Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

TOWARDS "ON THE FLY" PERFORMANCE MODELS FOR CONSERVATIVE ASYNCHRONOUS PROTOCOLS

Mary L. Bailey
Shane Walker

Department of Computer Science
The University of Arizona
Tucson, Arizona 85721

ABSTRACT

In this paper we present a simple model of communication costs for conservative asynchronous simulations using a bus structure. Not only is this a useful communication structure in practice, but it places some constraints on communication which make it more amenable for "on the fly" characterizations. Overall, the model shows promise when evaluated using test cases not involved in the model building. Most of the characterizations can be computed using simple formulas parameterized with variables obtained from the input program. A few characterizations are more complex, but these can be estimated by applying a sequence of simple functions.

1 INTRODUCTION

There are a large number of application areas where simulations are of prime importance in understanding aspects of physical systems. In many of these, the time taken by sequential simulation is far too great. In these cases parallel or distributed simulation may provide a viable solution – accurate simulation in a reasonable time frame. Thus there has been much activity in the parallel simulation community to provide general protocols for accurate and efficient implementations.

There are two major types of simulation protocols for parallel simulation, conservative and optimistic, that differ in the way simulation time advances. Conservative protocols only permit execution of correct computation, that is an event is processed only if it can be guaranteed that it is a correct next event. Parallel conservative protocols range from synchronous protocols with a single global clock and all logical processes advancing in lock-step, to asynchronous protocols with many local clocks and logical processes advancing at different rates (Bryant 1977, Chandy and Misra 1979). Optimistic protocols allow speculative event processing, and provide some mechanism for undoing computation if the speculation was erroneous (Jefferson 1985). There are applications where each protocol outperforms the other (Lipton and Mizell 1990), and it is often not clear which protocol is preferred until both have been tried.

Over the past few years there has been research predicting the execution time of the optimistic strategy (Felderman and Kleinrock 1991, Gupta, Akyildiz, and Fujimoto 1991, Lin and Lazowska 1990), and research in using formal models to compare the two strategies (Bailey and Lin 1993, Lipton and Mizell 1990). We are interested in predicting the execution time for conservative asynchronous protocols using simple models, models that can be computed quickly. To date, the only models for these protocols assume self-initiating systems, instead of the more common message-initiating systems (Nicol 1991). Because we want to use simple models, we have restricted our focus in this paper to a single application domain, distributed memory parallel programs, and a single communication structure, busses. Busses are a common structure in processor designs and are natural structures whenever a single channel connects more than two logical processes. Moreover, restricting communication to busses makes the overhead characterizations more straightforward than for general point-to-point communication because the effects of the communication topology are less varied.

In this paper we present our formulaic characterizations of communication costs using the bus communication primitive and the conservative asynchronous protocol. A crucial aspect of these characterizations is that they are functions of variables that can typically be obtained from the input programs. We then evaluate the accuracy of the characterizations using two test programs. In all cases the model predictions are within 15% of the actual costs, and the vast majority are within 10%.

2 THE EMPIRICAL SYSTEM

The CPoker simulator, our experimental system, is implemented on a shared memory multiprocessor, the Sequent Symmetry, and uses a conservative asynchronous protocol with deadlock avoidance. The implementation is optimized for a shared memory multiprocessor, so the resulting characterizations are most relevant to shared memory implementations, although we expect the general form of many of the characterizations to also hold for distributed memory multiprocessors. A CPoker program consists of a set of logical processes (LPs) that communicate via specified communication channels. Currently CPoker supports two types of point-to-point communication primitives. The first type, Read or Write, supports sending or receiving messages on a specific communication channel. The second, MultiRead, supports receiving messages from one of a set of communication channels (Bailey and Pagels 1991). This second type of communication is critical for many common simulation programs, such as queuing networks, where messages need to be processed in time stamp order, independent of sender.

In addition to the point-to-point communication primitives, CPoker supports analogous primitives for bus communication structures. A bus connects three or more LPs, and each LP can read from or write to the bus at various times. Multiple simultaneous readers are permitted; each will receive the same message provided that they are all ready to read when the data is written to the bus. When an LP wishes to perform a write, it must gain control of the bus. Once control is granted, the writer has exclusive write privileges during its transaction. After the reader(s) have finished reading the data, the writer is signaled and the bus is released. If multiple writers want control of the bus, the writer with the earliest request is given priority. If there are multiple writers with the earliest request time, the one with the smallest bus id will be selected. BusRead, BusWrite, and MultiBusRead are the bus communications analogous to Read, Write, and MultiRead.

3 MODEL CHARACTERIZATION

Each of the three bus communication primitives, BusRead, BusWrite, and MultiBusRead are characterized separately, although there are many similarities between the three. First, the general form of the code is quite similar in all three, with non loop and wait-loop sections in each. Second, all primitives use context switches and null messages in the routines. Since the cost of sending a null message or incurring a context switch is independent of the communication primitive, these are characterized separately, but the number of each of these does differ in the three primitives.

In all of the characterizations, we use factorial designs to determine the significant variables (or input parameters) in the communication costs. Each variable is measured using two values, and Yate's algorithm is used to determine the critical variables. Box, Hunter and Hunter (1978) contains more detailed information on factorial designs. In our experiments, we consider up to 8 different variables in the designs: the number of readers per bus (Rdrs), the number of logical processes (LPs), the number of physical processors (Procs), the bus length (BL), the lookahead (LA), the number of busses connected to each LP (BC), whether Read or MultiRead is used (RdTyp), and whether the workload is balanced among the LPs (Bal).

Due to space limitations, we cannot discuss the individual characterizations for each communication structure. However, we will show most of the formulas for BusRead to provide some indication of the structure and style of the characterizations.

There are three parts to BusRead. First, the LP declares that it is performing the read. Second, the LP waits until it knows which LP is the writer so that it takes the correct data at the correct simulation time. Here a wait-loop is executed until the determination can take place; at each iteration a context switch occurs and null message may be sent. Third, one of the readers signals the writer so that the writer can continue.

The first and third parts constitute the non wait-loop costs (RdNonLp). The second part, the wait loop cost, is further divided to make the analysis more straightforward. First, it is modeled as a linear function of the number of iterations, excluding the cost of context switches and null messages (RdWaitLp). Next, the number of iterations is computed (RdIter). Third, the total cost of context switches in BusRd is computed (RdCSCost). Finally, the cost of null messages sent in BusRd is computed (RdNMCost).

Figure 1 shows formulas for all BusRead costs, except for the number of wait-loop iterations. The number of iterations is the most difficult quantity to estimate, since this is where the non-deterministic nature of a parallel program impacts the code. Slight timing variances can cause an extra iteration of the loop, and the total number of iterations is quite small (typically between 1 and 5). If the number of busses connected to each LP is more than one, then a constant of 1.1 is sufficient to model RdIter except when there is a small number of LPs per processor (fewer than 10).

$$RdNonLp = \frac{224 + 7.5BL + 209(Rdrs - 1)}{Rdrs} + 15BC. \quad (1)$$

$$RdWaitLp = 140 + 7.5BL + \left(\frac{-50 + 12.5BL}{BL - 1}\right)(BL - Rdrs) + (5 + 22.5BL)RdIter. \quad (2)$$

$$RdCSCost = 68(RdIter + 1). \quad (3)$$

$$RdNMCost = (20 + 10BL + 100/BC + 5(\log(BL/4) + 1))NoRdNMs. \quad (4)$$

$$NoRdNMs = \begin{cases} 3 & \text{if } Rdr = BL - 1 \\ 2 & \text{otherwise.} \end{cases} \quad (5)$$

Figure 1: Formulas for BusRead Costs

In this case, there is too little computation on each processor and estimating the exact number of iterations is too complex. In reality, the workload is too small to take advantage of the amount of parallel processing, a situation that is not expected to occur in practice. If there is one bus connected to each LP, the number of iterations is a function of Procs, Bal, LPs, Rdrs, BL, and LA. We begin with a baseline formula for RdIter, which is a function of Rdrs, BL, and LA, and assume that Procs = 5, LPs = 256, and Bal = balanced. We then independently apply correction formulas for each of Procs, LPs, and Bal if their values differ from the baseline assumptions. The baseline formula together with the corrections gives an estimate for RdIter. Thus, RdIter is not a simple formula, but is a sequence of simple computations maintaining the "on the fly" characterization.

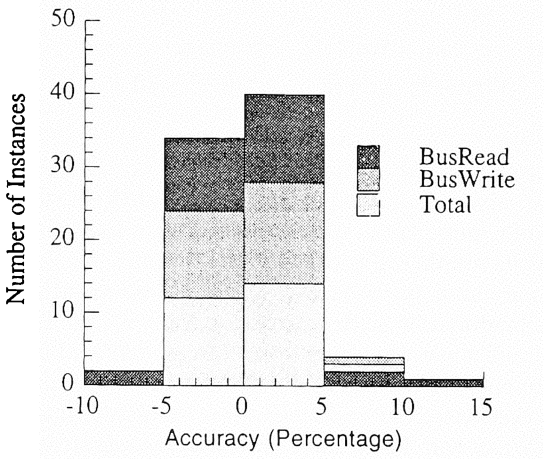

Figure 2: Accuracy of MixedBus Costs

4 MODEL EVALUATION

The model was evaluated using several test programs not involved in the characterizations. We will present the results of two of these here. The first test program, MixedBus, uses a number of different sized busses simultaneously. The second, MixedRead, uses a combination of BusRead and MultiBusRead.

MixedBus uses three different sized busses with each LP connected to exactly one bus (so no MultiBusReads are used). More specifically, in an 8x8 LP grid, there are 4 busses of length 8, 4 of length 6, and 2 of length 4. A single grid uses 64 LPs, while 4 grids require 256 LPs. Twenty-seven different instances of MixedBus were run, varying different parameters. Those parameters varied include the number of processors (5 and 11), the number of readers on each bus (1, half, all but one), the lookahead (3 values), and the number of LPs (64 and 256). Figure 2 shows the accuracy of the model for this experiment. Three quantities were measured in the evaluations, BusRead, BusWrite, and Total. For BusRead and BusWrite, we measured the average cost of a single communication, while for Total we used the cost of one BusRead plus one BusWrite. In each case, we took the ratio of the difference in the model prediction and actual cost to the actual cost. Thus a positive percentage means that we overestimated the cost, and a negative percentage implies that we underestimated the cost. Overall, the predictions are quite good. All predictions, except for one BusRead, are within 10% of the actual measurements. Moreover, for over 90% of the instances, the model predictions are within 5% of the measured results. The one instance where the accuracy of BusRead is poor (difference greater than 10%) spends very little time in BusRead, so that when it is combined with its corresponding BusWrite, the total is within 10% accuracy.

MixedRead uses a combination of MultiRead, BusRead, and BusWrite. Each LP is connected to two busses, with half of the readers on each bus performing BusReads and the other half performing Multi-

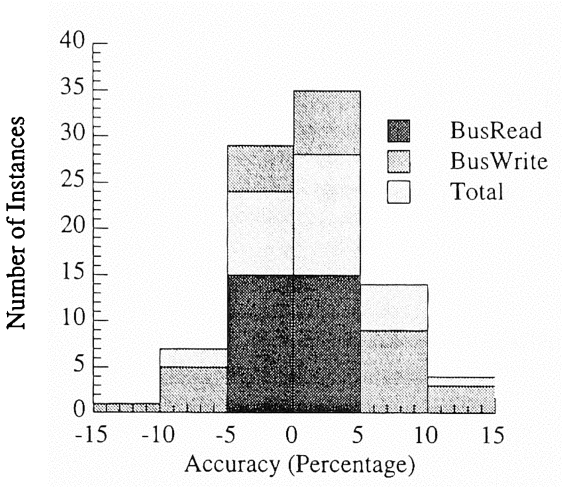

Figure 3: Accuracy of MixedRead Costs

BusReads. All 30 instances use 256 LPs; parameters that vary include the bus length (4 and 8), the number of readers (2, 4, and 6), the lookahead (3 values), and the number of processors (5, 7, 9, 11). As before we measured the accuracy of Reads (which includes BusRead and MultiBusRead), BusWrites, and Total. Figure 3 shows the accuracy of the model for this experiment. Here, Reads are quite accurate (within 5%) but there are a few instances where the predictions for BusWrite and Total differ from actual measurements by up to 15%. No one single parameter is responsible for this, and the predictions are sometimes too high and other times too low. Thus it is not clear that a simple change to the BusWrite model will result in greater accuracy.

5 CONCLUSIONS

We have presented a model for predicting the communication costs for busses that is fast, so that "on the fly" estimates can be made. The model has been evaluated on two test programs with good results. In all 57 instances, the predicted communication costs are within 15% of actual costs, with the vast majority of instances within 10% of actual costs.

ACKNOWLEDGMENTS

This work is funded in part by National Science Foundation Grant CCR-9212018.

REFERENCES

Bailey, M.L. and Lin, Y.B. 1993. Synchronization Strategies for Parallel Logic Simulation. *International Journal in Computer Simulation* **3(3)**:211-230.

Bailey, M.L. and M.A. Pagels. 1991. Measuring the Overhead in Conservative Parallel Simulations of Multicomputer Programs. In *Proceedings of the Winter Simulation Conference*, 627-636.

Bailey, M.L. and Walker, S. 1994. Towards "On the Fly" Performance Models for Conservative Asynchronous Protocols. Technical Report 94-15, Computer Science Department, University of Arizona.

Box, G.E., Hunter, W.G., and Hunter, J.S. 1978. *Statistics for Experimenters.* John Wiley and Sons.

Bryant, R.E. 1977. Simulation of packet communication architecture computer systems. Technical Report MIT-LCS-TR-188, MIT.

Chandy, K.M. and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. on Softw. Eng.* **SE-5(5)**: 440-452.

Felderman, R.E. and L. Kleinrock. 1991. Two processor time warp analysis: Some results on a unifying approach. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 3-10.

Gupta, A., Akyildiz, I.F., and Fujimoto, R.M. 1991. Performance Analysis of Time Warp with Multiple Homogeneous Processors. *IEEE Trans. on Softw. Eng.* **17(10)**:1013-1027.

Jefferson, D. 1985. Virtual Time. *ACM Trans. on Program. Lang. Syst.* **7(3)**:404-425.

Lin, Y.-B. and E.D. Lazowska. 1990. Optimality considerations for "time-warp" parallel simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 29-34.

Lipton, R.J. and D.W. Mizell. 1990. Time warp vs. chandy-misra: A worst-case comparison. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 137-143.

Nicol, D.M. 1991. Performance Bounds on Self-Initiating Discrete-Event Simulations. *ACM Trans. on Model. Comp. Simul.* **1(1)**:224-50.

AUTHOR BIOGRAPHIES

MARY L. BAILEY is an assistant professor in the Department of Computer Science at the University of Arizona. Her research interests include parallel and distributed simulation, computer-aided design for VLSI, special-purpose architectures, and parallel computation.

SHANE WALKER is finishing up a master's degree in Computer Science at the University of Arizona. His current research focuses on parallel and distributed simulation.