10:04:16 October 1, 1999

Interconnection Network Independent Characterization of Communication Traffic in the NAS Benchmarks via Processor Performance Monitoring Hardware

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

***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 UAH Logging, Trace Recording, and Analysis instrumentation (ULTRA) provides highly repeatable (0.0002% variation) application instruction counts for parallel programs which are invariant to the communication network used, the number of processors used, and the MPI communication library used. ULTRA, implemented as an MPI profiling wrapper, avoids the data collection system artifacts of time-based measurements by using instruction counts as the basic measure of work performed and records the operation performed and the amount of data sent for each network operation. These measurements can be scaled appropriately for various target architectures. ULTRA's instrumentation overhead is minimized by using the Pentium II processors's performance monitoring hardware, allowing large, production-run applications to be quickly characterized.

Traces of the NAS benchmarks representing 6.67×10^{12} application instructions were generated by ULTRA. The application instructions executed per byte injected into the network and the instructions executed per message sent were computed from the traces. These values can be scaled by the expected processor performance to estimate the minimum network performance required to support the programs. It is impossible to use time-based measurements for this purpose due to measurement artifacts caused by the background processes and the communication network of the data collection system.

1. Introduction

The growing complexity of parallel computer hardware and software makes it difficult for designers to accurately determine the performance of a proposed system without extensively using simulations. Given the size of proposed parallel computer systems, detailed instruction-level simulations are a very expensive method of determining performance. Lower cost, less detailed simulations, such as queuing system models, are required to provide some insight into the effects that different design parameter values have on the overall performance of the parallel computer. However, using the less detailed models means an attendant loss in accuracy in obtaining parameters that represent the characteristics of parallel programs. The parameters used by researchers in many cases are not based on any actual measurements. Further, when actual measurements are used, they can be corrupted by the specific characteristics of the computer system used for the data collection.

Time-based metrics are commonly used by researchers to describe the workload characteristics of parallel programs. This approach has the advantages that (a) most machines provide timers, (b) measurements of when time events occur can be efficiently obtained, and (c) time provides some indication performance. However, the time required is very dependent on the particular computer. Even processors with the same instruction set may have an order of magnitude difference in the processor clock rates, e.g. the Intel IA-32 architecture clocks range from 60 MHz to 600 MHz. Additionally, the timing of communication operations can be affected by the interconnection network used in the system. Thus, applying time-based measurements to computer systems other than the specific system (hardware *and* software) used for the measurements is very difficult.

A better approach is to use portable metrics that are invariant with respect to a computer's implementation and allow these metrics to be scaled appropriately to provide an indication of the amount of time required for the operation. Current processors such as the Intel Pentium II [Intel97], MIPS R1000 [Zagha96], and Power PC [Welbon94], all provide performance monitoring hardware to count instructions and memory references which are invariant with respect to the processor implementation. However, these processors have only a single copy of the performance monitoring hardware shared by all the processes executing on the processor. Special measures must be taken by the kernel to virtualize the performance monitoring hardware and make it appear that each process has its own copy of the performance monitoring hardware

[Rodgers99][Mucci99]. These modifications are essential for obtaining accurate measurements because almost all parallel machines have multiple processes on each processor. The additional processes include daemons to handle the processor's incoming and outgoing messages, so it is impossible to disable these processes while executing a parallel program.

The UAH Logging, Trace Recording, and Analysis instrumentation (ULTRA) has been developed to efficiently collect portable metrics that describe the processor workload and the communication operations of application programs written using the Message Passing Interface (MPI) libraries. ULTRA uses the performance monitoring hardware to reduce the overhead of collecting processor workload information. The traces collected by this instrumentation can be scaled as required to match the characteristics of the computer system being simulated. An additional benefit is that the traces lump the instructions between communication operations together, eliminating the need for detailed instruction-level simulation. This compression can significantly reduce the cost of determining the performance of a particular interconnection network, since the number of instructions between communication operations ranged from 10,000 (Class A CG on 32 processors) to 52 million (Class A BT on 4 processors) in the NAS benchmarks examined.

Section 2 describes the related work. Section 3 describes the equipment used to perform the experiments and the tests performed to determine the insensitivity of the instrumentation to changes in the data collection system. The resulting data collected from the class A NAS benchmarks [Bailey95] are given in Section 4. Section 5 describes how the measurements can be used to estimate the minimum network performance required. Section 6 summarizes the work performed.

2. Related Work

The goal of the ULTRA instrumentation is to provide accurate characterization of the intrinsic characteristics of large, parallel message passing programs that can be applied to other computer architectures, rather than to tune performance (e.g. Paradyn [Miller95], Paravision [Nutt95], and Pablo [Reed94]), which focuses on identifying performance problems in code on existing computers. Thus, most of the performance tools focus on time-based metrics because they try to identify how a program spends its time. However, the performance bottlenecks on one machine may not be the same as the performance bottlenecks on another machine.

The ULTRA instrumentation assumes that the code has been optimized and already executes with reasonable efficiency on existing systems. The ULTRA instrumentation's purpose is to gather traces and statistics which are free from artifacts from the data collection machine. These statistics and traces are useful as inputs into lower cost analytical models and simulators of multiprocessor machines to cull the design space as suggested by Sorin, et. al. [Sorin98]. To obtain high performance, the ULTRA instrumentation assumes that the processors in the parallel instrumentation system used to collect the data are similar to the processors in the simulations. The alternative of using software that simulates the execution of instructions at 300,000 instructions per second [Durbhakula98] is not a feasible approach for examining the characteristics of programs requiring hundreds of billions of instructions for a single simulation.

An approach similar to the Wisconsin Wind Tunnel (WWT) [Mukherjee97] allows ULTRA to execute code at normal rates between communication operations. However, the WWT system also couples the simulation with the data collection because the fine-grained shared memory operations would produce immense traces. The resulting approach requires all the instructions in the application code to be executed for each simulation, a rather expensive proposition for large codes. The ULTRA instrumentation records a trace of communication operations for each processor and information to estimate the time between communication operations for different performance processors. The metrics used to compute time estimates are collected by the performance monitoring hardware in the processor. The message passing programs have much coarser-grained communications than shared memory programs, millions of instructions between communication operations rather than hundreds of instructions, making it feasible for ULTRA to store traces.

An approach taken by Chong, et. al. [Chong98] to make the data collection computer's characteristics more closely match the architecture being studied was to use a portion of processors in the machine to produce additional traffic in the data collection computer to adjust the effective bandwidth and latency. To simulate networks with higher performance than the machine being used for the simulation, the clock rate of the processors was slowed. This is an unattractive solution for three reasons.

1. It is desirable for the processors to execute code as quickly as possible.
2. There are lower bounds on how much a processor can be slowed due to the dynamic circuitry in most processors.
3. In many machines, changing the clock frequency requires physical access to the machine to change jumpers on circuit boards.

Martin, et. al. [Martin97], used the processor in the Myrinet network interface to adjust the effective bandwidth and latency of the system. Again this approach allows the study of network configurations slower than the network in the parallel system, but it is difficult to simulate the operation of higher performance networks.

3. The Experiments

The experiments were designed to verify that the instrumentation was insensitive to the physical system used to collect the data and to collect data that provides insight into the characteristics of parallel programs. All the experiments for this work were performed on a 16-node PC cluster of identical Intergraph TDZ2000GL1 configured with:

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

For most of the experiments, a dedicated 24-port 3Com SuperStack II 3300 Fast Ethernet switch connected the PCs. To test the ability of the instrumentation to eliminate artifacts due to the network, some of the experiments were repeated with a 16-port Netgear FE116 100 Mb/s hub. The traffic on the network on these interconnection networks was limited to traffic for these machines.

All the PCs in the cluster run a modified version of the Linux 2.2.5-15 kernel that includes the UTK Portable Application Performance Instrumentation (PAPI) patch [Mucci99] 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. Most experiments used LAM 6.3-b1 as the communication library. However, some runs were made with MPICH 1.1.2 to compare the sensitivity of the instrumentation to the communication library. All the experiments were performed on the NAS 2.3 parallel benchmarks.

3.1. Measurement Approach

The goal of the ULTRA instrumentation is to provide 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 the particular combination of hardware and software used to run the program. The ideal level of efficiency would be to have the instrumentation introduce no overhead at all, allowing data to be collected on large production runs of parallel programs (10^{12} total application instructions or more). Finally, the instrumentation should not require changes to the source code of the application program. 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 use of profiling wrappers allows the instrumentation to be inserted by simply relinking the application code with the instrumented version of the MPI communication library. From the point of view of the application code, the libraries are identical to the original, uninstrumented versions. 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 does not use non-deterministic programming techniques such as dynamic load balancing.

When an MPI operation is called, the arguments include the source, destination, and size of each message. However, what is occurring in the application program between calls to the MPI library, is harder to discern. In the past, one would have to resort to binary rewrites of the code using tools such as EEL [Larus95] or ATOM [Srivastava94] to insert instrumentation to count memory accesses and instructions. Even with optimizations to limit the instrumentation code to one instrumentation operation per basic block, this can introduce significant overhead. An alternative approach taken for this work is the use of the performance monitoring hardware included on the processor. In most cases, the raw performance monitoring hardware measures events across all running processes. This is not desirable because it introduces artifacts into the collected data. Kernel modifications such as IN-Tune [Rodgers99] and PAPI [Mucci99] 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, which is outweighed by the improved accuracy of data for each process.

The current instrumentation wrapperscount only the number of instructions in the application code. It is a trivial modification to set the performance monitoring hardware to measure other quantities such as the number of floating point operations performed and the number of memory accesses. The measurements from the performance monitoring registers are recorded as intervals, the difference between the counts observed at the exit from the previous wrapper and the entry into the current wrapper. MPI functions which do not create a log entry must adjust the counter to eliminate their contributions. One example of such 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 provides measurements that are insensitive to the computer system used to collect the data as shown by the following section.

3.2. Measurement System Verification

The measurements taken by this system should be relatively independent of the underlying hardware and software used to collect the data. Data collection was performed with the NAS BT benchmark under a variety of conditions to determine whether the instrumentation is sensitive to differences in the system load, interconnection network, or message passing libraries. Three experiments were performed:

1. Changing the number of physical processors used to run the parallel program
2. Replacing the interconnection network's switch with a hub
3. Replacing the LAM implementation of MPI with the MPICH implementation of MPI.

Despite wide variations in wall clock times caused by the changes in the data collection system, there was extremely good agreement between the recorded traces.

3.2.1. Virtual Processor Insensitivity

In many cases, the machine used to collect the traces may have fewer processors than the target architecture. Thus, the physical processors must be virtualized to emulate the required number of processors. To test the instrumentation's sensitivity to variation in the number of physical processors used to collect data, the same sample class NAS BT benchmark compiled with the MPICH implementation of MPI was run on both four processors and a single processor. The runtimes were significantly different, 4.49 seconds for the 4-processor run versus 2 minutes 24 seconds for the single processor run. The single processor run is slowed down excessively because MPICH uses polling to check whether messages have arrived. Thus, rather than performing useful work on the single processor run, much of the time is spent polling.

In general, there was excellent agreement between the runs. Of the 5936 processing intervals, 1240 were identical, 2483 differed by less than 10 instructions, and 5197 differed by less than 100 instructions. The differences between the instruction counts were sorted and Figure 1 shows the value of those differences for the 5936 samples taken. The first entry in each of four virtual processor traces had significantly larger differences ($53x10^6$ instructions to $41x10^6$ instructions) due to the MPICH's polling loop in an MPI function that was not instrumented. The instruction counts were 7.60564e+08 for the four processor run and 9.54288e+08 for the uniprocessor run. If the anomalous log entries at the beginning of each processor's trace are excluded, the instruction counts are 760,410,080 for the four processor run and 760,486,888 for the uniprocessor run, only 76,808 difference in the application instruction counted, a 0.01% difference.

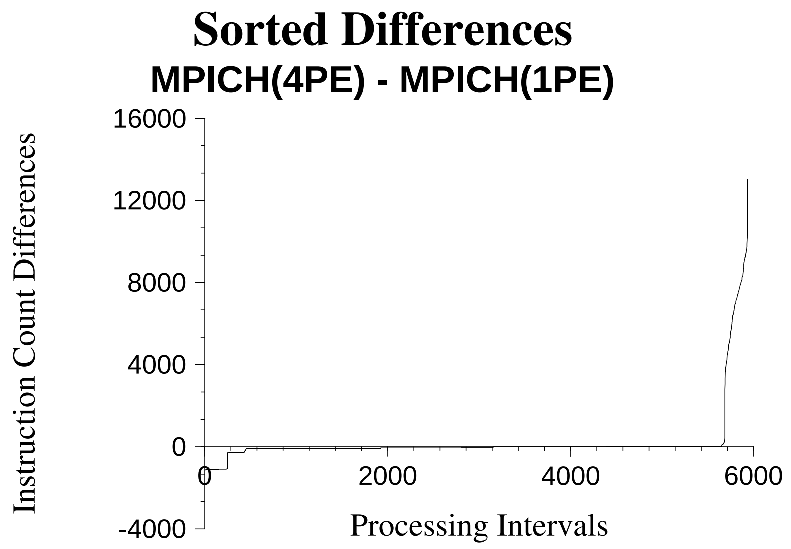

Figure 1: Difference in instruction counts between BT running on 4 processors and 1 processor (BT compile with MPICH).

3.2.2. MPI Implementation Insensitivity

The instrumentation is portable due to the well defined profiling interface specified for MPI. The same code for BT was compiled and linked with the LAM implementation of MPI. The code was run on the same group of four processors. The traces collected from this run were compared to the MPICH run with four processors. The LAM run required 10.20 seconds to complete compared with 4.49 seconds for MPICH. The LAM run measurements were in good agreement with the MPICH runs. Figure 2 plots the sorted differences between the LAM run and the MPICH run on four processors. Of the 5936 measurements, 2482 differed by less than 10 instructions, 5201 differed by less than 100 instructions and only nine measurements differed by more than 10,000 instructions.

The difference between the LAM run and MPICH run is greater than the comparison of MPICH runs because of differences in their Fortran to C wrappers. The instrumentation wrappers surround the C code used to implement the MPI functions, but they do not surround the Fortran to C wrappers. Thus, the Fortran to C wrappers appear to be part of the application code to the instrumentation. Despite the variations caused by the Fortran to C wrappers, the total application instruction counts differed by only 0.16% (761,801,465 for LAM versus 760,563,974 for MPICH using 4 processors).

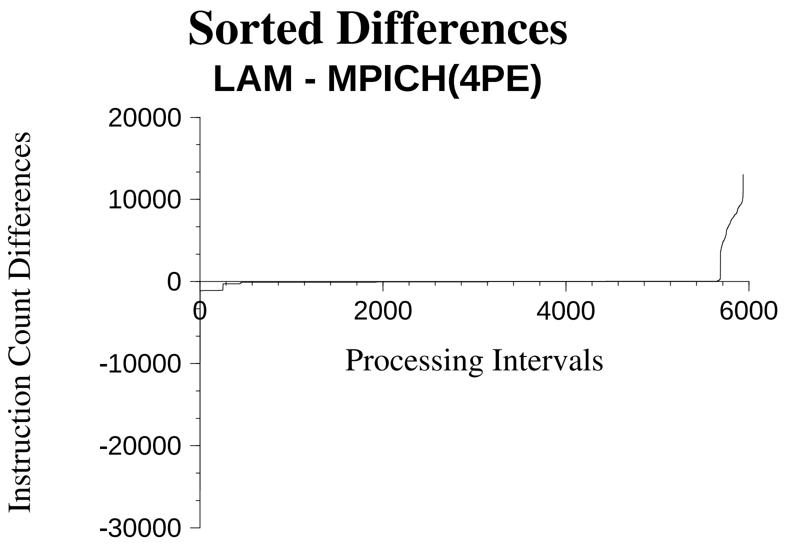

Figure 2: Difference in instruction counts between LAM and MPICH for BT.

3.2.3. Communication Network Insensitivity

The final aspect of portability is the ULTRA instrumentation's insensitivity to the interconnection hardware used for communication of data between the processors. To test the sensitivity of the instrumentation to variations in the interconnection network, two runs of the class A NAS BT benchmark were made. One run of the benchmark was made using a 24-port 3Com Superstack II 3300 switch to connect the PCs. A second run was made replacing the fast Ethernet switch with a 16-port FE116 Netgear fast Ethernet hub.

The run using the switch required 835.2 seconds while the run using the hub required 920.4 seconds (10.2% more time). The resulting traces were compared. The difference in the instruction counts was computed and plotted as a histogram shown in Figure 3. The measurements between the switch and hub on the LAM runs of BT were very close. Over 42,000 of the measurements (65%) had no differences at all! 57,800 (88.5%) were within 10 of each other, resulting in the peak at 0 for Figure 3. Out of the 65,304 samples, only 7 deviated by more than 100, and only one of those was significantly greater than 100 (3977). The run on the switch had 4.78226e+11 application instructions counted and the run on the hub had 4.78225e+11 application instructions, a difference of less than 0.0002%!

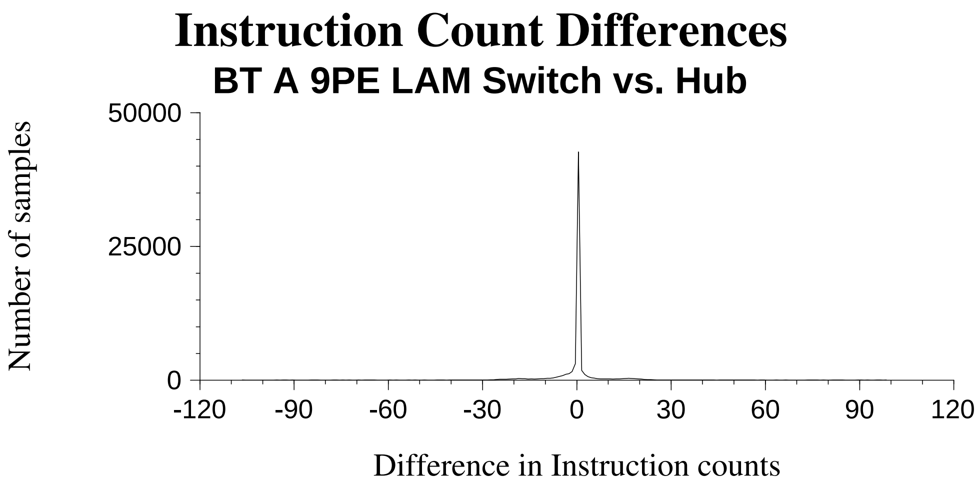

Figure 3: Histogram of difference between instruction counts for class A BT running on switch and hub.

4. NAS Benchmark Characterization

This instrumentation enables the characterization of significantly larger runs of parallel programs than traditional instruction-level simulation via software. For the NAS benchmark measurements discussed in this section, 6.66×10^{12} application instructions were executed. Making the optimistic assumptions that the communication operations required no additional instructions and the processors could on average execute one instruction per cycle, this is equivalent to over 5.5 hours of computer time. Assuming a software simulator could simulate 300,000 instruction per second [Durbhakula98], 6.166×10^{3} hours (256 days) would be required to make the equivalent runs. ULTRA's approach creates compact traces which can be used to drive numerous, low-cost, system-level simulations.

4.1. Mean Message Size

Figure 4 summarizes the average message sizes observed in the benchmarks for the various number of processors. These mean message sizes were computed by processing the ULTRA instrumentation traces and agree with earlier measurements [Cohen98]. There is a significant range in message sizes, from 117 Kbytes for the four processor run of BT to 1477 bytes for the 32 processor run of LU. The problem sizes are fixed in the NAS benchmarks. Consequently, as the number of processors working on the problem increases, the amount of data that the processor has locally decreases. Thus, the amount of data a processor communicates in a communication operation also decreases as shown in Figure 4.

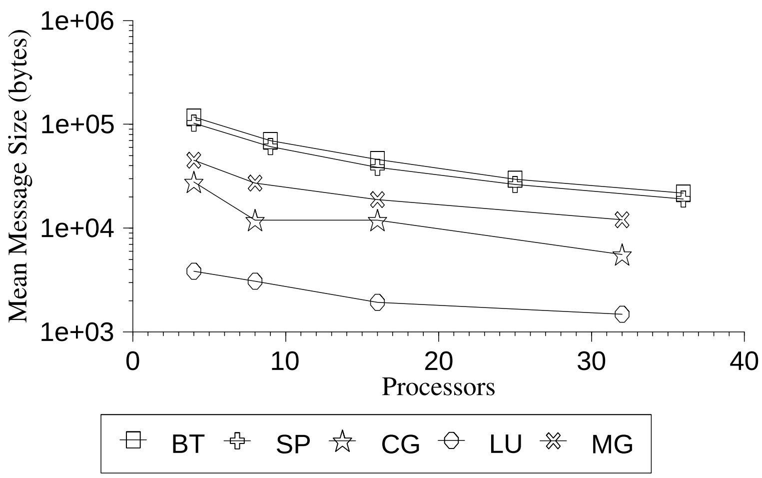

Figure 4: Mean message size for NAS benchmarks.

4.2. Total Instruction Counts

By using instruction counts as the unit of measure between communication operations, a more accurate picture of the amount of work performed by an application appears. Figure 5 contains the total number of application instructions in the traces for each application. These instruction counts exclude the operations performed by the message passing library. There would have been some difficulty obtaining similar accuracy with time-based measurements for the parallel runs requiring more processors than the cluster contained. Nine of the 30 measurements in Figure 5 are for runs requiring more processors than the cluster contained. The time-based measurements would also differ significantly due to the performance of the communication network. Because the benchmarks are fixed problem size the total count of application instruction do not vary much. The class A NAS benchmarks are considerably larger than the programs in the SPLASH benchmarks. The largest application in SPLASH required 2.83 billion instruction [Woo95]. The largest NAS benchmarks are two orders of magnitude larger than the largest SPLASH benchmarks.

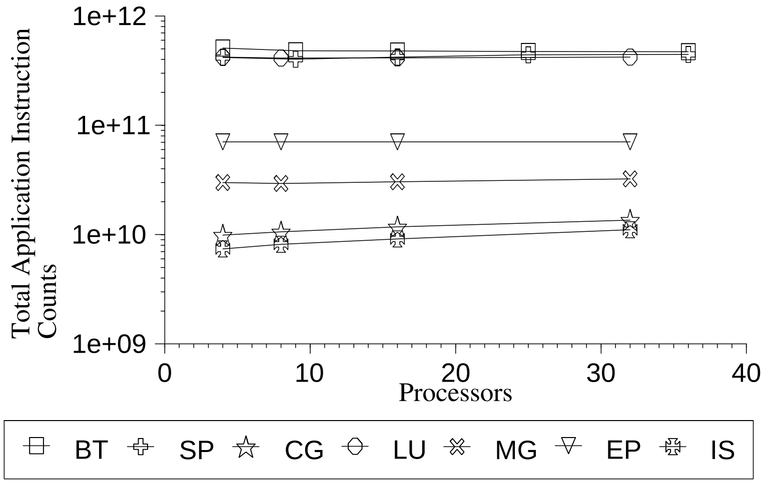

Figure 5: Total application instructions executed for NAS benchmarks.

4.3. Ratio of Computation to Communication

Given the total number of bytes sent in the application and the total instructions, an estimate was obtained for the number of application instructions executed per byte sent as shown in Figure 6. Because the integer sort (IS) uses reduction operations for most of its communication and the embarrassingly parallel benchmark (EP) does not use communication, they were not included in the graph. This ratio of application instructions per byte sent provides a means for estimating the bandwidth required for the application. CG with 32 processors is the most demanding application, requiring only 18 instructions per byte sent. In general, as more processors were used to run the benchmark, the instructions per byte communicated dropped, indicating that higher performance networks are needed for larger machines to maintain the same level of performance.

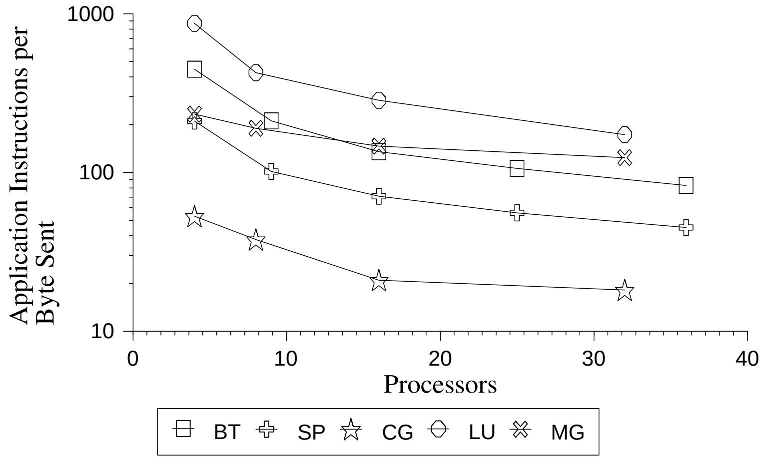

Figure 6: Average number of application instruction per byte sent in NAS benchmarks.

Figure 7 shows the average number of application instructions between sends. This is a gauge of the relative frequency of communication operations. The NAS benchmark that consistently had the fewest instructions between sends is CG. This is expected given the small message size for CG shown in Figure 4 and the few instructions per byte in Figure 6. However, LU also has relatively few instructions between sends. Despite the high number of instructions per byte in Figure 6, the small average message size results in sends occurring frequently in LU.

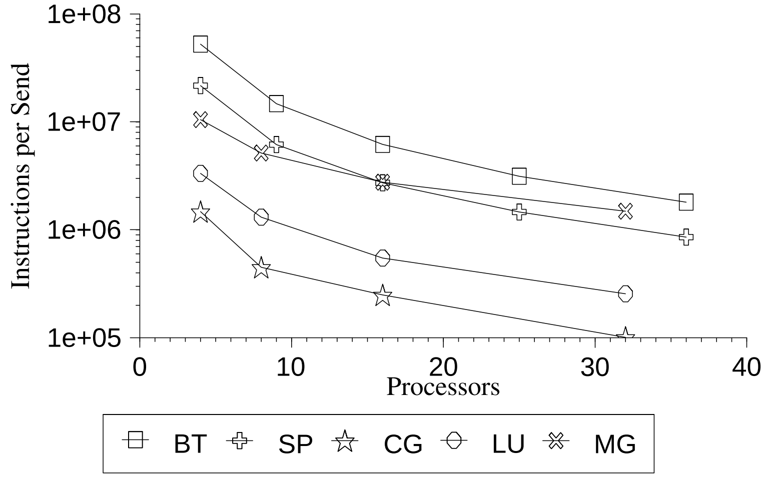

Figure 7: Average number of application instructions per message sent in NAS benchmarks.

5. Estimating Bandwidth Requirements

The main advantage of this approach to characterizing parallel programs is that the data is devoid of the usual artifacts due to processor speed and interconnection network. This purity of data allows some estimates to be made about minimum network performance required given a particular level of performance in the node processors. These estimates are rather simple lower bounds on the minimum required performance. More detailed estimates could be generated using simulators and the original traces used to generate the plots in Figures 4, 5, 6, and 7. The basic assumption is that communication and computations can be fully overlapped. The initial analysis will assume no startup overhead for sending a message, then a refinement will be made to account for the startup overhead.

In the case where there is no startup overhead and it is assumed that communications can be completely hidden by computations, the amount of time taken to send a byte over the network (t_w) must be less than the time taken to execute the number of instructions used to create a byte (t_{comp}). It is a simple process to compute t_{comp} from the instructions per byte sent (i_b) in Figure 6 multiplied by the average time required for the processor to execute an instruction (t_i). Thus, the following inequality needs to be true for the computer's network to provide sufficient performance for the application program:

$$t_w \le t_i i_b$$

The reciprocal of per byte transfer time (t_w) is the minimum bandwidth (B_p) required for a single processor. The total aggregate bandwidth required can be estimated by multiplying minimum processor bandwidth by the number of processors running the application.

For CG class A running on 32 processors, each capable of 1000 MIPS (t_i=1 ns), t_w is less than 18 ns. Thus, the network must provide a minimum bandwidth of 55 MB/s for each processor and a total bandwidth of 1760 MB/s. This is a lower bound on the required performance of the interconnection network, assuming that all communications can be perfectly overlapped with computations, there is no startup overhead, nor any contention in the network This estimate can be refined by computing the time required to send a message which includes startup overhead (t_s) and making sure that the time required to transmit the mean message of m bytes is less than the time required for the processor to execute the average number of instruction between between sends (i_s) shown in Figure 7. Again assuming that communication and computations can be perfectly overlapped, the following condition must be true:

$$t_s + t_w m \le t_i i_s$$

Through algebraic manipulation the previous equation can be solved for t_w:

$$t_w \le \frac{t_i i_s - t_s}{m}$$

Given that the processor's network bandwidth requirements are the reciprocal of t_w:

$$B_p \ge \frac{m}{t_i i_s - t_s}$$

The previous equation also bounds t_s to be less than the average time between sends. For the same example of CG executing on 32 processors, this limits t_s to less than less than 100.5 microseconds for a 1000 MIPS processor. For CG on 32 processors, t_w would have to be less than 14.5 ns given the 1.005×10^5 instructions between sends and assuming one instruction per nanosecond and 20 microseconds for t_s. This translates to a bandwidth requirement of 68.8 MB/s per processors, about 25% greater than the previous estimate made without the startup overhead.

These equations combined with the data collected by the ULTRA instrumentation provide a means of determining whether the network will limit the program's performance. A network may still limit performance even if its bandwidth exceeds theses estimates because of the assumptions made about the ability to overlap communications with computations. However, these estimates can be made without actually running the application on the target machine.

6. Conclusions

Given the complexity of computer systems, accurate characterizations of parallel programs are required to allow computer engineers to produce future machines optimized to provide the best performance possible. Although there is a growing interest in shared memory architectures, programmers will continue to refine existing programs and develop new programs that use message passing. Efficient support of these programs on future computer systems will remain an important concern.

The accuracy of the ULTRA instrumentation was demonstrated with repeatable measurements of the NAS benchmarks under a variety of conditions. Traces for the NAS parallel benchmark suite were collected. The resulting traces were analyzed and the NAS benchmarks were characterized in term of the amount of data and number of application instructions executed. Given these parameters, the computations to estimate the minimum network bandwidth required to support the applications were derived.

The ULTRA instrumentation provides an effective means of collecting traces that describe the intrinsic properties of a parallel program's execution. Analysis of the collected data provides information on the lower bound requirements for the communication network. The traces can also be used as input into an event driven simulator to provide an accurate estimate of a proposed network's performance for a particular program. Given the number of instructions between communication operation in the traces, this method can provide a significant performance advantage over instruction-level simulations.

7. Acknowledgments

The authors would like to acknowledge Dr. Jeff Kulick who initially supervised Jeremy Rodgers and Ashley Brown on the development of a Linux kernel that made measurements on a per thread basis. Dr. Kulick and Dr. James Swain provided some excellent suggestions on how to analyze and present the collected data. The PC cluster and network equipment used for this research were purchased with money from NSF grants 9729889 and 9720653. We would also like to thank the people at UTK making the PAPI instrumentation patches publicly available.

8. References

[Bailey95] 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.

[Durbhakula98] M. Durbhakula, V. S. Pai, and S. Adve, ‘‘Improving the Accuracy vs. Speed Tradeoff for Simulating Shared-Memory Multiprocessors with ILLP Processors,’’ Technical Report 9802, Dept. of Elec. and Comp. Engineering, Rice University, April, 1998.

[Chong98] F. T. Chong, R. Barua, F. Dahlgren, J. D. Kubiatowicz, and A. Aragwal, ‘‘The Sensitivity of Communication Mechanisms to Bandwidth and Latency’’, Proceedings of the 4th Int’l Symposium on High Performance Computer Architecture, Las Vegas, NV, Feb 1-4, 1998.

[Cohen98] W. E. Cohen and B. A. Mahafzah, ‘‘Statistical Analysis of Message Passing Programs to Guide Computer Design,’’ in the Proceedings of the Hawaii International Conference on System Sciences 31, Kohala Coast, Hawaii, USA, pp. VII 544-443, January 6-9, 1998.

[Intel97] Intel Architecture Optimization Manual, Mt. Prospect, Illinois, 1997, Intel, http://www.intel.com/design/pentium/manuals/242816.htm, order number 242816-003.

[Larus95] J. R. Larus and E. Schnarr, ‘‘Eel: Machine-independent executable editing,’’ in *Proceeding of the SIGPLAN ’95 Conference on Programming Language Design and Implementation (PLDI)*, pp. 291-300, June 1995.

[Martin97] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, ‘‘Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture,’’ *Proceedings of the 24th Annual Internation Symposium on computer Architecture*, June 2-4, 1997, Denver, Colorado, pp. 85-97.

[Miller95] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam and T. Newhall, The Paradyn Parallel Performance Measurement Tools, IEEE Computer 28(11), (November 1995).

[Mucci99] 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/papi.html.

[Mukherjee97] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. D. Hill, J. R. Larus, and D. A. Wood, ‘‘Wisconsin Wind Tunnel II: A Fast Portable Parallel Architecture Simulator,’’ in *Workshop on Performance Analysis and Its Impact on Design (PAID)*, June 1997.

[Nutt95] Gary J. Nutt, Adam J. Griff, James E. Mankovich, and Jeffrey D. McWhirter, ‘‘Extensible Parallel Program Performance Visualization,’’ International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS’95) 1995.

[Reed94] Daniel A. Reed, Ruth. A. Aydt, Roger J. Noe, Philip C. Roth, Keith A. Shields, Bradley Schwartz, and Luis F. Tavera, ‘‘Scalable Performance Analysis: The Pablo Performance Analysis Environment,’’ Anthony Skjellum(ed), *Proceedings of the Scalable Parallel Libraries Conference*, IEEE Computer Society, pp. 104-113, October 1993.

[Rodgers99] J. B. Rodgers, R. K. Gaede, and J. H. Kulick, ‘‘IN-Tune: An In-Situ Non-Invasive Performance Tuning Tool for Multi-threaded Linux on Symmetric Multiprocessing Pentium Workstations,’’ accepted for publication in Software: Practice and Experience, 1999.

[Srivastava94] Amitabh Srivastava and Alan Eustace, ATOM: A System for Building Customized Program Analysis Tools, Research Report 94.2, DEC Western Research Laboratory, March 1994.

[Site95] R. L. Sites and R. T. Witek. Alpha AXP Architecture Reference Manual, Digital Press, Newton, Massachusett, second edition, 1995.

[Sorin98] D. J. Sorin, V. S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood, ‘‘Analytic Evaluation of Shared-Memory Systems with ILP Processors,’’ Proceedings of The 25th Annual International Symposium on Computer Architecture, June 27-July 1, 1998, Barcelona, Spain, pp. 380-391.

[Welbon94] E. H. Welbon, C. C. Chan-Nui, D. J. Shippy, and D. A. Hickset, ‘‘The POWER2 Performance Monitor,’’ IBM Journal of Research and Development, 38(5), September 1994.

[Woo95] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, ‘‘The SPLASH-2 Programs: Characterization and Methodological Considerations,’’In Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36, Santa Margherita Ligure, Italy, June 1995.

[Zagha96] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz, ‘‘Performance Analysis Using the MIPS R10000 Performance Counters,’’ in the proceedings of Super Computing 1996, November 1996.