EXECUTION-DRIVEN SIMULATION OF SHARED-MEMORY MULTIPROCESSORS

S. Dwarkadas, J. R. Jump, R. Mukherjee, and J. B. Sinclair
Department of Electrical and Computer Engineering
Rice University
Houston, Texas 77251-1892

ABSTRACT

This paper describes an efficient execution-driven technique for the simulation of shared-memory multiprocessors driven by real programs. Our simulator offers substantial advantages in terms of reduced time and space overheads when compared to instruction-driven or trace-driven simulation techniques, without significant loss of accuracy. The technique produces correctly interleaved address traces at run time without disk access overhead, allowing accurate simulation of a variety of architectural alternatives for programs. We present the results of several validation experiments used to quantify the accuracy and efficiency of the technique for three shared-memory multiprocessors and several parallel algorithms. These experiments show that prediction errors of less than 5% and overheads 5 to 25 times lower than those incurred by cycle-level simulation can be achieved.

INTRODUCTION

In this paper, we present an efficient technique for the simulation of shared-memory multiprocessors. We also present the results of several validation experiments performed to evaluate the effectiveness of the approach. The technique is an extension of execution-driven simulation (Covington et al. 1988; Covington et al. 1991), which was originally designed to simulate distributed memory systems with no cache memory. The extensions described here add the capability to simulate cache-based and/or shared-memory systems that require an analysis of the program address trace for accurate performance prediction. We have implemented the new technique as part of the Rice Parallel Processing Testbed (RPPT) (Covington et al. 1988), a tool for evaluating the performance of parallel computing systems.

TRAPEDS (Stunkel and Fuchs 1989) is another extension of execution-driven simulation that can simulate systems with cache memory, but does not deal with shared-memory systems. TRAPEDS also does not model communication accurately, since it does not evaluate the effects of contention. MPtrace (Eggers et al. 1990) uses the execution-driven approach to generate a template of trace information that is used at a later time to generate an address trace. It has an overhead of 2 to 3 times normal execution time (excluding trace storage) to generate this trace template. However, the post-processing phase needed to construct the actual address trace is extremely slow, generating about 3000 addresses per second from the saved template. Link time code modification has been used to generate long traces on sequential RISC machines (Borg et al. 1990). ATUM (Agarwal et al. 1986) captures address traces using microcode. While this technique is efficient, long traces must be stored on disk and a processor with user-programmable microcode is needed to run the system.

Tango (Davis et al. 1990) is a multiprocessor simulation and tracing package based on the execution-driven approach that concentrates on program data accesses. Tango originally used UNIX processes to simulate parallelism, resulting in high context switching overhead. A recent version of Tango uses light weight processes as in our simulator. PROTEUS (Brewer et al. 1991), another execution-driven simulation system, also concentrates only on shared data addresses and instruments the high-level language to generate the shared data addresses, while profiling at assembly level to extract timing information. Neither Tango nor Proteus have provided extensive validation results for shared-memory machines.

EXECUTION-DRIVEN SIMULATION OF SHARED MEMORY ARCHITECTURES

In execution-driven simulation, the execution of a real program drives the simulation of an architecture, avoiding the emulation of instructions necessary in more conventional instruction-driven approaches. Execution-driven simulation modifies the program using a profiler, which identifies each basic block (i.e., maximal sequence of instructions, which when once entered, has every instruction executed exactly once) of the program and determines the time required to execute all the instructions in the block. For non-cache and non-shared-memory systems the simulator need only increment simulation time (by executing the instructions inserted by the profiler), and execute the basic block.

To simulate parallel computers, we must also simulate the movement of data between the modules of the simulated architecture whenever one module communicates with another. We call these points process interaction points. In order to keep the actions of the different architectural modules interleaved properly according to program dependencies, it is necessary for each processor to delay by an amount equal to the accumulated cycle count

This research was funded in part by the Center for Research on Parallel Computation through NSF Cooperative Agreement No. CCR-9120008.

of the processor each time it reaches a process interaction point.

Shared-memory and/or cache-based systems result in every memory access being a potential process interaction point. This means that execution-driven simulation of these systems will incur more overhead than for message-based systems, since the simulator must trace addresses and simulate memory accesses and cache misses to accurately model program behavior. The overhead is still less than for cycle-level simulation, which incurs the extra overhead of a detailed emulation of each instruction in addition to that for memory simulation.

We have extended the execution-driven simulator implemented in the RPPT to deal with both shared memory and caches. The extensions consist of two main additions. First, we modified the profiler to generate memory addresses. For those addresses that are not known until run time, the profiler inserts code into the program to generate them dynamically during the simulation. A detailed description of some of these techniques and problems encountered in implementing them can be found in (Dwarkadas et al. 1989). The program may be modified to generate all instruction and data addresses, or to generate only instruction or only data addresses. Second, we developed simulation models for shared memory and cache memory that accounted for the delay due to address conflicts and cache coherency protocols.

Figure 1 illustrates the mechanism by which traces are generated. The profiler must instrument the program to generate address traces for cache/shared-memory simulation. It does this by constructing an address trace template for each basic block. Since instruction addresses are known to the profiler, it can insert them directly into the corresponding template. When an address must be determined dynamically at run time, the profiler inserts code in the basic block at the point of the access to extract the address at run time and places a marker in the template that indicates that a reference will be inserted later.

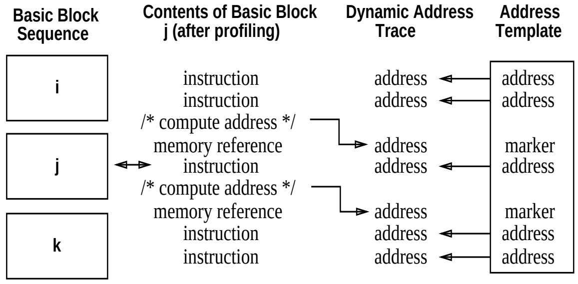

Figure 1 Address Trace Generation

In the case of systems with private caches and no shared data (e.g., a message-based system with private caches), cache accesses do not constitute potential process interaction points. Hence, the entire basic block address trace may be processed together at the end of the basic block. For shared-memory systems, every access to a shared address must be treated as a potential process interaction point. The cache simulation routine must be called for each memory access (this may be restricted to only shared data accesses when there is local memory for private data) generated to determine when misses occur and to simulate the coherence mechanism. This will force the simulator to delay the process reading a shared location until all processes that could alter the data at that address have done so. This guarantees that the addresses generated by different processes will be interleaved correctly in simulation time. Maintaining correct interleaving of multiprocessor address traces is very difficult when the traces are saved and processed later.

EVALUATION

We have conducted extensive experiments to validate the execution-driven approach to simulation. These experiments were designed to measure the accuracy and overhead of the technique. The accuracy is evaluated by simulating the execution of several parallel programs on parallel computers and comparing the performance predictions with timing measurements obtained from executing the same programs on the real systems. The overhead is computed as the ratio of the time required to simulate the execution of a program to the time to execute it as a multithreaded program on a uniprocessor. This comparison provides a measure of the extra work required to simulate the movement of data between the modules of a parallel system.

We presented validation results for the basic execution-driven technique applied to distributed-memory, message-based architectures in previous papers (see, for example, Covington et al. 1991). In this section, we extend those results to shared-memory systems. To this end, we present the results of validation experiments on two commercial shared-memory multiprocessors, the Sequent Symmetry* (Lovett and Thakkar 1988) and the BBN Butterfly GP1000 (BBN Laboratories 1985). We also present comparisons of the execution-driven testbed with a cycle-level simulator of a hypothetical single-bus multiprocessor based on the SPARC architecture (Ross Technology Inc. 1990).

The Sequent Symmetry

The Sequent Symmetry is a single-bus, shared-memory multiprocessor with a two-way set-associative cache private to each processor. The coherence protocol is write back with invalidation of other copies when the data is shared. A cache that supplies the dirty copy of a cache line also invalidates its own copy regardless of whether the request was for a read or a write access. Coherence is maintained by using the bus' snooping ability. The bus uses split transactions.

The validation experiments were driven by parallel versions of a radix 2 Fast Fourier Transform (FFT) program on 8192 points, a Gaussian Elimination (GAUSS) program on a 96x96 matrix, and a Subgraph Isomorphism (ISO) program that found 6840 subgraphs isomorphic to a particular 3-node subgraph within a 20-node graph. These programs were chosen as representative of a range of numeric and non-numeric programs.

Figure 2 demonstrates the accuracy of the Sequent simulations, which are within 5% in most cases. More extensive results can be found in (Dwarkadas et al 1992). For these experiments, error is defined as the percentage difference between the prediction of execution time and the

*Use of the Sequent Symmetry S81 was provided by the Department of Computer Science at Rice University under NSF CISE Infrastructure (II) Grant CDA-8619393.

measured execution time. The errors remain fairly constant as the number of processors increases, indicating that contention on the Sequent bus is modeled accurately. The inaccuracies and variations that do exist can be partly attributed to errors in real time measurement and the fact that the time taken by certain instructions (especially floating point instructions) is data-dependent and cannot be predicted exactly prior to execution. Also, FFT and ISO make a number of calls to process scheduling routines that were not accurately modeled in the simulation.

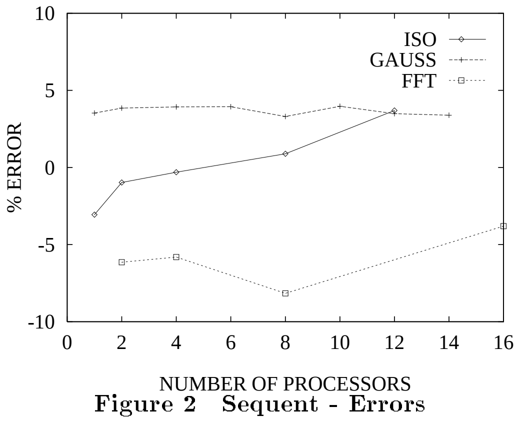

Figure 2 Sequent - Errors

Algorithm	Average Overhead
Subgraph Isomorphism (ISO)	639x
Fast Fourier Transform (FFT)	505x
Gaussian Elimination (GAUSS)	598x

Table 1 Sequent - Overheads

Table 1 shows the overheads incurred. Most of this overhead is due to the need to simulate the movement of data between processors and memory. This, in turn, depends on the characteristics of the program that determine the amount of data moved. For example, the increased overhead of ISO relative to the other two programs is due in part to the absence of floating point code resulting in an increase in the ratio of communication to computation.

The BBN Butterfly

The BBN Butterfly is a shared-memory multiprocessor that is quite different from the Sequent Symmetry. The Butterfly is a non-uniform-memory-access (NUMA) architecture with memory physically distributed among the processors. Therefore, the distribution of data among the different modules is a major factor in the performance of the system. The Butterfly does not have cache memory. Finally, the Butterfly uses a multistage interconnection network to connect the processors, while the Sequent uses a single shared bus to connect all processor and memory modules.

We used the same set of benchmarks for the evaluation of the Butterfly architecture as for the Sequent Symmetry. Due to problems in porting the run time parallel environment, we simulated the equivalent sequential programs run on a single processor but with all global data placed on remote nodes to force the programs to make remote accesses across the network. This allowed us to evaluate the accuracy of the Butterfly architecture model. All programs were run with several data sets of varying sizes. The FFT data ranged from 1024 to 32,768 points, the GAUSS program was applied to matrices ranging from 50x50 to 175x175, and the ISO program was used to find all subgraphs of 15- and 18-node graphs that were isomorphic to given 3- and 4-node graphs.

Figure 3 shows that the accuracy of the Butterfly simulations vary from +10% for GAUSS to a little less than -10% for ISO. The differences in error between the different programs is partly due to the different amounts of network traffic they generate. The percentage of data accesses that were remote was approximately 5% for ISO, 50% for FFT and 62% for GAUSS. Another source of error is that the simulation model for the interconnection network was based on a number of simplifying assumptions to reduce the simulation time to a manageable level.

Average simulation overheads for the three programs are given in Table 2. Once again, the differences are largely due to the different amounts of network traffic generated by the three programs. Since most of the simulation overhead is due to the time required to simulate the movement of data through the network, the increased network traffic of GAUSS and FFT resulted in increased simulation overhead compared to ISO.

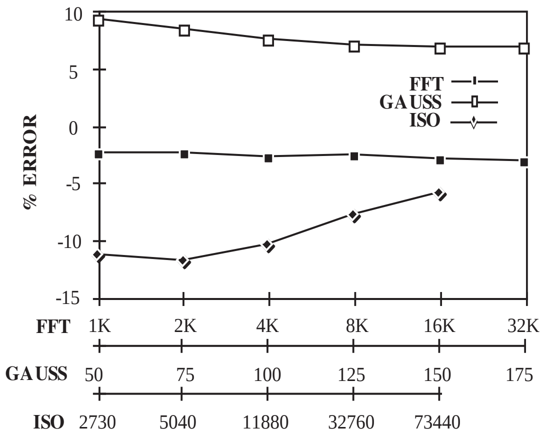

Figure 3 Butterfly - Errors

Algorithm	Average Overhead
Subgraph Isomorphism (ISO)	150x
Fast Fourier Transform (FFT)	780x
Gaussian Elimination (GAUSS)	1200x

Table 2 Butterfly - Overheads

Single Bus SPARC Architecture

This section evaluates execution-driven simulation relative to cycle-level simulation. This is done by simulating a hypothetical single-bus SPARC-based multiprocessor with both types of simulators and comparing their execution time predictions and overheads. The cycle-level simulator used is derived from MPSAS (Federwisch and Ball 1990; Greenwood 1992), a multiprocessor simulator developed by Sun Microsystems.

We simulated a system consisting of 20 MHz SPARC processors with a 10 MHz, non-split transaction bus. We associated a 64 KB, 32 bytes/line, direct-mapped cache with each processor. The cache protocol was write through with no write allocate, similar to the CY605 (Ross Technology Inc. 1990). Shared copies were invalidated on a write. The bus used a round robin arbitration scheme.

The simulations were driven by parallel versions of a Matrix Multiply (MM) program and a Successive Over-Relaxation (SOR) program, both applied to a 128x128 matrix. The results of these experiments are shown in Figure 4, which plots error (% difference between the execution times predicted by the two simulators) against the number of processors in the system. The larger errors for SOR may be attributed to the effect of different data placements in the direct-mapped cache by the two simulators.

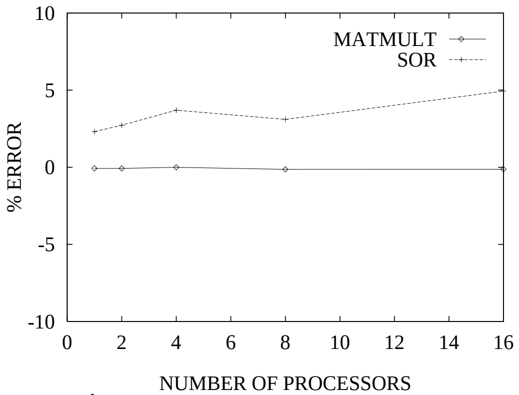

Figure 4 SPARC MP - Errors

Algorithm	Execution-Driven Overhead	Cycle-level Overhead Overhead
SOR	1354x	7682x
MM	487x	12740x

Table 3 SPARC MP - Overheads

The measured overheads for the two simulators are given in Table 3. These results demonstrate the added overhead of emulation in the case of cycle-level simulation. They show that the execution-driven simulator is approximately 6 times faster than the cycle-level simulator when running SOR, but about 26 times faster for MM. This difference is due to the way the simulators treat memory accesses. The MM program has far fewer memory writes than the SOR program, and the execution-driven simulator only incurs a high overhead on data writes, while the cycle-level simulator incurs this overhead on all instruction and data accesses. The overheads for the execution-driven simulator for the SOR algorithm are somewhat higher than they were when simulating the Sequent Symmetry. This is because the simulated SPARC system used a write-through protocol, resulting in a larger number of bus accesses than with the write-back protocol used in the Sequent.

CONCLUSIONS

Execution-driven simulation can be an effective performance evaluation tool for shared-memory multiprocessors. Although it is not as efficient as it is for message-based architectures, it is 5 to 25 times faster than cycle-level simulation and has an error of less than 5% for most simulations. Predictions of relative performance metrics such as speedup tend to be even more accurate, making this technique especially attractive for comparative investigations of parallel system designs.

REFERENCES

Agarwal, A.; Sites, R.L.; and M. Horowitz. 1986. "ATUM: A New Technique for Capturing Address Traces Using Microcode." In *Proceedings of The 13th Computer Architecture Symposium* (June). Vol 14. Pages 119-127.

BBN Laboratories. 1985. "*Butterfly (TM) Parallel Processor Overview*" (Version 1).

Borg, A.; Kessler, R.E.; and D.W. Wall. 1990. "Generation and Analysis of Very Long Address Traces." In *Proceedings of the 17th Computer Architecture Symposium* (May).

Brewer, E.A.; Dellarocas, C.N.; Colbrook, A.; and W.E. Weihl. 1991. "PROTEUS: A High-Performance Parallel-Architecture Simulator." Technical Report MIT/LCS/TR-516. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA (September).

Covington, R.G.; Madala, S.; Mehta, V.; Jump, J.R.; and J.B. Sinclair. 1988. "The Rice Parallel Processing Testbed." In *Proceedings of the ACM SIGMETRICS Conference* (May). Pages 4-11.

Covington, R.G.; Dwarkadas, S.; Jump, J.R.; Madala, S.; and J.B. Sinclair. 1991. "Efficient Simulation of Parallel Computer Systems," *International Journal of Computer Simulation.* (June) Vol 1, No 1. Pages 31-58.

Davis, H.; Goldschmidt, S.R.; and J. Hennessy. 1990. "Tango: A Multiprocessor Simulation and Tracing System." Technical Report, Computer Systems Laboratory, Stanford University.

Dwarkadas, S.; Jump, J.R.; and J.B. Sinclair. 1989. "Efficient Simulation of Cache Memories." In *Proceedings of the Winter Simulation Conference (invited paper).* (December. Pages 1032-1041.

Dwarkadas, S.; Jump, J.R.; Mukherjee, R.; and J.B. Sinclair. 1992. "Execution-Driven Simulation of Shared-Memory Multiprocessors." Technical Report 9204. Department of Electrical and Computer Engineering, Rice University.

Eggers, S.J.; Keppel, D.R.; Koldinger, E.J.; and H.M. Levy. 1990. "Techniques for Efficient Inline Tracing on a Shared Memory Multiprocessor." In Proceedings of the ACM SIGMETRICS Conference.(May).

Federwisch, M.; and L. Ball. 1990. "MPSAS: A Programmer and User Manual." Sun Microsystems.

J. Greenwood. 1992. "The Design of a Scalable Hierarchical-Bus, Shared-Memory Multiprocessor." M.S. Thesis. Department of Electrical and Computer Engineering, Rice University. (May).

Lovett, T.; and S. Thakkar. 1988. "The Symmetry Multiprocessor System." in *Proceedings of the International Conference on Parallel Processing.* (August). Pages 303-310.

ROSS Technology Inc. 1990. SPARC RISC User's Guide, 2nd Edition.

Stunkel, C.B.; and W.K. Fuchs. 1989. "TRAPEDS: Producing Traces for Multicomputers Via Execution Driven Simulation." In *Proceedings of the ACM SIGMETRICS Conference.* (May). Pages 70-78.