Parallel Program Traces for Accurate Prediction of Proposed Cluster Performance

W. E. Cohen*, W. D. Garrett+, and R. K. Gaede*

***Electrical and Computer Engineering**
University of Alabama in Huntsville
Huntsville, AL 35899
{cohen,gaede}@ece.uah.edu

+Mitre Corporation
1500 Perimeter Pkwy., Suite 310
Huntsville, AL 35806
wgarrett@mitre.org

Abstract

The ULTRA (UAH Logging, Trace Recording, and Analysis) instrumentation system provides an accurate and low-cost means of collecting traces of Message Passing Interface (MPI) program execution. These traces preserve the original parallel program's data-dependencies by recording each MPI operation performed; the message source, destination, and size; and the number of application instructions preceding the operation. Using these traces enables low-cost simulations for determining the performance of various node and communication hardware combinations although the cluster used to collect the data and the cluster being simulated differ significantly.

The portability of ULTRA traces is in direct contrast to time-based measurements which have previously been used. With time-based measurements, it is impossible to remove artifacts caused by network contention or data dependencies, thereby rendering them ineffective for drawing conclusions about any configurations other than the one used to collect the measurements. Further, ULTRA modifications in the Linux kernel allow the Intel Pentium II processor's performance monitoring hardware to track the application instructions counts on a per thread basis without interference. Thus, multiple threads mapped to single processor in a small cluster can be used to collect accurate trace data for larger clusters.

Traces for the NAS BT, SP, CG, EP, IS, LU, and MG benchmarks, representing 5.8×10^{12} instructions, have been used to drive simulations of PC clusters with fast Ethernet switches and hubs. There was good agreement between the predictions made by the simulators and the original data collection computer. Two representative benchmarks, CG and SP have been selected from the suite and discussed. The portability of the traces is demonstrated by using the data collected for the CG benchmark executing on a cluster with a fast Ethernet switch to accurately predict the performance of a cluster with a fast Ethernet hub. This portability of the trace data allows estimates to be made for networking systems such as Gigabit Ethernet, Myrinet, and proposed optical interconnects.

1. Introduction

Due to the interactions between hardware, software, and interconnection topology, it is often difficult to predict the performance of a proposed cluster for a particular parallel computer system before it is constructed. The same hardware may provide very different levels of performance for various application programs. Thus, characterization of the application programs is essential. The trillions (10^{12}) of instructions in the typical application program renders instruction-level simulators impractical for characterization. Thus, the only alternative is to execute the code on existing parallel computers and attempt to extrapolate the data collected to the proposed computer. The major drawback to this approach is the artifacts in the data from the existing data collection computer. The ULTRA (UAH Logging, Trace Recording, and Analysis) instrumentation was developed to minimize the artifacts in the collected data.

The ULTRA instrumentation consists of a Message Passing Interface (MPI) [7] wrapper that is inserted between the application code and the communication library. The ULTRA instrumentation records a trace of each processor's actions during the program's execution. The traces log every communication operation performed by each processor and the number of instructions executed in the application code between communication operations. The overhead incurred by the wrapper is relatively low (about 10 microseconds per sample). The wrapper uses the performance monitoring hardware on the Intel Pentium II [2] to count the application instructions.

The traces contain enough information to reconstruct the data-dependencies of the original program. The traces also divide the data into intra-processor and inter-processor metrics. Thus, the events in the traces can be retimed based on the costs and constraints of the proposed parallel computer. However, a first step in this goal of predicting machine performance is to verify that the simulator accurately predicts the performance of the system used to collect the data. The simulator developed for this work accurately predicts the performance of the original data collection computer. With the basic costs of communication operations for proposed networks, the simulator can predict the performance of proposed systems.

Each entry in the trace files represents one event for the event-driven simulator. Because each entry in the trace file represents thousands to millions of instructions, the simulator requires fewer computational resources than the data collection. For the NAS class A benchmarks [1] running on one to 16 processors, the ratio of application instructions per simulator instruction ranged between 1,402,732 and 17, allowing the simulator to execute in seconds on a uniprocessor PC, while the original data collection runs required minutes on a PC cluster. This efficiency provides an ideal platform on which parameter studies can be conducted to quickly determine which set of interconnection characteristics can provide the best performance for an application.

Traces representing 5.8×10^{12} application instructions were collected for the NAS class A benchmarks SP, BT, MG, IS, LU, EP, and CG running on one to 36 processes. Section 2 discusses ULTRA's instrumentation approach in greater detail. Section 3 describes the PC cluster and system software used to collect the traces. In Section 4 two sets of representative traces for the SP and CG benchmarks are used to drive the simulator and show the simulator's low cost and accuracy despite differences between the data collection and simulated computer systems. The summary of the benefits of this approach are in Section 5.

2. Instrumentation

The ULTRA instrumentation provides a simple, portable method of instrumenting parallel programs to efficiently collect measurements that accurately reflect the intrinsic characteristics of parallel programs. The intrinsic characteristics are measurements unaffected by the particular combination of hardware and software used to run the program. The instrumentation only introduces a small amount of overhead (approximately 10 microseconds) when an MPI communication library function is called, allowing data to be collected on large production runs of parallel programs. Finally, the source code of the application program is unchanged by ULTRA. The current version of the instrumentation accomplishes all this by using profiling wrappers inserted between the application code and the functions that implement the MPI operations.

The profiling wrappers allow the instrumentation to be inserted by simply relinking the application code with the instrumented version of the MPI communication library. From the application code's point of view, the instrumented library is identical to the original, uninstrumented version. The instrumentation may affect the overall performance of the code and may be observable to the application code through differences in the amount of time required to perform certain operations. However, the instrumentation itself does not use time-based measurements and the data it collects will not be affected assuming that the application code is free of non-deterministic programming techniques such as dynamic load balancing.

Inter-processor metrics are easy to obtain; when an MPI operation is called, the arguments include the source, destination, and size of each message. Intra-processor metrics are obtained using the performance monitoring hardware included on the Intel Pentium II processor, allowing inexpensive measurements of instruction counts and data memory references. The raw performance monitoring hardware measures events across all running processes. This approach is undesirable because it introduces artifacts into the collected data. Kernel modifications such as IN-Tune [6] and PAPI [5] virtualize the performance monitoring hardware, making it appear that each process has its own set of performance monitoring registers. The kernel modification causes a very slight increase in the cost of performing a context switch between processes, a cost which is outweighed by the improved accuracy of data collected for each process.

The performance monitoring registers measurements are recorded as intervals, the difference between the counts observed at the exit from the previous MPI wrapper and the entry into the current MPI wrapper. MPI functions which do not create a log entry must adjust the instruction count to eliminate their contributions. One example of such a function is the non-blocking receive. The event for the receive is logged when the MPI test is performed and verified the message has arrived. The count contribution from the non-blocking receive is computed and added to the previously recorded count. This approach is very efficient for the NAS benchmark because the MPI calls are typically separated by several million instructions. It also eliminates the contribution to the instruction count from the MPI library, reducing the artifacts caused by the use of a particular implementation of MPI.

3. Data Collection

All the traces for the NAS benchmarks were collected on a PC cluster, which consists of 18 identical Intergraph TDZ2000GL1 machines configured with:

- 333 MHz Pentium II processors
- 440FX motherboard chipset
- 128 MB DRAM (66MHz system bus)
- (2) 4 GB IDE drives
- Intel EtherExpress Pro 10/100

A dedicated 24-port 3Com SuperStack II 3300 Fast Ethernet switch connects the PCs. To verify that the changes in the network did not corrupt the measurements, some of the experiments were repeated with a 16-port Netgear FE116 100Mb/s hub. A comparison of the traces showed that the effects due to changes in the network hardware were negligible for the BT class A benchmark running on 9 processors.

All the PCs in the cluster run a modified version of the Linux 2.2.5-15 kernel that includes the University of Tennessee Knoxville Portable Application Performance Instrumentation (PAPI) patch [5] which virtualizes the processors' performance monitoring hardware. The code for the experiments were compiled using the EGCS-2.91.66 (GNU C Compiler) and the F77 version 0.5.24-19981002 with the ''-O3'' level of optimization. The traces were collected with programs linked with MPICH 1.1.2 [4]. Some traces were also recorded with LAM 6.3-b1 [3] as the communication library to verify the implementation of MPI had a minimal impact on the traces. There was little change due to the MPI implementation.

Traces representing 5.8×10^{12} application instructions were collected for the NAS BT, SP, CG, EP, IS, LU, and MG class A benchmarks. BT and SP were run for one, four, nine, 16, 25, and 36 processes. CG, EP, IS, LU, and MG were run for one, four, eight, 16, and 32 processes. Given the small amount of variation between traces collected under differing conditions, it appears that typical artifacts in time-based measurements have been eliminated. Thus, the traces provide good data for driving a simulator.

4. Simulator Accuracy

Two traces, SP and CG, were chosen to show the good agreement between the simulator and original data collection run. SP is a typical NAS benchmark in terms of the number of application instructions per byte sent. CG executes the fewest instructions per byte sent and presents a much heavier load to the interconnection network. Although these are the only two benchmarks detailed in this paper due to space considerations, the simulator has similar performance for the other benchmarks with the exception of IS. The simulator does not accurately model collective operations and IS consists almost entirely of collective operations.

The simulator has adjustable parameters such as the Cycles Per Instruction (CPI), the Communication Startup Overhead Time (CSOT), and the Per Byte Communication Time (PBCT). Each of these parameters were calculated for the benchmarks. The CPI was calculated by measuring the runtime for the uniprocessor run of the benchmark and multiplying this by the CPU clock rate and dividing by the total instructions executed. The CSOT and the PBCT were determined by using the `perftest` utility provided with the distribution of MPICH. The `perftest` utility sends many different message sizes and measures the times that it took to send these messages. It then performs linear regression on a series of timed measurements to determine CSOT and PBCT. CSOT is the Y-intercept of the line and PBCT is the slope of the line.

Table 1 presents the data attained by the simulator for the SP benchmark for the switch configuration. Figure 1 provides a graph of the simulator data run tracking the original data collection run of SP on the switch. Three of these four runs provided margins of error of 5% or less. Of the one that provided larger margins of error, the SP nine processor still provided a result that was within a margin of error of 10%.

The simulation results for CG listed in Table 1 show very good agreement; three of the four runs are within 5%. However, the sixteen processor run of CG displayed an error margin of 35%. This large margin

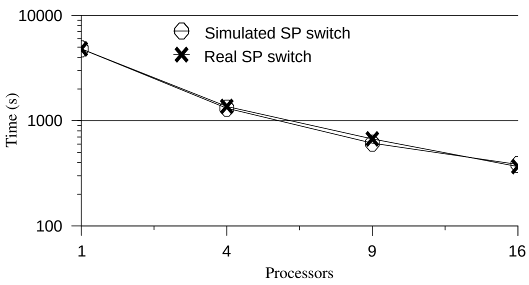

Figure 1: Plot of simulated and real runtimes of NAS class A SP benchmark on fast Ethernet switch.

of error is due to the startup overhead, which is not captured by the trace collection routines, and it becomes a significant factor in the error as the runtimes recorded become much smaller. The measured startup overhead was 4.31 seconds for a trivial sixteen processor program and the real runtime recorded for the sixteen processor CG run was 26.36 seconds.

The last column of Table 1 list ratios of the total application instructions for the real benchmark data collection run to the instructions required to simulate the benchmark. This measure provides a fair comparison of the improvement of the simulator over the real benchmark runs. The simulation of the CG benchmark provides a speedup over the real run that ranges from 23,933 times faster for the uniprocessor run to 17 times faster for the 16 processor run. The CG benchmark is the shortest of the benchmarks and provides the lowest speedup ratio of all the benchmarks. The SP simulation is 956,467 times faster for the uniprocessor run and is 182 times faster for the 16 processor run. It should be noted that these values are conservative in that although the instructions counted for the simulator encompassed all instructions executed, the instructions counted for the real benchmark runs do not include instructions that are executed by the MPI library or the kernel for the message passing operations.

To validate the prediction capabilities of the simulator, one benchmark was chosen to evaluate for a different architecture, a fast Ethernet hub, while using the traces collected for the switch configuration. For this purpose, the CG benchmark was chosen because it provides a significant load to the interconnection network. Table 2 presents the comparison of the real runtimes and the simulated runtimes of the hub that were obtained using the switch trace file. Figure 2 graphically depicts this comparison. Figure 2 shows that the simulator

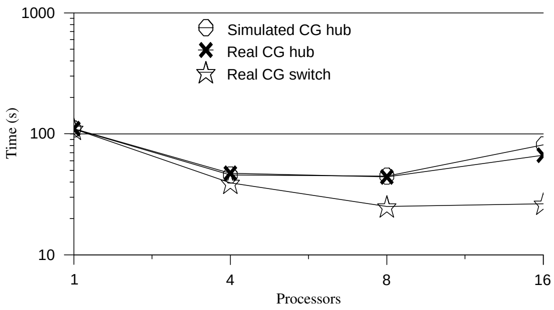

Figure 2: Plot of simulated and real runtimes of NAS class A CG benchmark on fast Ethernet hub.

accurately predicts that the runtimes of the CG benchmark will begin to level off for the four and the eight processor runs and accurately predicts that the runtime increases for the sixteen processor run.

In Figure 2 it can be seen that the curve associated with the switch is not similar to the curve associated with the hub. The graph shows that as processors are increased, the hub configuration experiences a leveling off of the runtime curve for the four and eight processor runs and experiences an increase in

Table 1: Runtimes measurement of CG and SP on Switch and resulting simulation using traces.

Benchmark			Real Switch Time (s)	Sim. Switch Time(s)	Sim. Time/ Real Time	Total App. Inst/ Sim Inst.
SP	A	1	4795.00	4791.19	1.00	956466.7
SP	A	4	1358.07	1306.20	0.96	1494.8
SP	A	9	668.46	609.81	0.91	422.3
SP	A	16	365.25	383.03	1.05	182.3
CG	A	1	109.69	109.77	1.00	23933.3
CG	A	4	39.33	39.95	1.02	106.6
CG	A	8	25.04	24.16	0.96	32.5
CG	A	16	26.36	17.20	0.65	17.5

Table 2: Comparison of CG measurements on hub and simulation of CG on hub with switch trace as input.

Benchmark			Real Hub Time(s)	Sim. Hub Time (s)	Sim./ Real Hub Time	Real Switch/ Real Hub Time
CG	A	1	109.69	109.77	1.00	1.00
CG	A	4	45.53	47.09	1.03	0.86
CG	A	8	44.59	44.03	0.99	0.56
CG	A	16	80.95	66.49	0.82	0.33

the runtimes for the sixteen processor run. The switch however experiences a leveling off of the runtime curve for the eight and sixteen processor runs. These differences in the graphs of the two configurations shown in Figure 2 and the ‘‘Real Switch/Real Hub Time’’ ratio shown in Figure 2 leads to the conclusion that a simple linear scaling factor would not be appropriate for prediction the runtimes of the hub from the switch runtimes and that the simulator more accurately predicts the performance for the hub than would a linear scaling factor applied to the runtime for the cluster with a switch.

5. Conclusions

The ULTRA instrumentation provides a means of inexpensively collecting traces which accurately characterize program execution. These traces preserve the data dependencies of the original program and minimize the artifacts of the data collection computer. The simulators using the resulting traces can accurately predict the performance of parallel computer hardware that differs from the computer used to collect the traces, allowing designers to more easily determine a configuration which would best serve the needs of a particular application program.

6. Acknowledgments

The authors would like to acknowledge Dr. Jeffrey Kulick who initially supervised Jeremy Rodgers and Ashley Brown on the development of a Linux kernel that made this work feasible. The PC cluster and network equipment used for this research were purchased with money from NSF grants 9729889 and 9720653. UAH contract number: DAAH01-97-D-R005 DO 19 from the Army Software Engineering Directorate at Redstone Arsenal provided support for equipment and research time. We would also like to thank the people at UTK making the PAPI instrumentation patches publicly available.

7. References

[1] Bailey, D., T. Harris, W. Saphir, R. Wijngaart, A. Woo, and M. Yarrow, ‘‘The NAS Parallel Benchmarks 2.0,’’ NASA Ames Research Center, Moffett Field, California, NAS-95-020, December 1995.

[2] Intel Architecture Optimization Manual, Mt. Prospect, Illinois, 1997, Intel, http://www.intel.com/design/pentium/manuals/-242816.htm, order number 242816-003.

[3] LAM / MPI Parallel Computing, http://www.mpi.nd.edu/lam/.

[4] MPICH-A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich/.

[5] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho, ‘‘PAPI: A Portable Interface to Hardware Performance Counters,’’ presented at the DoD High Performance Computing Modernization Program User Group Conference, in Monterey, CA on June 7-10, 1999. http://icl.cs.utk.edu/projects/papi/dodugm99/pa-pi.html.

[6] J. Rodgers, Rhonda Kay Gaede, and Jeffrey H. Kulick, ‘‘IN-Tune: An In-Situ Non-Invasive Performance Tuning Tool for Multi-threaded Linux on Symmetric Multiprocessing Pentium Workstations‘‘, Software - Practice and Experience, Volume 29(9), pp. 775-792, September, 1999.

[7] M. Snir, et al., MPI: The Complete Reference, MIT Press, Cambridge, MA, 1995.