lead to good performance. This is particularly difficult when memory is limited (the typical case) and the initial partition and map must be good enough so that available memory is not exceeded on any one processor.

Parallelism is interesting in that it tends to be lost at every step in the transformation from a real system to a useful model. Many physical systems of interest have a high degree of parallelism. However, as noted by Fujimoto, most models require events that have no counterpart in the actual system, e.g., statistics collection. It is therefore crucial that parallelism be measurable in a way that ties losses to model structure and behavior.

Our approach to developing a model that exhibits acceptable execution speed is thus a repetition of three steps: *partitioning*, *mapping*, and *analysis of performance statistics*. The last step requires the measurement of performance parameters which can guide subsequent changes to the model to improve speedup.

2. Four Important Performance Parameters

There are four key, measurable parameters that constrain the performance of a Time Warp system. Two of these parameters characterize the model: granularity and time-advance. The other two characterize the Time Warp executive: state saving overhead (or whatever mechanism enables rollback) and overhead associated with each event (e.g., event list insertion, maintenance of information that enables rollback, message interactions). This division is approximate in that state saving overhead can depend on both the model and executive.

Granularity for a given simulation event is the execution time for that event. Time-advance for a given event is the difference in virtual time between the time the event is scheduled and the time when the event is to occur. Granularity and time-advance can be computed for each event or averaged for each phase of each logical process, for each logical process throughout the simulation, for the entire simulation, etc.

In general, the objective of partitioning is to maximize both granularity and time-advance. If granularity is large compared to Time Warp state saving and event overheads then speedup will not be constrained by these overheads. The model developer can control granularity to some extent by decomposing the model in different ways to vary the number of Time Warp events (versus local events within a logical process). Increasing the size of processes increases granularity but also decreases the potential parallelism.

An upper bound on achievable speedup that we have found useful is the sum of all time-advances in the simulation divided by the duration of the simulation. For example, if the average time-advance is $T/20$ for a total of 500 events scheduled in a simulation of duration T that involves 200 logical processes then the maximum possible speedup is 25.

The model developer also has some control over time-advance. A partition and the implementation of logical processes should aim to both schedule events early and schedule events as far into the future as possible. Small model changes can yield dramatically different average time-advances.[2] We have also seen telecommunication applications where the number of messages, i.e., events, versus virtual time is small compared to the number of processes. Here the time-advance upper bound is relevant.

3. Incremental Model Development Tools

It has taken several decades to learn how to develop efficient sequential simulations. We have been highly motivated to improve sequential model execution efficiency as discrete event simulations are commonly computation intensive. The typical simulation project runs into performance limitations at some point in its life often requiring model simplifications or reductions in the number and length of experiments.

Sequential performance analysis tools and the ability to incrementally improve performance have been important. These methods can also be applied to the development of efficient parallel simulations. We believe that flexible tools that enable the measurement of model parallelism, granularity, time-advance, and other performance indicators such as rollbacks will be required to enable the development of efficient parallel models.

A family of tools that we have envisioned centers first on the measurement of performance parameters as a function of both virtual time and execution time for each process in a simulation model. For example, the maximum possible parallelism that exists for a given partition and the maximum parallelism that exists for a given partition and mapping can both be computed during a single sequential run. The actual parallelism achieved in a given parallel execution can also be measured. Similarly, event granularity, time-advance, rollbacks, and computation progress can be measured versus time per process.

We have built several of these tools and have had some success with their application to problems in telecommunications, microelectronics design, and military operations. Jade's critical path analysis tool estimates maximum potential parallelism versus both virtual time and execution time for a given model and partition. Another tool estimates the parallelism achievable for a given model partition and mapping, and the actual parallelism achieved in a particular run. Other tools measure rollbacks, positive and negative messages, and simulation progress for each process. All are versus virtual time and execution time.

Another category of tools required are libraries to support optimized state saving within Time Warp for specific applications. Examples include incremental state saving for large, sparsely modified data structures, replication of shared data structures including support for maintaining consistency, and support for read-only data structures.

4. Summary

We agree with the thrust of Fujimoto's paper: parallel discrete event simulation will only succeed as a generally applicable technology when models are much easier to develop. In this commentary, we have described what we believe to be the current state of the art in a commercial setting, that is, experienced people carefully designing new models using special purpose languages and performance analysis tools that are specifically designed for Time Warp.

This adds a fifth silver bullet to Fujimoto's list: *tools that support the incremental performance improvement of parallel simulations*. In the short term, we believe that improvements in performance analysis tools, class libraries for custom state saving, and application specific libraries are the ways that progress will be made. In support of the feature article's thesis, it is a very expert-intensive process and requires an organization that contains both its own Time Warp implementation and Time Warp experts.

In the long term, all of the problems must be solved. Our experience indicates that the hardest ones are support of shared state including global variables, automatic partitioning and mapping, automatic balancing of computation, and intelligent reduction of state saving and event execution overheads.

We would like to make a strong statement about the usability hurdles that a practical widely used Time Warp based simulation system must surmount:

- It must be able to run *existing* models with *no* modification. It need only provide performance comparable to sequential on one or two processors at first. This is much harder in some existing languages than others. Two cases where it is easier include Prolog[4] and VHDL (the emerging standard hardware description language which currently does not permit the use of any shared or global state whereas most other modeling languages do).
- It must be possible to *incrementally* improve performance and good tools must record the progress of modifications and point to directions for improvement.

We believe that these two major points imply some technical requirements for any Time Warp system:

- The fixed (non-state-saving) overheads must be close to those of existing high performance sequential simulators (say within a factor of two).
- The worst case state-saving overhead of model code cannot be more than 50% of the code execution time. This implies either very good compiler support or hardware assist in the state-saving process. It is also necessary that the mechanism be robust enough to deal with small pieces of state that are very actively modified or

very large pieces of state that are sparsely or infrequently modified.

- Global and shared state must be efficiently and transparently usable.

Taken together these points imply that for the near term the only viable platforms are those that provide shared memory. It is impossible to get the overheads of message passing and sharing state sufficiently low on explicit message passing systems. Also the problems of balance and memory usage and exhaustion are much easier in a shared memory setting. Recent architectures, such as the KSR1 (the Kendall Square Research multiprocessor), provide a virtual shared memory view on a machine that has essentially message passing hardware.

We believe that it is an open question whether all of this can be achieved without special hardware. If hardware assist is needed, it will probably be provided only if it can be shown that Time Warp has general utility outside simulation, for example, as a technique for synchronizing distributed databases or general distributed computations. Many of these points are applicable to the entire realm of distributed computing and suggest that only shared memory systems will survive outside of niche applications.

Acknowledgments

We appreciate the support of Jade's people and environment, particularly discussions of Fujimoto's paper with Darrin West. Our work has been partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

We would like to thank the editors for the opportunity to comment on what is a very important paper. We look forward to much energetic discussion on the issues that have been raised.

References

1. D. Baezner, G. Lomow and B. Unger, 1993. A Parallel Simulation Environment based on TimeWarp, *International Journal in Computer Simulation* (in press).
2. D. Baezner, C. Rohs and H. Jones, 1992. U.S. Army MODSIM on Jade's TimeWarp, in *Proceedings of the 1992 Winter Simulation Conference*, pp. 665–671.
3. E. Leung, J. Cleary, G. Lomow, D. Baezner and B. Unger, 1989. The Effects of Feedback on the Performance of Conservative Algorithms, *SCS Conference on Distributed Simulation, Simulation Series 21:2*, pp. 44–49.
4. X. Li, J. Cleary and B. Unger, 1992. Virtual Time and Virtual Space, *International Journal for Parallel Programming* (in press).
5. B. Unger, J. Cleary, A. Dewar and Z. Xiao, 1990. A Multi-Lingual Optimistic Distributed Simulator, *Transactions of the Society for Computer Simulation 7:2*, 121–152.