ORSA Journal on Computing

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

Commentary—A Survival Guide for Parallel Simulation

Rajive Bagrodia,

To cite this article:
Rajive Bagrodia, (1993) Commentary—A Survival Guide for Parallel Simulation. ORSA Journal on Computing 5(3):234-235. https://doi.org/10.1287/ijoc.5.3.234

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

0899-1499/93/0502-0234 $01.25

COMMENTARY

A Survival Guide for Parallel Simulation

RAJIVE BAGRODIA / *Computer Science Department, UCLA, Los Angeles, CA 90024-1596; Email: rajive@cs.ucla.edu*

The comprehensive article by Fujimoto clearly outlines the path that parallel simulation must follow in the future. To paraphrase a winning slogan from the recent presidential elections—"The tools, stupid." This paper elaborates on Fujimoto's insightful observations on what is required to make parallel simulation more accessible to the discrete event simulation community and presents experiences derived from the author's research to reinforce the conclusions.

1. Brief History

Parallel simulation has historically been interwoven with parallel computation and it is instructive to examine the history of parallel programming, however briefly, lest we be "condemned to repeat it." Until recently, parallel computation was subdivided into many specialized domains. Developing an efficient parallel program involved a large amount of low-level code tuning whose primary purpose was to optimize the use of the architectural characteristics of the target parallel machine. A significant amount of overhead was involved in transforming a correct parallel program into an *efficient* one. Over the last few years, parallel computation research has been focused on devising tools and standardized programming notations to simplify the programming task and to encourage the use of parallel execution by the vast scientific computation community. Interestingly, the approaches that have been suggested are not that different (as is to be expected) from the four silver bullets suggested by Fujimoto for parallel simulation.

In a similar vein, for most of the 1980s, parallel simulation research was essentially partitioned into two main camps—the conservatives and the optimists (or the pessimists and the speculators, depending on one's religion!). Substantial speedups that were reported were obtained only after the arduous work of low-level performance tuning that optimized specific features of the particular synchronization algorithm. I hope that the parallel simulation community will also focus its efforts on the design of tools and techniques that encourage the larger simulation community to use the parallel simulation technology.

2. Silver Bullets

Of the four silver bullets identified by Fujimoto, automatic parallelization, perhaps, offers the least chance for success. Although automatic parallelization of sequential code was successful for vector processors, it has had a limited impact on concurrentization for scalable distributed memory parallel architectures (e.g., the Intel Paragon). The automatic parallelization techniques also cannot uncover the new parallel solutions that typically yield the greatest payoff. Lastly, efficient execution of simulations on scalable parallel architectures will require sophisticated analysis of causality and locality in the model. This appears to be a hard problem in the context of a simulation model that has been developed for sequential execution.

Although library-based solutions are attractive, the design and maintenance of efficient libraries is itself an onerous task. Unless suitable tools to help the library designers are available, it is unlikely that many such libraries will be designed. Also, many of the optimizations in parallel simulation are dynamic and rely on non-local information. It is unclear how hard-coded libraries will be able to exploit this effectively. The most promising approach to the problem seems to lie in the design of parallel simulation languages that can present an appropriately simplified abstract machine to simulationlists.

3. Parallel Simulation Languages

With respect to the future form of simulation languages, it is my belief that the evolutionary approaches offer the most promise. Perhaps the best approach towards facilitating the transition for simulationists to parallel simulation is in adopting the process interaction approach which appears to be a natural paradigm for parallel simulations. A viable approach is to design a parallel simulation kernel that can be added to existing process (or object-based) parallel languages. This kernel provides constructs to schedule, and possibly unschedule, events using the message passing or remote procedure call (RPC) constructs of the host language. Most existing imperative parallel simulation languages (PSLs) adopt this approach.

The most important characteristic of a successful PSL will be its adaptability. Adaptability is used to refer to a language that is not, a priori, committed to a specific synchronization algorithm or to a specific implementation of a synchronization algorithm—for example, one that only supports a conservative algorithm that uses aggressive null-message based synchronization. The design of a language that will allow simulationists to couple a variety of synchronization algorithms with a simulation model, and possibly allow different submodels to select different synchronization algorithms in a *dynamic* manner, is an important objective for the parallel simulation community. The flexibility in selecting synchronization algorithms is important for two reasons. First, the field has not yet amassed a sufficiently rich intuition, empirical database, or prediction techniques to estimate whether one algorithm is likely to be more efficient than others in the simulation of a given model. Second, the optimistic and conservative techniques are often complementary in that models that perform badly

using one technique may sometimes perform well using the other technique.

The realization of this vision requires a concentrated research effort in the following primary directions.

- The design of parallel simulation languages that allow implementations using sequential, conservative, and optimistic algorithms. The ability to couple a sequential algorithm is particularly important to allow consistent measurements of realized speedups.
- Language constructs that allow overheads of the synchronization algorithms (e.g., rollbacks, state saving, or null message transmission) to be optimized using application semantics.
- Algorithmic techniques to support model execution that combine both conservative and optimistic techniques. Techniques to integrate these methods have been suggested, but much more work remains to be done.
- Standard interfaces to parallel libraries and performance profiling tools.

4. A Case Study

The preceding objectives are an integral component of the ongoing research at UCLA in the design of the Maisie simulation language.[2] With few modifications, a Maisie program may be executed using a sequential simulation algorithm, a parallel conservative algorithm, or a parallel optimistic algorithm. The Maisie design also allows the runtime system to transparently reduce recomputation and state saving overheads for optimistic simulations, as well as synchronization overheads for conservative implementations. Maisie currently supports the optimistic space-time simulation algorithm,[1, 3] and a variety of conservative algorithms.[6] It is also the first language to support semantic optimizations, that is, the use of application semantics to reduce the overhead of both conservative and optimistic parallel simulations. It is clear that eventually the degree and ease with which a PSL allows a simulationist (or a compiler) to monitor and control the overheads of parallel execution with different synchronization algorithms will be a major determinant in the use of this technology by the larger simulation community.

A preliminary study on the performance of the optimistic and conservative algorithms running under Maisie has yielded insights that further support the case for adaptable PSLs with semantic optimizations. A priority preemptible server performs poorly under conservative methods due to its poor lookahead abilities.[4] However, when using the semantic optimizations provided by Maisie, the conservative implementation was found to yield a speedup that was comparable to that yielded by optimistic implementations for a queuing network of 32 nodes, each node having up to 10 preemptible servers.[5] A second benchmark that simulated a network of FIFO servers yielded *superlinear* speedup: a single node conservative implementation that used application semantics to reduce synchronization overheads was *faster* than the implementation using the standard splay tree (a data structure for handling simulation event lists) based sequential algorithm which did not use the corresponding optimization. The corresponding optimization is now being incorporated even in sequential Maisie models. Finally, using application semantics to vary the frequency of checkpointing (i.e., making a copy of the state of a process) was found to increase the speedups by almost a factor of two for some benchmarks. The point is *not* that tuning improves performance; rather, many parameters that significantly affect performance can be monitored and controlled through structured interfaces at the language level.

5. Conclusion

Fujimoto correctly concludes that parallel simulation will thrive only if the parallel simulation community makes the transition easier for the discrete event simulation community. It is unlikely that we will see a simulation language that automatically executes a sequential model efficiently on a parallel architecture using an appropriate synchronization algorithm. The challenge then is to design environments that permit a simulationist to migrate to parallel simulation in an iterative manner.

References

1. R.L. Bagrodia, K.M. Chandy, and W-T. Liao, 1991. A Unifying Framework for Distributed Simulation, *ACM Transactions on Modeling and Computer Simulation* 1:4, 348–385.
2. R.L. Bagrodia and W-T. Liao, 1992. A Language for Iterative Design of Efficient Simulations, Technical Report UCLA-CSD-920044, Computer Science Department, UCLA, Los Angeles, CA.
3. K.M. Chandy and R. Sherman, 1989. Space-Time and Simulation, in *Proceedings of the 1989 Distributed Simulation Conference*, Miami, pp. 53–57.
4. R.M. Fujimoto, 1988. Lookahead in Parallel Discrete Event Simulation, in *Proceedings of the 1988 International Conference on Parallel Processing*, vol. III, pp. 34–41.
5. V. Jha and R. Bagrodia, 1992. Transparent Implementation of Conservative Algorithms in Parallel Simulation Languages, Technical Report UCLA-CSD-930007, Computer Science Department, UCLA, Los Angeles, CA.
6. J. Misra, 1986. Distributed Discrete-Event Simulation, *ACM Computing Surveys 18:1*, 39–65.