Appeared in "Proceedings of the 9th International Parallel Processing Symposium, April 1995'

Symbolic Performance Prediction of Scalable Parallel Programs †

Mark J. Clement
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576

Michael J. Quinn
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-3202

Abstract

Recent advances in the power of parallel computers have made them attractive for solving large computational problems. Scalable parallel programs are particularly well suited to Massively Parallel Processing (MPP) machines since the number of computations can be increased to match the available number of processors. Performance tuning can be particularly difficult for these applications since it must often be performed with a smaller problem size than that targeted for eventual execution. This research develops a performance prediction methodology that addresses this problem through symbolic analysis of program source code. Algebraic manipulations can then be performed on the resulting analytical model to determine performance for scaled up applications on different hardware architectures.

1 Introduction

Scalable applications are particularly well suited to execution on Massively Parallel Processing (MPP) systems because the number of parallel operations can be scaled up as the number of processors is increased. The High-Performance Computing and Communications (HPCC) program has identified scalable applications that are important to making progress in the fields of fuel combustion, ocean modeling, ozone depletion and several other areas where sequential processing power is not sufficient [4].

Achieving acceptable performance for these applications is often difficult because of the increased complexity involved in performance analysis. The programmer must be able to account for parallel overhead with varying problem size and the numbers of processors in order to asses the quality of a problem implementation. Symbolic performance prediction provides essential feedback that should increase the efficiency of multicomputers on this important class of applications. In particular, symbolic performance prediction can be an important component of performance debugging tools, which are sorely needed for parallel application development [9].

Accurate performance prediction information can be used by programmers for performance debugging and machine selection, by compilers to guide optimizations, and by system architects to determine optimal hardware configurations for important classes of applications. This research uses symbolic performance prediction techniques to make progress in solving this important problem.

† This research was supported by Intel Foundation and NSF grants ASC-9208971 and CDA-9216172.

2 Symbolic Performance Prediction

Many of the goals of this research were motivated by meetings with a commercial MPP vendor to determine requirements for a performance analysis tool. Members of the programming tools and system architecture groups suggested the following requirements for performance prediction.

- The model should allow a user to predict performance for larger problem sizes and larger number of processors than the machine used during performance debugging. This allows smaller parallel machines or workstations to be used during program development with large MPP machines being reserved for production runs.
- One means of determining the portability of an application is to determine the sensitivity of a program to changes in critical system parameters. A performance model should allow the user to view the sensitivity to system parameters as the problem size and number of processors vary.
- Several of the MPP systems currently in production support advanced operating systems with virtual memory. An effective performance prediction model should account for paging activity as the data size grows to the point that it does not fit in physical memory.

The symbolic performance prediction model described here differs from previous work in addressing these fundamental requirements [1, 5, 7].

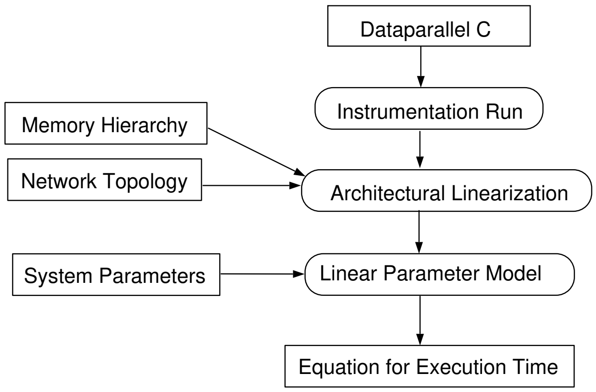

Figure 1: Block diagram for performance prediction system

Figure 1 shows a block diagram of the performance prediction system. Our implementation focuses on analysis for the Dataparallel C parallel programming language [8]. The basic constructs can be extended to other explicitly data-parallel languages, such as Fortran 90, High Performance Fortran and C*. Given Dataparallel C source code, instrumentation is inserted to determine execution characteristics that will be used to build a call graph for the application. Architectural specifications for the target machine are then passed to a linearization phase that outputs operation counts for significant application parameters. These counts are then combined with costs in time for each operation type, resulting in a symbolic equation for execution time. Since the result of this model is an equation rather than a time estimate for a given problem size, the execution time can be differentiated with respect to a given system parameter. The resulting equation can be used to determine the sensitivity of the application to changes in that parameter as the problem is scaled up.

2.1 Instrumentation Run

Instrumentation required by the performance prediction system can be inserted by a source-to-source compiler. It consists of static declarations of predefined data types and calls to a prediction library routine.

Iteration constructs are instrumented with a loop descriptor structure that specifies the symbolic name of the iteration variable along with the initialization, termination and increment expressions. Conditional code is also instrumented to determine the true ratios. Related research has shown that true ratios often remain constant as the problem size for an application is scaled up [6]. Information on the total data size accessed during each virtual processor emulation loop is used to determine the number of cache misses. The type of communication and the size of the data instance to be transferred are also accounted for in each communication block.

At the conclusion of the instrumentation run, the performance prediction library builds a call graph data structure that is used to scale the number of loop iterations and the size of each shape to the problem size. The instrumentation library code recursively descends the call graph structure to determine the number of times each block of code will be executed.

This instrumentation strategy is effective in structured programs where the iteration variables are not modified within the loop body. It is also restricted to non-recursive applications without "goto" statements. In a survey of 112 supercomputer applications from the College of Oceanographic and Atmospheric Sciences at Oregon State University, 98% of the loop constructs were amenable to this analysis strategy.

2.2 Architectural Linearization

The architectural linearization phase of symbolic performance prediction reduces complex machine characteristics to equations linear with respect to the speed of hardware subsystems. Statistical inference can then be used to automatically determine the coefficients using linear modeling techniques [3]. The major architectural features we analyze are:

- On-chip cache and page fault behavior.
- Message startup times for interprocessor communications.
- Bandwidth characteristics for different communication patterns.

References to parallel variables in virtual processor emulation loops have a highly sequential access pattern, enabling accurate prediction of the number of cache misses and page faults that will occur during the execution of a scalable application. For our analysis we assume that if the size of all data accessed during a virtual processor emulation loop is greater than the cache size, then the processor will miss in the cache for the whole data array. Otherwise there is no cache miss penalty. This applies to both on-chip cache and virtual memory. As a result of this linearization step, expressions for the number of cache misses can be derived.

Interprocessor communication can have a significant impact on the performance of a parallel application. We have found that modeling the number of message startup times necessary for a communication and the number of bytes transmitted results in an accurate estimation of the total communication cost. For each communication type, we model the number of messages that must be initiated to perform the transfer. Neighbor communications require a single message while broadcasts using a binomial tree algorithm require time logarithmic in the number of processors. The complexity of the other communication patterns for Dataparallel C has been thoroughly investigated previously [8] and is included in our model of the application. Equations for message startup cost and available bandwidth must be altered to model different topologies, but will be constant for architectures with similar communication networks.

As a result of the architectural linearization phase, expressions for operation counts are computed as a function of the problem size and the number of processors utilized.

2.3 Linear Parameter Model

Given counts for each operation and the cost for that operation on a given system, total execution time can be predicted for the application being modeled. Let

$$\mathcal{X} = \begin{bmatrix} \mathcal{X}_{Ops_0} & \mathcal{X}_{VP_0} & \mathcal{X}_{L1_0} & \mathcal{X}_{St_0} & \mathcal{X}_{Bw_0} \\ \mathcal{X}_{Ops_1} & \mathcal{X}_{VP_1} & \mathcal{X}_{L1_1} & \mathcal{X}_{St_1} & \mathcal{X}_{Bw_1} \\ \mathcal{X}_{Ops_2} & \mathcal{X}_{VP_2} & \mathcal{X}_{L1_2} & \mathcal{X}_{St_2} & \mathcal{X}_{Bw_2} \\ \mathcal{X}_{Ops_3} & \mathcal{X}_{VP_3} & \mathcal{X}_{L1_3} & \mathcal{X}_{St_3} & \mathcal{X}_{Bw_3} \end{bmatrix} \quad \text{and}$$

$$\beta = \begin{bmatrix} \beta_{Ops} \\ \beta_{VP} \\ \beta_{L1} \\ \beta_{St} \\ \beta_{Bw} \end{bmatrix}$$

where each row of $\mathcal{X}$ corresponds to counts of arithmetic operations ($\mathcal{X}_{Ops}$), virtual processor emulation loops ($\mathcal{X}_{VP}$), level one cache misses ($\mathcal{X}_{L1}$), messages sent ($\mathcal{X}_{St}$) and the number of bytes transmitted through a communication channel on a processor ($\mathcal{X}_{Bw}$) for particular values of N and P. This array can be built automatically given the equations resulting from the linearization phase by the Maple [2] symbolic computation system. The cost vector β represents the cost in seconds for each system parameter. The predicted execution time vector $T = \beta\mathcal{X}$ contains the time in seconds for the algorithm to execute for particular values of N and P. Multivariate techniques can be used to obtain these costs automatically, given a statistically significant number of experimental runs with different problem sizes and numbers of processors [3].

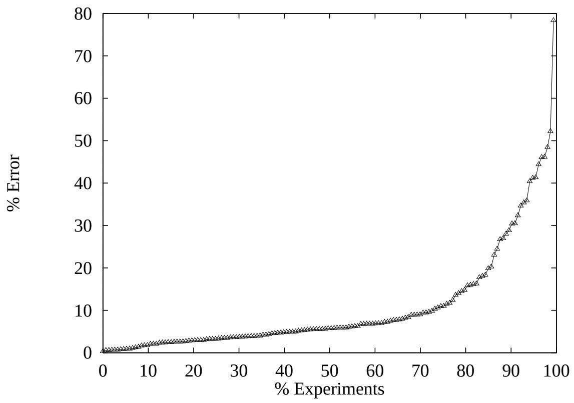

Figure 2: Percent error for 153 experimental runs of applications in the validation suite.

3 Experimental Results

Several sets of experiments were performed to validate the symbolic performance prediction system. We examined Gaussian elimination, matrix multiplication and ocean modeling algorithms on the nCUBE 3200, iPSC/860 and iWarp processors in order to determine prediction error across different application classes. For our error analysis we have used the classical error computation method: $Error = (Predicted - Actual)/Actual$. The error contour shown in Figure 2 indicates that over 80% of the experimental runs achieve less than 20% error. Larger error values are observed for extremely short total execution times where the time spent in starting the computation is dominant.

4 Analysis Using Performance Prediction Results

Performance analysis is used by system architects in order to improve the performance of next generation machines. It can also be used by programmers in performance debugging and by customers to assist in making procurement decisions. The symbolic performance prediction system described here performs analysis on scalable applications where other analysis techniques cannot be used. The following examples show how this performance prediction system has been used to analyze performance as a problem is scaled up.

Figure 3 compares performance for Gaussian elimination on the nCUBE 3200 and a network of SPARC workstations. For small numbers of processors the workstations are significantly faster than the multicomputer. As the number of processors and problem

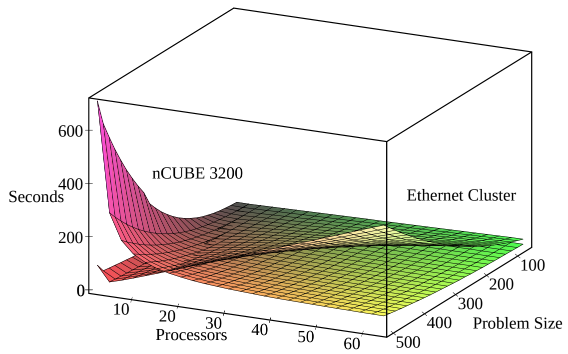

Figure 3: Execution time plots comparing the nCUBE 3200 with a network of Ethernet connected workstations.

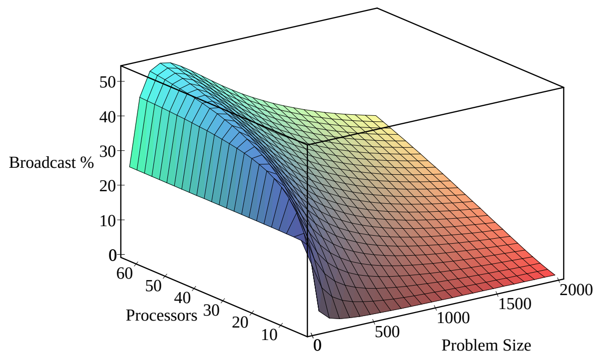

Figure 4: Percentage of execution time spent in broadcasting the pivot row for Gaussian elimination on an nCUBE 3200.

size increases, the bandwidth limitations of Ethernet reduce the performance of a workstation cluster. The ability to predict performance on differing architectures is important in selecting a machine configuration for an application.

Another fundamental use of performance prediction results is in the area of performance debugging. Figure 4 illustrates the percentage of time spent in broadcasting the pivot row for the Gaussian elimination application on an nCUBE 3200. For extremely small problem sizes, the execution time is dominated by message startup costs for the communications. Since all communications in the application grow linearly with P, the percentage of time spent in this basic block is constant. As the problem size is scaled up the fraction of time grows logarithmically with P due to the number of messages required for the binomial tree broadcast algorithm. For larger problem sizes the communication ratio grows linearly with the number of processors.

The ability to predict the sensitivity of an algorithm to changes in system parameters is critical to determining its portability. As architects are able to predict the sensitivity of applications to changes in system parameters the efficiency of parallel hardware should increase. Since the result of this performance prediction system is a symbolic expression for execution time, the equation can be differentiated with respect to critical system parameters in order to view the effect modifications will have on performance as the problem size is scaled up. In Figure 5, the execution time was differentiated with respect to message startup cost. The vertical scale shows the increased execution time in seconds for each increase of 10μsec in message startup

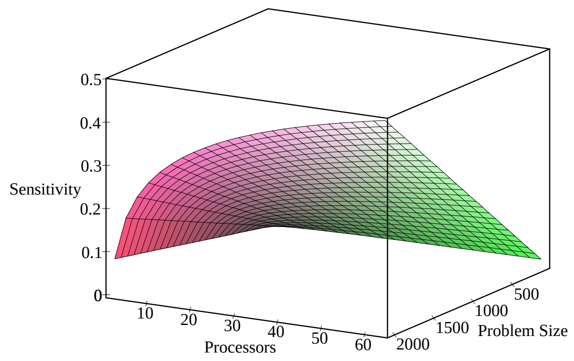

Figure 5: Sensitivity to changes in message startup cost for Gaussian elimination as a function of problem size and number of processors.

cost. The sensitivity increases linearly with problem size since a constantly growing number of communications must be performed as problem size increases. The sensitivity grows logarithmically as the number of processors increases due to the complexity of the binomial tree broadcast algorithm.

Recent advances in the speed of workstations have motivated several systems vendors to introduce workstation clusters as parallel computing platforms. Performance prediction information can be used to determine the cost optimal interconnection network for such systems. In order to examine Gaussian elimination from a price/performance perspective, we obtained quotes for high speed network connections using switched Ethernet, FDDI and ATM technology.

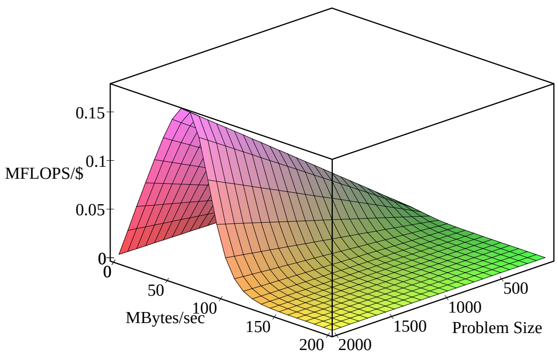

Figure 6: Predicted MFLOPS/$ for Gaussian elimination on 8 workstations as network bandwidth varies.

We then fit a curve to these points and derived an expression for dollar cost as a function of network bandwidth. Figure 6 indicates that a network bandwidth of 50Mbytes/sec is near the cost optimal point for a network of 8 workstations.

5 Conclusions

The symbolic performance prediction methodology developed here allows performance analysis to be performed on scalable parallel applications. This important class of programs imposes different requirements on an analysis system than traditional, constant size problems do. We have found that the use of symbolic information from the source code improves the accuracy of performance prediction over other sampling methods.

Further work is currently being performed to improve the accuracy and utility of the system. Multivariate statistical analysis methods can be applied to improve the accuracy of the cost vector and to provide a prediction interval for projected execution times. Research is being conducted to determine if the elapsed time of the instrumentation run can be used to improve the accuracy of symbolic analysis for systems utilizing vector processors. Several graphical tools have also been proposed to illustrate the percentage of time being spent in each basic block as the problem size and number of processors vary.

Symbolic performance prediction has been shown to be effective in meeting the requirements suggested for an analytical model. The use of this type of performance analysis system by system architects and programmers should increase the efficiency of MPP systems in general and scalable applications in particular.

References

[1] V. Balasunderam, G. Fox, K. Kennedy, and U. Kremer. A static performance estimator to guide data partitioning decisions. *SIGPLAN Notices*, 26(7):213 – 223, July 1991.

[2] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual*. Springer-Verlag, New York, 1991.

[3] M. J. Clement and M. J. Quinn. Multivariate statistical techniques for parallel performance prediction. In *Proceedings of the 28th Hawaii International Conference on System Sciences, HICSS-28*, January 3-6 1995.

[4] Committee on Physical, Mathematical, and Engineering Sciences Federal Coordinating Council for Science, Engineering, and Technology. *Grand Challenges 1993: High Performance Computing and Communications*. National Science Foundation, Washington, D.C., 1993.

[5] M. E. Crovella and T. J. LeBlanc. The search for lost cycles: A new approach to parallel program performance evaluation. In *Proceedings of Supercomputing '94*, November 1994.

[6] T. Fahringer. *Automatic Performance Prediction for Parallel Programs on Massively Parallel Computers*. PhD thesis, University of Vienna, 1993.

[7] A. J. Goldberg and J. L. Hennessy. Mtool: An integrated system for performance debugging shared memory multiprocessor applications. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):28–40, January 1993.

[8] P. J. Hatcher and M. J. Quinn. *Data-Parallel Programming on MIMD Computers*. The MIT Press, Cambridge, Massachusetts, 1991.

[9] P. H. Smith. System software and tools for high performance computing environments. Report on the Findings of the Pasadena Workshop, April 14-16 1992.