Reducing Synchronization Overhead in Parallel Simulation

Ulana Legedza*
Parallel Software Group
ulana@lcs.mit.edu

1 Introduction

Software simulators are used in parallel computing for architecture design and application development. They are necessary because hardware prototypes of parallel computer architectures are time-consuming to build and difficult to modify. Parallelism is needed in multicomputer simulators in order to provide fast, detailed simulation of large target systems. However, despite its benefits, parallel simulation introduces some new costs. The foremost of these is the overhead cost of synchronization, which often dominates simulator execution time. Synchronization overhead is a particular problem in multicomputer simulators because accurate simulation of the fast networks found in parallel systems requires frequent synchronization.

We present **local barriers** and **predictive barrier scheduling**, two techniques for reducing synchronization overhead in parallel simulation of message-passing multicomputers. Local barriers use nearest-neighbor synchronization to reduce waiting time at synchronization points. Predictive barrier scheduling, a novel technique which schedules synchronizations using both compile-time and runtime analysis, reduces the frequency of synchronization operations. These techniques were evaluated by comparing their performance to that of periodic global synchronization. Experiments show that local barriers improve performance by up to 24% for communication-bound applications, while predictive barrier scheduling improves performance by up to 65% for applications with long local computation phases. This work was done in the context of **Parallel Proteus**, a new parallel simulator of message-passing multicomputers.

2 Synchronization Overhead

In conservative parallel simulation of multicomputers, each processor of the host parallel machine simulates one or more processors of the target system. Synchronization of the host processors is necessary in order to correctly simulate communication occurring in the target system. The simplest way to perform the necessary synchronization is with periodic global barriers. For simulations of multicomputers with low-latency communication, these barriers must be performed very frequently – sometimes as often as every 2 cycles of simulated time. This high frequency, in addition to long waiting times at synchronization points, often makes synchronization overhead the dominant cost in parallel simulation. In Parallel Proteus, synchronization overhead accounts for 70% to 90% of total simulation runtime and therefore limits the speedup that can be achieved.

3 Local Barriers

In the local barrier approach, nearest-neighbor synchronization is used to reduce waiting time at synchronization points. The nearest neighbors of each host processor are determined by the topology of the network being simulated and the layout of the target nodes on the host machine. At each synchronization point, each host processor only waits for its nearest neighbors before continuing with the simulation.

Loosely synchronizing the processors in this way will shorten barrier waiting times in the situation where the simulation work is not equally distributed among the host processors. Only several host processors will have to wait for each slow (heavily loaded) processor to reach a synchronization point.

4 Predictive Barrier Scheduling

Synchronization of host processors is not necessary during periods when no communication is taking place in the target system. Predictive barrier scheduling combines knowledge of this fact together with compile-time and runtime analysis to reduce the frequency of synchronization operations. At compile-time, applications to be run on the target machine are analyzed (using dataflow techniques) in order to determine their future communication behavior. More information about communication behavior is produced at runtime, when application threads are allowed to run ahead of simulated time. Using this information, the simulator tries to schedule synchronizations only during communication phases.

Predictive barrier scheduling seems especially promising for applications that have long phases of local computation, alternating with communication phases. The synchronization during the communication phases would remain frequent, while the synchronization during computation phases would be eliminated.

*This work was supervised by Professor Bill Weihl and was supported in part by the Advanced Research Projects Agency under Contract N00014-94-1-0985, by grants from IBM and AT&T, and by an equipment grant from DEC. Ulana Legedza is also supported by a National Science Foundation Graduate Fellowship.

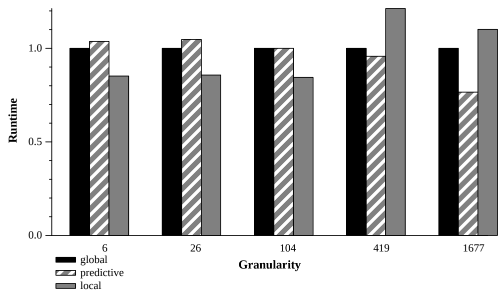

Figure 1: Normalized simulator runtime for *SOR*, 625 target processors, varying grid size.

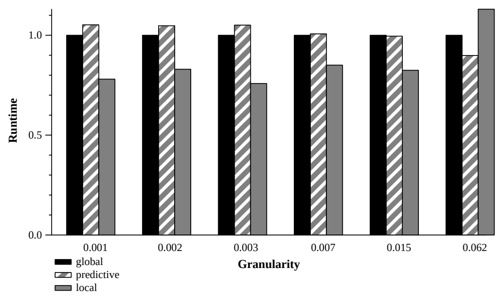

Figure 2: Normalized simulator runtime for *radix*.

5 Experiments

The local barrier approach and predictive barrier scheduling were incorporated into Parallel Proteus, and their performance was compared with that of the baseline periodic global barrier approach. Experiments were run on a 32-node CM5 partition. The parallel applications used were SOR (successive over-relaxation) and radix sort, running on target machines with 64 to 1024 processors.

The local barrier approach adds some overhead to the execution of each synchronization. However, it often produces an overall performance improvement because it significantly shortens the time spent waiting at each synchronization point. The largest performance improvements (up to 24% in our experiments) are achieved in simulations of communication-bound applications, where waiting times under periodic global barriers are high. In simulations of computation-bound applications, local barriers slow down simulations by up to 75%.

Under predictive barrier scheduling, simulations perform 10% to 95% fewer synchronizations than under periodic global barriers, achieving overall performance improvements in the range of -5% to +65%. The largest performance improvements occur in computation-bound applications, complementing the behavior of the local barrier approach.

To illustrate this complementary behavior, we graph normalized simulator runtime versus *granularity*. Granularity is a measure of the length of local phases of computation relative to the length of communication phases. Applications with large granularity have long local computation phases. Those with small granularity are dominated by communication and short computation phases. The graphs in figures 1, 2, and 3 show that predictive barrier scheduling improves performance for applications with large granularity, while local barriers improve performance for applications with small granularity.

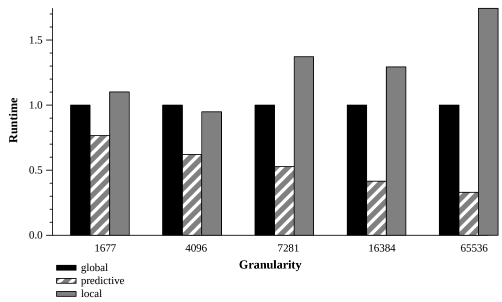

Figure 3: Normalized simulator runtime for *SOR*, 1024 x 1024 grid size, varying number of target processors.

This complementary relationship between predictive barrier scheduling and the local barrier approach suggests that one might combine the two techniques in the hope of achieving more consistent performance improvement. One way to do this is to use each technique in the situation where it does better: the local barrier approach during communication phases and predictive barrier scheduling during computation phases. In this way, unnecessary barriers in the computation phases will be eliminated by predictive barrier scheduling. For its part, the local barrier approach will shorten the waiting times in communication phases.

6 Conclusion

Parallelism is necessary for fast, detailed simulation of large multicomputers. Synchronization overhead, however, often severely limits the performance of conservative parallel simulators. This occurs because low-latency communication in simulated networks requires frequent synchronization of simulator processes. Our work demonstrates that local barriers and predictive barrier scheduling are effective techniques for reducing synchronization overhead in simulations of message-passing parallel computers. Combining the two techniques is likely to improve performance even more.

References

[Leg95] Ulana Legedza. Reducing synchronization overhead in parallel simulation. Master's thesis, MIT, June 1995.