Appeared in "Proceedings of the 28th Hawaii International Conference on System Sciences January, 1995'

Multivariate Statistical Techniques for Parallel Performance Prediction †

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

Performance prediction can play an important role in improving the efficiency of multicomputers in executing scalable parallel applications. An accurate model of program execution time must include detailed algorithmic and architectural characterizations. The exact values for critical model parameters such as message latency and cache miss penalty can often be difficult to determine. This research uses multivariate data analysis to estimate the values of these coefficients in an analytical model. Representing the coefficients as random variables with a specified mean and variance improves the utility of a performance model. Confidence intervals for predicted execution time can be generated using the standard error values for model parameters. Improvements in the model can also be made by investigating the cause of large variance values for a particular architecture.

1 Introduction

Although the peak performance of distributed memory multicomputers continues to increase, their acceptance and use is still limited to a small part of the potential market. One reason for the underutilization of these machines is that they are more difficult to program and use than traditional supercomputers. The programmer must often study the architecture of the machine and become familiar with detailed compiler transformations in order to make appropriate design decisions. An analytical model can simplify the process of program development by providing essential information about the relative performance of alternate implementations. System architects can also use predicted execution times to evaluate the relative merits of future hardware implementations.

Applications with scalable problem sizes are particularly well suited to execution on Massively Parallel Processing (MPP) systems, because the number of parallel operations can be increased as a larger number of processors are used. The High-Performance Computing and Communications (HPCC) program has identified scalable applications, which 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 [7].

Performance prediction is particularly important for scalable applications because they are not amenable to traditional performance debugging techniques. Given a sample application that runs for one hour on 1000 processors, it is impractical to gather trace data or to perform a simulation of the program to gather performance information. If we assume that accurate trace-based analysis requires 2 Mbytes/sec of data [14] then our sample application would generate 6 Tbytes of trace data. This data volume is clearly impractical, even if the data could be written in real time without significant performance perturbation. Simulation techniques would require in excess of 1000 hours to emulate the execution of 1000 processors for a one hour job. This would exceed the maximum acceptable overhead for performance debugging. With multiple MPP systems becoming available to a single researcher, performance prediction is also an important tool in matching the appropriate hardware platform to a specific application.

This research uses an instrumented run of a target application to build an analytical model which is adaptable to machine specific characteristics of a given multicomputer platform. In order to calculate coefficients for this model, the characteristics of the hardware, operating system and compiler must all be determined. Although lower bounds on specific hardware parameters can be obtained from technical specifications, exact access times from user programs can be difficult to determine. This is especially true for workstation clusters where network access times can vary depending on network traffic and system load. For many of these variables, however, the coefficients

† This research was supported by Intel Foundation and NSF grant ASC-9208971.

can be approximated by random variables with a normal distribution.

Multivariate data analysis techniques are used widely in the behavioral and biological sciences to analyze the effect of several variables on a measurable outcome. Software packages make it possible to analyze large quantities of complex data with relative ease. Using statistical techniques to estimate the coefficients for an analytical model has several advantages:

- Statistical packages provide standard error values for each of the prediction variables. These confidence intervals allow us to specify a confidence interval as well as an expected value for predicted performance on a target architecture.

- The model can be fit in an automated and structured way using real applications similar to the expected load for a parallel system.

- Standard information available from statistical software packages assess the correlation of the model to experimental data. This information allows us to tune the model in order to reduce prediction error. Extraneous variables can also be eliminated when a model parameter is statistically indistinguishable from zero. Models with smaller numbers of variables can often achieve higher accuracy.

Multivariate statistical analysis often requires a larger number of samples than other fitting methods. Scalable applications are good candidates for this environment because multiple samples can be obtained from a single program using different problem sizes. Our results indicate that a reasonably accurate fit can be obtained from a limited number of applications. Statistical methods for finding coefficients are also applicable to non-scalable applications if a larger number of sample applications are available.

In Section 2 we describe the symbolic performance model used to gather algorithmic information for our statistical analysis. Section 3 applies multivariate analysis to the problem of determining coefficients for the model. Experimental results are detailed in Section 4 and related research into performance prediction is examined in Section 5.

2 Symbolic Performance Prediction

The symbolic performance prediction system we describe here is designed to make use of statistical analysis in determining model coefficients. Multivariate techniques are an essential part of the overall modeling system.

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 specifications for a performance prediction system.

- 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 solving computational problems. It also speeds up instrumentation runs of a program since smaller problem sizes can be used during performance debugging.

- 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. The validity of the model can also be determined through analyzing the sensitivity of the model to a particular parameter and the confidence level for that parameter.

- 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 meets these requirements as well as allowing the user to determine the relative contribution of each basic block in application code.

Figure 1 shows a block diagram of the performance prediction system. Our implementation focuses on analysis for the Dataparallel C parallel programming language [13]. The basic constructs can be extended to other explicitly data-parallel languages, such as Fortran 90, High Performance Fortran and C*.. We have found that programs written in high level parallel languages enable more accurate and complete performance analysis techniques than code written in an imperative language with message passing. This occurs because the communication patterns are more

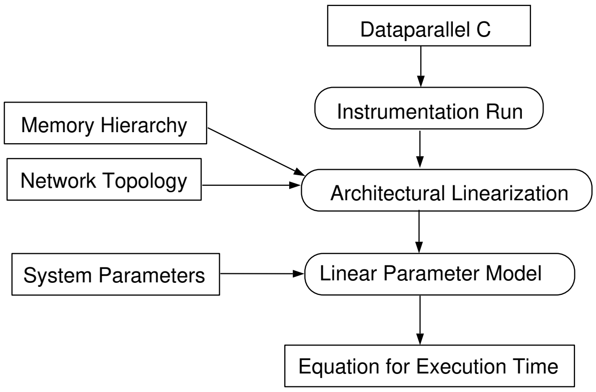

Figure 1: Block diagram for performance prediction system

predictable and recognizable. The data access patterns can also be determined more easily for programs written in a high level parallel language.

Given Dataparallel C source code, instrumentation code is inserted to determine execution characteristics which will be used to build a call graph for the application. Architecture specifications for the target machine are then passed to a linearization phase which outputs operation counts for significant system 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. Execution time for the instrumentation run is used to refine the estimate for the number of computations involved in the application.

Dataparallel C replicates scalar values on all processors and distributes parallel variables across the nodes of the system. Total data volume for scalable applications will be dominated by parallel variables. The instrumentation code accounts for the size of parallel variables through declaring a shape descriptor structure which specifies the number of dimensions in a shape and the number of positions in each dimension.

Iteration constructs are instrumented with a loop descriptor structure which specifies the symbolic name of the iteration variable along with the initialization, termination and increment expressions. This symbolic information will be used to analyze complex iteration constructs. 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 [10]. Virtual processor emulation loops are instrumented with shape instance descriptor structures which are organized in a linked list. Information on the data size accessed during each virtual processor loop will be 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 which is used to scale the number of loop iterations and the size of each shape to the problem size. A sample call graph is shown in Figure 2. The instrumentation library code recursively descends the call graph and for each loop construct the following attempts are made to determine the complexity of the loop.

1. The most accurate results occur when the number of loop iterations can be determined from symbolic information. A search is made beginning from the parent node of the loop under analysis to the top of the call graph tree. If the initialization or termination values for the loop are iteration variables for an enclosing loop, then that symbolic value is used in subsequent analysis.

2. If a symbolic value cannot be found, then a simplified fit is attempted to determine if the number of iterations found in the instrumentation run is an even multiple of the problem size.

3. If the initialization or termination expression cannot be scaled to problem size, then it is assumed to be constant.

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 con-

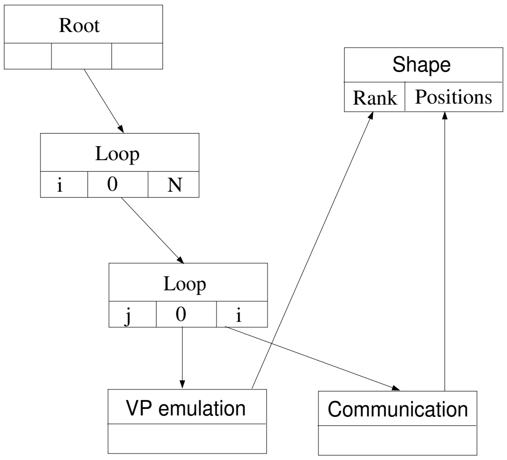

Figure 2: Call Graph Data Structure

structs were amenable to this analysis strategy. Parallel variable descriptor structures are also scaled to the problem size in a similar manner. Since the call graph structure is constructed symbolically, program complexities which can not be analyzed computationally can be entered by a programmer in symbolic form if additional accuracy is required. This manual specification of loop complexity has not been necessary for any of the applications we have examined.

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 determine the coefficients using linear modeling techniques. The major architectural features we analyze here 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 which 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 which 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 [13] 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 [4] 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 given a statistically significant number of experimental runs with different problem sizes and numbers of processors.

3 Statistical Approximation of System Parameters

Multivariate statistics refers to a group of inferential techniques that have been developed to handle situations where sets of variables are involved as predictors of performance [12]. In classical scientific experiments, an effort is made to eliminate all but one causal factor through experimental control. The variables in our analytical model are difficult to isolate, and more complex methods are needed to estimate the value of model coefficients. Statistical software packages allow large quantities of multivariate data to be analyzed with relative ease. The use of the S-PLUS [19] multivariate statistical package allows us to quantify the contribution of each system parameter to execution time and provides confidence intervals on the resultant predictions [3].

The performance prediction model developed in this research was designed to create a linear model with respect to the important system characteristics we have identified. The following assumptions have been made in order to apply statistical techniques to this model:

- Both the predictor variables and the model errors are statistically independent. As the number of mathematical operations performed by a parallel application increases there will not necessarily be a corresponding increase in the number of cache misses or communications performed by the application. The independence of these variables is important to the application of linear regression methods. This assumption is approximately true within runs of a single application as the problem size is varied. The assertion that the $\mathcal{X}$ values are independent is even stronger when multiple applications are included in the set of programs used to fit the model.
- The $\mathcal{X}$ matrix is able to characterize important performance indicators equally well from application to application. Given an application $\mathcal{A}$ which we would like to make predictions for, and a set of applications $\mathcal{S}$ used in fitting the model, where $\mathcal{A} \notin \mathcal{S}$, we assume that the $\mathcal{X}$ matrix represents algorithmic characteristics equally well for $\mathcal{A}$ as for the other applications in $\mathcal{S}$. This assertion is particularly strong among programs addressing a single computational problem. A supercomputing site may have several researchers developing fluid dynamics applications, and a model fit to existing fluid flow programs will generalize well to other applications in this class. Our experience has shown that as long as $\mathcal{S}$ contains samples from several different types of parallel applications the fit is quite good for other programs.
- The true relationship between the predicted time and model variables is linear, and the algorithmic measurements are accurate. Inaccuracies in $\mathcal{X}$ values can degrade the validity of regression results. Our results indicate that accurate operation counts can be obtained through an instrumented run of a scaled down version of an application.
- Errors are normally distributed with zero mean and a constant standard deviation. The shape of the quantile-quantile curve can be used to determine the correlation of residuals to a normal distribution. The quantile plot of Figure 3 indicates that the residuals for the Meiko CS-2 have slightly longer tails than a normal distribution.

Multiple linear regression techniques model a numeric response variable, y, by a linear combination of p predictor variables x_j for $j = 1, \ldots, p$. The predicted values are the sum of coefficients β_j multiplied by the corresponding x_j.

$$y = \beta_0 + \beta_1\mathcal{X}_1 + \cdots + \beta_p\mathcal{X}_p$$

Linear *least-squares* models (LSQ) estimate the coefficients to minimize the squared sum of errors between predicted and experimental values. If the response and predictors corresponding to the ith of n observations are $y_i, x_{i1}, x_{i2}, \cdots, x_{ip}$, then the fitting criterion chooses the β_j to minimize $\sum_{i=1}^{n}(y_i - (\sum_{j=1}^{p}\beta_j x_{ij}))^2$ [3].

One side effect of using the LSQ criterion is that outliers (experimental values with a large error) tend to have a big effect on the derivation of β values. From one perspective, this is reasonable, because the

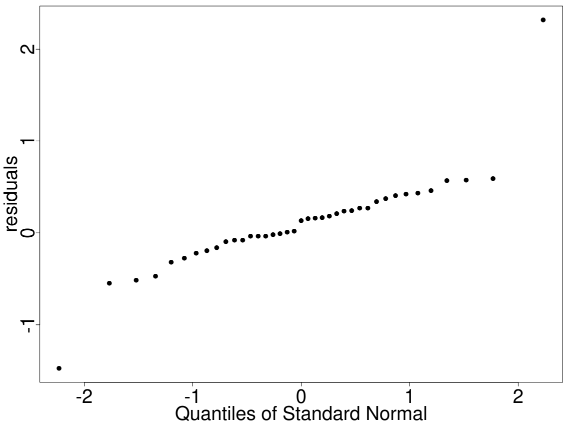

Figure 3: Quantile plot of model errors for experimental data derived from 5 example programs on the Meiko CS-2 vs. a normal distribution of residuals.

seriousness of an error in prediction goes up much faster than linearly with the magnitude of the error. Although we would like to have all predicted values within a certain percentage error of the experimental value, it is more important to estimate values correctly when the execution time is in thousands of seconds than it is with millisecond run times. In order to minimize the effects of erroneous measurement we have manually removed outliers which were suspect. The technique of "ridge regression" could also be used which would allow some bias in the estimated β values in exchange for a potentially large decrease in variability in the presence of "wild" observations [12].

Our analytical performance model uses counts of critical operations generated from the linearization module as predictor variables for multivariate analysis. The β values generated by the statistical package are estimates of the actual values for system parameters in a particular parallel architecture. A number of real data-parallel applications are run on an existing parallel machine in order to fit the model. At this point the prediction system can make predictions, with confidence intervals, for other parallel applications on the selected platform.

4 Experimental Results

Several sets of experiments are described here to validate the performance prediction system using multivariate analysis. Instrumentation is added to the intermediate code of several significant scientific applications. The applications are then run with an extremely small problem size in order to produce the analytical model. This instrumentation run takes less than 10 seconds for each application. The output from the instrumentation run is then reduced by the Maple [4] program into expressions which are functions of the fundamental architectural features of a particular parallel implementation. The model is fit using actual runs on the Intel iPSC/860, nCUBE 3200 and Meiko CS-2 multicomputers. We use a "Systematic Sampling" approach to selecting experimental values for the fitting process [18]. The S-PLUS [19] software package is then used to predict performance for two ocean modeling applications. We also discuss several ways in which this prediction system can be used to analyze performance.

System Parameter	Value (μsec)	Std. Err.	Significance
β_{Ops}	0.6001	0.0044	0.00000000
β_{VP}	15.2648	5.1386	0.00000000
β_{St}	367.8870	757.9475	0.02351422
β_{Bw}	2.3690	0.5341	0.00004550

Table 1: Output of statistical parameters for the nCUBE 3200 multicomputer.

4.1 Multivariate Model Analysis

Our initial experiments were run with the nCUBE 3200 multicomputer. The model was fit using three applications (Gaussian elimination, matrix multiplication and the "shallow-water" model) with several different problem sizes for a total of 58 experimental runs. The resulting β values are shown in Table 1.

The coefficient of determination (Multiple R-squared term) for this regression is 0.9978 indicating that over 99% of the total variation in the response is accounted for by variation in the fitted values [3]. Values in Table 1 are given in microseconds. Given the system clock cycle time of 125$n$sec, the β_{Op} value indicates that it takes approximately 5 clock ticks for an average operation. The β_{VP} value suggests that it takes approximately 120 cycles to set up a virtual processor emulation loop. The startup time of 367μsec is similar to that found in previous research on the nCUBE [6].

The standard error column in Table 1 is an estimate of how much the regression coefficient β will vary from sample to sample. If multiple samples of the same size were taken from the same population and used to calculate the regression equation, this would be an estimate of how much the regression coefficient

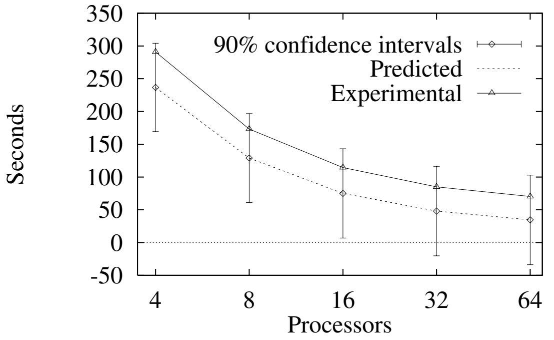

Figure 4: Experimental and predicted execution time in seconds for the "ocean circulation model" on the nCUBE 3200 multicomputer with 128 segments in the east-west direction.

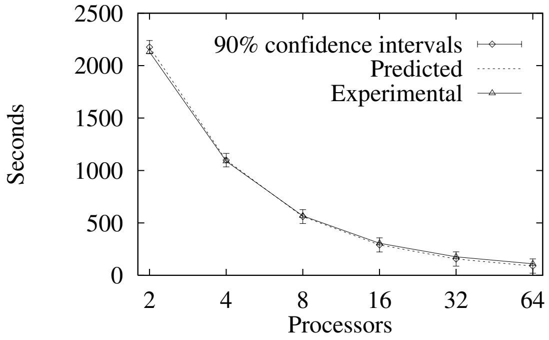

Figure 5: Experimental and predicted execution time in seconds for the "ocean circulation model" on the nCUBE 3200 multicomputer with 640 segments in the east-west direction.

would vary from sample to sample [15]. The large standard error value for the message startup cost indicates that a more detailed model needs to be investigated for this parameter. The "Significance" column describes the result of an $\mathcal{F}$-test to determine the probability that the variance accounted for by the coefficient could come from a F distribution. Small values indicate that the variable is important in explaining variance. All of the entries in this column indicate that the coefficients are highly significant in accounting for variance in the experimental data.

Figure 4 and Figure 5 illustrate predicted output for the "ocean circulation model" on the nCUBE 3200. The program models wind-driven circulation in a density-stratified ocean [13]. The problem scales up by increasing the number of segments modeled in the east-west direction. The vertical bars join the upper and lower twice-standard-error points, meant to represent approximately 90% confidence intervals for the mean response. Since the standard error value for communications is higher than that for computations, the confidence interval in Figure 4 is larger than the predicted interval in Figure 5. This occurs because communication costs make up a higher fraction of total execution time for the smaller problem in Figure 4.

Figure 6 examines predicted output for the "shallow water model" application on the Intel iPSC/860. The National Center for Atmospheric Research has developed this application for use in benchmarking the performance of MPP systems. The program solves a set of nonlinear shallow-water equations in two horizontal dimensions [13]. For this experiment we rely exclusively on previous experimental data for execution times in order to fit the model. The instrumentation run for the applications was performed on a

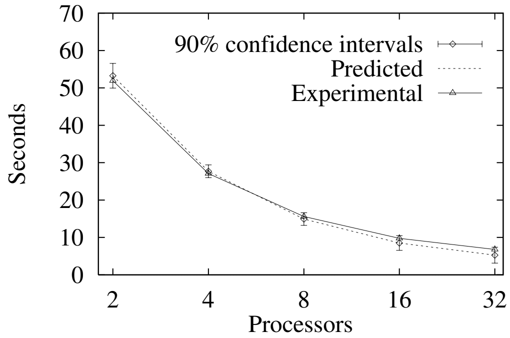

Figure 6: Predicted and experimental values for the "shallow water" model with a 64X64 grid and 1200 iterations for an Intel iPSC/860 multicomputer.

single processor workstation and yet accurate results were still obtained.

The Meiko CS-2 multicomputer consists of SPARC processors connected in an Omega network configuration [16]. Each node is equipped with two vector processors to improve floating point performance. A copy of the multi-user Solaris operating system executes on each node increasing the variability of successive runs of the same program on the machine. Figure 7 and Figure 8 show the predicted results obtained for the "ocean circulation model." The experimental data has a much larger number of outliers than were found on the other two machines. Some of this variability can be attributed to the multi-user nature of the machine. The current implementation of Dataparallel C on the Meiko relies on libraries written for the Intel iPSC/860 which use the NX message passing interface provided on the Meiko. This extra level of software

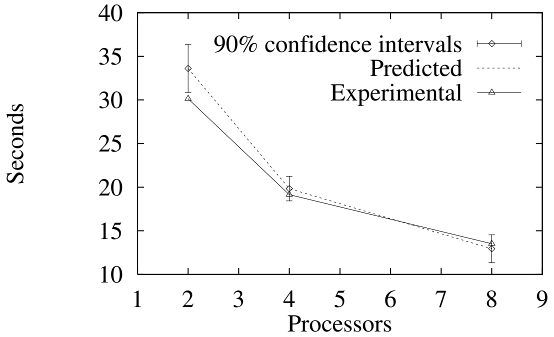

Figure 7: Experimental and predicted execution time in seconds for the "ocean circulation model" on the Meiko CS-2 multicomputer with 640 segments in the east-west direction.

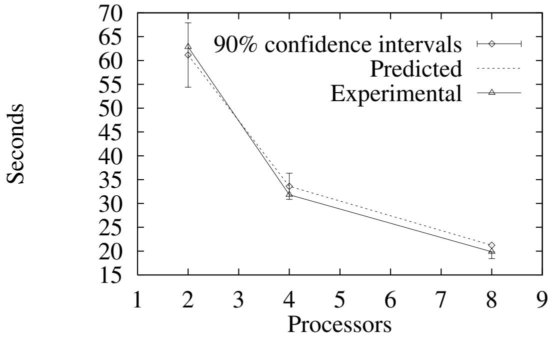

Figure 8: Experimental and predicted execution time in seconds for the "ocean circulation model" on the Meiko CS-2 multicomputer with 1280 segments in the east-west direction.

indirection may also account for some of the inaccuracies in the model. Future work will focus on adapting this modeling technique to networks of workstations where high levels of variability exist in the message passing latency. This work on the Meiko is a first step in that direction.

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.

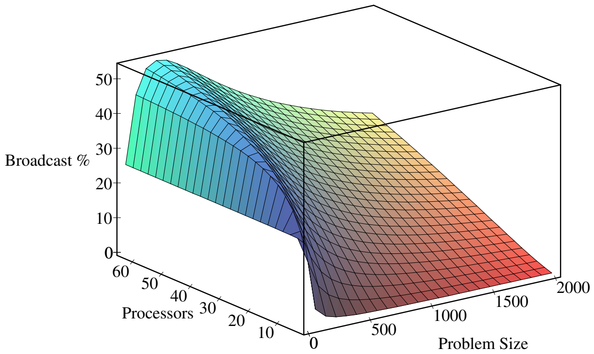

Figure 9: Percentage of execution time spent in broadcasting the pivot row for Gaussian elimination on an nCUBE 3200.

4.2 Analysis Using Performance Prediction Results

The performance of parallel applications is affected by many factors which need not be considered with sequential programs. Consequently, performance analysis for parallel applications much take more of a multidimensional approach [17]. The following examples illustrate the utility of this symbolic performance model in analyzing the impact of several effects on a scalable application.

A major use of performance prediction results is in the area of performance debugging. Figure 9 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

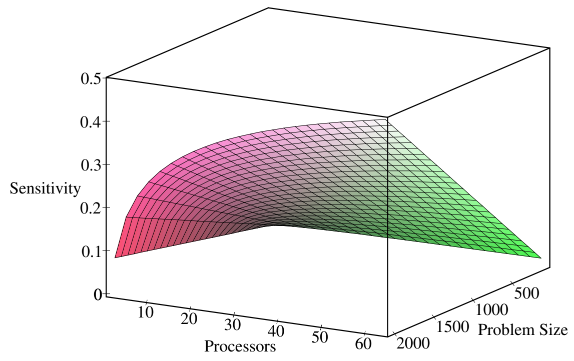

Figure 10: Sensitivity to changes in message startup cost as a function of problem size and number of processors.

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 10, 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 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.

5 Related Work

Several different approaches have been taken in predicting performance for parallel systems. To our knowledge this research is the first to employ multivariate statistical analysis in finding coefficients for parallel performance models. The linearization phase of our algorithm produces a model which can be used to analyze the impact of system parameters on applications.

Several analysis techniques have been investigated which incorporate information from the source code or from an instrumentation run of a target application. These methods provide accurate prediction results for applications with a fixed problem size and allow a programmer to view the effects of modifications to an algorithm and implementation [1, 2, 5, 8, 11]. These methods combine effects from all basic blocks in an application into a total count of the number of critical operations performed. The cost of these operations is then estimated using a training set of benchmark programs, or by using specifications provided by system manufacturers. The statistical techniques described in this paper could also be applied to these systems.

Research into *lost cycle analysis* has attempts to fit a curve to the number of communications and computations using multiple runs of an application [9]. Several runs are made to determine the coefficient for each variable in the model and the resultant equations are used to predict performance on scalable applications. The multivariate techniques described here should be particularly applicable to this model because of the large number of samples which are already being obtained.

The combination of symbolic performance prediction coupled with the application of statistical analysis is unique to this research.

6 Conclusions

MPP systems users, programmers and designers are all interested in performance analysis, since their goal is to achieve the highest possible performance at the lowest cost. Multivariate data analysis techniques simplify the performance prediction process by deriving system parameters from experimental runs of sample applications. The utility of prediction data is also increased as users are able to evaluate the confidence interval for estimated execution time. Improvements to the underlying analytical model can be made through focusing on predictor variables which have a large variance in the statistical analysis. Multivariate statistical techniques improve the utility and accuracy of performance prediction models. The use of performance prediction systems by system architects and programmers should increase the efficiency of MPP systems in general and scalable applications in particular.

References

[1] M. Annaratone and R. Ruhl. Balancing interprocessor communication and computation on torus-connected multicomputers running compiler-parallelized code. In *Proceedings SHPCC 92*, pages 358–365, March 1992.

[2] 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.

[3] J. M. Chambers and T. J. Hastie. *Statistical Models in S.* Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California, 1992.

[4] 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.

[5] M. J. Clement and M. J. Quinn. Analytical performance prediction on multicomputers. In *Proceedings of Supercomputing '93*, pages 886–905, November 1993.

[6] M. J. Clement, M. J. Quinn, and B. Baxter. Medium grain size applications on distributed memory multicomputers. Technical Report 93-80-13, Department of Computer Science, Oregon State University, September 1993.

[7] 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.

[8] M. Crovella, R. Vianchini, T. LeBlanc, E. Markatos, and R. Wisniewski. Using communication-to-computation ratio in parallel program design and performance prediction. In *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 238–245, December 1992.

[9] M. E. Crovella and T. J. LeBlanc. The search for lost cycles: A new approach to parallel program performance evaluation. Technical Report 479, Computer Science Department, University of Rochester, Dec. 1993.

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

[11] T. Fahringer and H. P. Zima. A static parameter based performance prediction tool for parallel programs. Technical Report ACPC/TR 93-1, University of Vienna Department of Computer Science, January 1993.

[12] R. J. Harris. *A Primer of Multivariate Statistics.* Academic Press Inc., New York, 1985.

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

[14] J. K. Hollingsworth, B. P. Miller, and J. Cargille. Dynamic program instrumentation for scalable performance tools. In *Scalable High Performance Computing Conference, Knoxville, Tennessee*, May 1994.

[15] J. Joseph F. Hair, R. E. Anderson, and R. L. Tatham. *Multivariate Data Analysis with Readings.* Macmillan Publishing Company, New York, 1987.

[16] Meiko World Incorporated. *Computing Surface 2 Overview Documentation Set*, 1993.

[17] C. M. Pancake. Software support for parallel computing: Where are we headed? *Communications of the ACM*, 34(11):53–64, November 1991.

[18] M. R. Sampford. *An Introduction to Sampling Theory.* Oliver and Boyd Ltd., Tweeddale Court, Edinburgh, 1962.

[19] Statistical Sciences Inc. *S-PLUS Users Manual*, September 1991.