Automated Performance Prediction for Scalable Parallel Computing †

Mark J. Clement
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
clement@cs.byu.edu

Michael J. Quinn
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-3202
quinn@cs.orst.edu

March 6, 1997

Abstract

Performance prediction is necessary in order to deal with multi-dimensional performance effects on parallel systems. The compiler-generated analytical model developed in this paper accounts for the effects of cache behavior, CPU execution time and message passing overhead for real programs written in high level data-parallel languages. The performance prediction technique is shown to be effective in analyzing several non-trivial data-parallel applications as the problem size and number of processors vary. We leverage technology from the Maple symbolic manipulation system and the S-PLUS statistical package in order to present users with critical performance information necessary for performance debugging, architectural enhancement and procurement of parallel systems. The usability of these results is improved through specifying confidence intervals as well as predicted execution times for data-parallel applications.

1 Introduction

Several implementations of "Grand Challenge" problems require far more processing power than any sequential processor can deliver [7] . Although significant progress is being made in the speed of microprocessor based computers, there is nearly universal agreement that continuing increases in computational power for large scientific applications will require parallel processing.

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

Massively Parallel Processing (MPP) systems promise to provide continued increases in performance by employing thousands of computing elements in the solution of a single problem. These distributed memory machines can be assembled using the fastest microprocessors with relatively inexpensive communications networks. The actual performance achieved on these machines for real applications, however, can be disappointing. The causes of inefficiency can often be traced to unbalanced hardware systems and inefficient algorithmic implementations.

Several problems have been noted in the parallel processing environment that cause reported performance results to be much poorer than would be expected when actual programs are developed to run on multicomputers:

- **Performance Debugging**

  It is difficult for a programmer to predict the impact that system parameters will have on parallel code. As a result, software developers are frequently unable to evaluate different algorithmic implementations in order to arrive at an optimal solution. There is also little information available to indicate which parallel architecture, if any, will execute a given program in an efficient manner.

- **Architectural Improvement**

  Parallel system designers are handicapped by the lack of information on the effect changing system parameters will have on the performance of actual parallel programs. As a result, systems may be designed with such a high level of imbalance that few parallel programs will be able to execute in an efficient manner.

- **Machine Selection**

  The high performance computing environment has changed significantly with the advent of high speed workstations. Much of the work that was previously performed on large centralized computers is now done on desktop workstations. Clusters of workstations are often available for larger jobs with Massively Parallel Processing (MPP) machines being reserved for the most demanding applications. Selecting the appropriate computing platform to achieve efficient execution can be an important part of program development.

Solving the problems of performance debugging, architectural tuning and machine selection is a critical step in improving the efficiency of multicomputers. The performance of parallel systems is influenced by multiple factors related to the structure of applications and to their ability to exploit the parallel features of a particular system. As a result of the multidimensional nature of this environment, users must be able to account for and visualize the effects of multiple variables on system performance [16]. Without an accurate model that takes all of these factors into account, it is difficult to determine what changes are needed to improve the performance of multicomputers. This multi-dimensional nature of distributed memory architectures makes sophisticated performance tools necessary in order to effectively utilize these systems [23, 24]. Performance prediction tools have been identified as an important technology in achieving the solution to grand challenge problems [7].

Although predicted execution times can be useful in improving the performance of multicomputers, their value can be limited unless there is some way of determining the accuracy of the prediction. By utilizing multivariate statistical techniques, this research specifies a confidence interval for predictions, indicating the expected range of values. This estimate of accuracy is particularly important for applications with variable problem size which we will be focusing on in this paper.

Previous research has investigated models which assume a statistical characterization of application programs [25]. These methods cannot provide accurate estimates of performance for real applications. Simulation based systems [21] require excessive time to arrive at predicted values. Although task graph representations [17] of algorithms expose many of the details of parallel execution, rewriting an application in another language may impose an unreasonable requirement on a developer. Static performance prediction techniques [12, 1, 14] do not use any execution of the application in the prediction process. They are often unable to estimate execution time for applications with variable problem sizes. More recent work with the Vienna Fortran language has shown that dynamic techniques combined with static analysis can produce accurate results [11]. This research conducted by Thomas Fahringer along with his recent book [10] demonstrate that performance prediction can be used with languages other than Dataparallel C. Our research extends this work through using symbolic equations and through statistical analysis of system parameters.

The isoefficiency metric [20] deals effectively with problem size scaling, but the asymptotic equations generated are not sufficiently detailed to generate exact execution times. Although multiple runs of an application with different problem sizes can be used to determine the constants for these equations [8], this curve fitting technique may be impractical for some users of performance prediction information.

The dynamic performance model developed here uses a single instrumentation run of the program with a reduced problem size to generate an equation for execution time which includes the contributions of each basic block in a program. Previous research has shown that compile time analysis can be used to characterize loop iterations for several static applications [4, 14], but for many programming constructs a sample execution of the application will be required. With compiler inserted instrumentation code, little effort is required from the programmer. The equation generated by the performance prediction library can be differentiated to determine the sensitivity of an application to changes in the hardware architecture. Since the instrumentation run can be performed on a single processor workstation, large MPP systems need not be involved in the performance prediction system.

In this paper we first describe the class of scalable algorithms which we focus on in this research. We then detail the performance modeling system implementation. Multiple regression is then examined as a means for determining the coefficients for this model. Several examples are then given to show the utility of this approach in examining performance issues along with conclusions and future work.

2 Scalable Algorithms

Scalable, data parallel applications are an important sub-class of problems which can be solved on multicomputers. It has been estimated that 90% of scientific and engineering applications are data parallel in nature [13]. Many of these programs can be scaled up by varying the number and size of data elements in order to improve accuracy and extend results.

Scalable algorithms are able to utilize increasing numbers of processing elements efficiently. The number of processors that a fixed problem size application can effectively use is limited by the fraction of the program which is sequential in nature. This limitation does not apply to scalable applications since the ratio of parallel to sequential code can be increased as more processors are added to the problem [15].

The complexity of manual performance analysis for scalable applications is significantly greater than that found on parallel programs with constant problem size. This research uses dynamic performance prediction techniques to make progress in solving this important problem.

3 Dynamic Performance Prediction

The direction for this research was 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 a 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.

- In order to enable performance debugging, a performance model must allow the user to determine the relative contribution of each basic block in an application program.

Figure 1 shows a block diagram of the dynamic performance prediction system. Our implementation focuses on analysis for the Dataparallel C programming language [19]. The basic constructs can be extended to other explicitly data-parallel languages. These prediction techniques have also been successfully applied to data parallel PVM applications [27, 6]. Given Dataparallel C source code, instrumentation code is inserted to gather execution statistics which will be used to build a call graph for the application. Architectural 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. We can use this methodology because of the SIMD execution model of Dataparallel C. 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.

3.1 Instrumentation Run

Instrumentation required by the dynamic 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

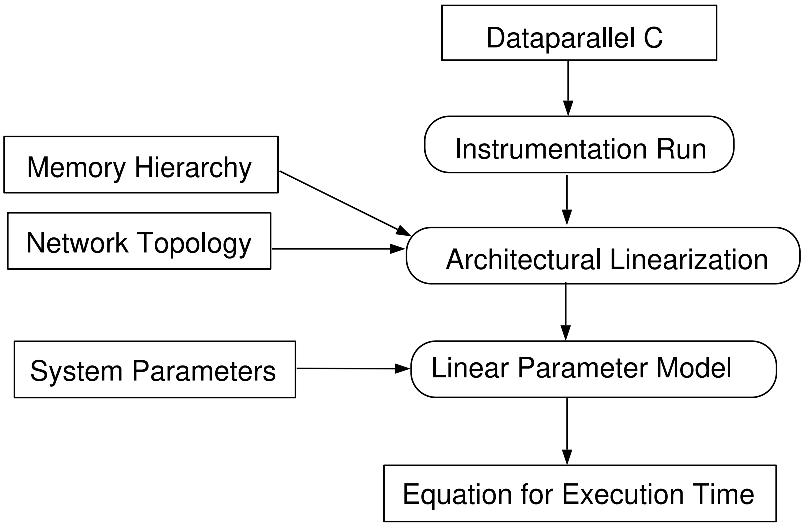


Figure 1: Block diagram for dynamic performance prediction system

routine. The experiments described here were performed by inserting calls to the instrumentation library by hand, but related work [27] has implemented a source-to-source compiler using the same instrumentation techniques.

Data parallel compilers often replicate scalar values on all processors and distribute array variables across the nodes of the system. Total data volume for scalable applications is dominated by array variables. The instrumentation code accounts for the size of array variables by declaring a "shape" descriptor structure that specifies the number of dimensions in a array variable and the number of positions in each dimension.

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. This symbolic information is used to analyze complex iteration constructs. Conditional code is also instrumented to determine true ratios. Related research with Vienna Fortran has indicated that true ratios remain fairly constant as problems are scaled up[9]. If a conditional block of code was entered 20% of the time during the instrumentation run, we assume the the production run will have similar results. Sections of code which will be executed in parallel are instrumented with descriptor structures which are organized in a linked list. Information on the data size accessed during parallel execution is used to estimate 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 system 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. 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 integer 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 constructs were amenable to this analysis strategy. Shape descriptor structures are also scaled to the problem size in a similar manner.

The instrumentation library outputs a symbolic equation for the execution time of each basic block. These equations are independent of the particular machine architecture used and will be passed to the linearization phase where a specific architecture will be modeled. The equations are written to an output file in the Maple symbolic computation language. When these equations are combined with Maple code describing system parameters, the Maple system can plot performance graphs and provide other performance information.

3.2 Architectural Linearization

The architectural linearization phase of performance prediction reduces complex machine characteristics to equations linear with respect to the speed of hardware subsystems. 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.

Predicting cache misses accurately is an extremely difficult problem and the approach taken here provides only a crude estimation of what actual cache behavior will be. This methodology was chosen in order to meet our design goals of using only static analysis and an instrumentation run in order to build our application model. References to array variables in Dataparallel C have a highly sequential access pattern, making the estimation of cache and virtual memory behavior somewhat easier. For our analysis we assume that if the portion of the array variable residing on each processor is greater than the cache size, then the processor will have a miss in the cache for the whole data array. The statistical analysis performed to determine the cost of a cache miss helps to mitigate errors introduced by this admittedly rough estimation of cache misses. As a result of this linearization step, expressions for the number of cache misses and virtual memory faults can be derived.

Interprocessor communication can have a significant impact on the performance of a parallel application. We have characterized communications by specifying the message startup cost, the number of bytes transmitted and the type of communication. Neighbor communications require a single message while broadcasts using a binomial tree algorithm require time logarithmic in the number of processors. A fairly small number of communication patterns are used in data parallel programs. The complexity of these patterns has been investigated in previous work [19] and is included in our model of the application.

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.

3.3 Linear Parameter Model

Given counts of each operation ($\mathcal{X}$ values) and the cost for that operation (β values) on a given system, total execution time can be predicted for the application being modeled. For a given problem size and number of processors the execution time

$$t = \mathcal{X}_{Ops}\beta_{Ops} + \mathcal{X}_{PAR}\beta_{PAR} + \mathcal{X}_{L1}\beta_{L1} + \mathcal{X}_{L2}\beta_{L2} + \mathcal{X}_{St}\beta_{St} + \mathcal{X}_{Bw}\beta_{Bw} + e$$

where e is the error, or difference between the predicted and actual time. The variable β_{Ops} is the time in nanoseconds to perform one CPU operation and $\mathcal{X}_{Ops}$ the number of instructions. β_{PAR} is the overhead associated with the transition between serial and parallel execution and $\mathcal{X}_{PAR}$ is the number of transitions.

The values β_{L1} and β_{L2} represent the penalty for a miss in level one and two cache respectively with $\mathcal{X}_{L1}$ and $\mathcal{X}_{L2}$ the number of cache misses in each cache. The variable β_{St} is the message startup cost with $\mathcal{X}_{St}$ the number of messages sent. Bandwidth considerations are dealt with through β_{Bw} as the time required to transmit one byte through a connection to a processing element and $\mathcal{X}_{Bw}$ and the number of bytes transmitted. The predicted execution time $\hat{t} = t - e$.

If we consider a sample of n observations with $\mathcal{X}$ values from applications with different problem sizes and numbers of processors, then in vector notation, we have:

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} = \begin{bmatrix} \mathcal{X}_{Ops_1} & \mathcal{X}_{PAR_1} & \mathcal{X}_{L1_1} & \mathcal{X}_{L2_1} & \mathcal{X}_{St_1} & \mathcal{X}_{Bw_1} \\ \mathcal{X}_{Ops_2} & \mathcal{X}_{PAR_2} & \mathcal{X}_{L1_2} & \mathcal{X}_{L2_2} & \mathcal{X}_{St_2} & \mathcal{X}_{Bw_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathcal{X}_{Ops_n} & \mathcal{X}_{PAR_n} & \mathcal{X}_{L1_n} & \mathcal{X}_{L2_n} & \mathcal{X}_{St_n} & \mathcal{X}_{Bw_n} \end{bmatrix} \begin{bmatrix} \beta_{Ops} \\ \beta_{PAR} \\ \beta_{L1} \\ \beta_{L2} \\ \beta_{St} \\ \beta_{Bw} \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

or

$$\boldsymbol{t} = \boldsymbol{\mathcal{X}}\boldsymbol{\beta} + \boldsymbol{e}$$

where

$$\begin{aligned} \boldsymbol{t} &= \text{a column vector of } n \text{ observed values of } t = \{t_1, t_2, \ldots, t_n\} \\ \boldsymbol{\mathcal{X}} &= \text{an } n \text{ row column matrix containing linearized algorithmic characteristics} \\ \boldsymbol{\beta} &= \text{a column vector containing the cost in seconds for each operation} \\ \boldsymbol{e} &= \text{a column vector of the errors between predicted and experimental values} \end{aligned}$$

Values for the $\boldsymbol{\beta}$ vector can be obtained from system manufacturers or benchmark programs. The matrix formulation of the model also makes it possible to use multivariate techniques to obtain β values given a statistically significant number of experimental runs with different problem sizes and numbers of processors. These statistical techniques will be explored in the next section.

The dynamic performance prediction methodology presented here derives detailed symbolic equations accounting for major components in execution time. These equations can be used to analyze performance for scalable applications as the problem size and architecture varies.

4 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 [18]. 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; hence, more complex methods are needed to estimate the value of model coefficients. Statistical software packages such as S-PLUS [26] allow large quantities of multivariate data to be analyzed with relative ease [2].

4.1 Advantages

Using statistical techniques to estimate the coefficients (β values) for the dynamic performance prediction model has several advantages:

- Statistical packages provide standard error values for each of the prediction variables. These values 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 assesses the correlation of the model to experimental data. This information allows us to tune the model in order to reduce prediction error.

Multivariate statistical analysis must have access to a large number of samples in order to fit the model. 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. The coefficients derived from S-PLUS were used to specify the β values used by the Maple system. Statistical methods for finding coefficients may also be applicable to non-scalable applications if a larger number of sample applications are available.

4.2 Assumptions

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. 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 $\mathcal{S}$. Our experience has shown that as long as $\mathcal{S}$ contains a good distribution of samples, the fit is quite good for programs not in the fitting set.

- 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 and our results indicated that this assumption was somewhat reasonable. Actual errors will probably be skewed towards positive values since negative times are not possible. Additional research should be performed to determine which distributions would be a better fit to prediction errors.

4.3 Statistical Model

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.

$$y = \beta_1 x_1 + \cdots + \beta_p x_p$$

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 $\sum_{i=1}^{n}(y_i - (\sum_{j=1}^{p} \beta_j x_{ij}))^2$ [2].

Our dynamic 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. Since the algorithmic characteristics have been abstracted into simple operation counts, the statistical package can make predictions, with confidence intervals, for other parallel applications on the selected platform.

4.4 Experimental Results

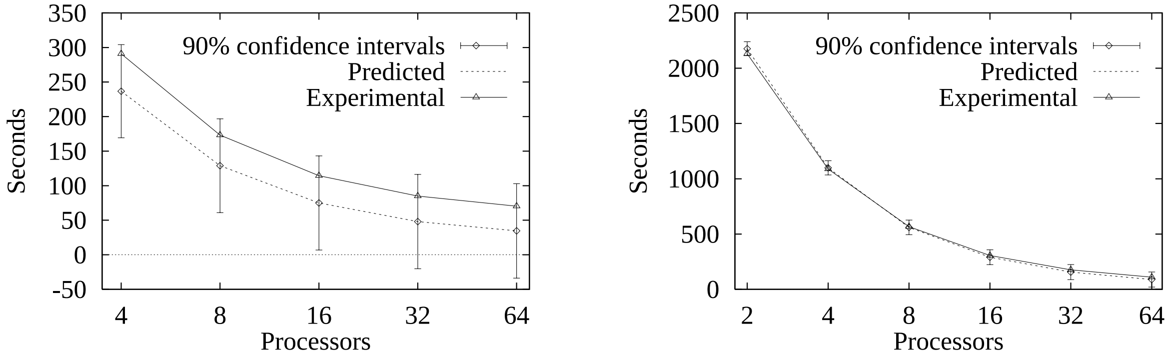
The statistical model was fit using actual runs on the iPSC/860, nCUBE 3200 and Meiko CS-2 multicomputers. The S-PLUS software package was then used to predict performance for two fluid dynamics modeling applications not included in the fit process. The results indicate that β values derived from sample applications can generalize to future programs analyzed by this performance prediction methodology.

Our initial experiments were run with the nCUBE 3200 multicomputer. The model was fit using three applications (Gaussian elimination, matrix multiplication and a Shallow Water Model) with several different problem sizes for a total of 58 experimental runs. The Shallow Water Model solves the system of shallow water equations using a finite difference method. A detailed description of the experimental results was presented previously [5]. The model was then tested on an ocean circulation model and a "sharks world" simulation application [3].

Figure 2 illustrates predicted output for the Ocean Circulation Model on the nCUBE 3200. The program models wind-driven circulation in a density-stratified ocean [19]. 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. Communication costs make up a higher fraction of total execution time for the smaller problem in Figure 2. Since the standard error value for communications is higher than that for computations, the confidence interval for the 128 segment problem is wider.

Figure 3 examines predicted output for the Shallow Water Model application on the 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 [19]. For this experiment we rely exclusively on published experimental data for execution times in order to fit the model. The instrumentation run for the applications was performed on a 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 [22]. Each node is equipped with two vector processors to improve floating point performance. A



(a) (b)

Figure 2: Predicted and experimental execution time in seconds for the Ocean Circulation Model on the nCUBE 3200 multicomputer with (a) 128 and (b) 640 segments in the east-west direction.

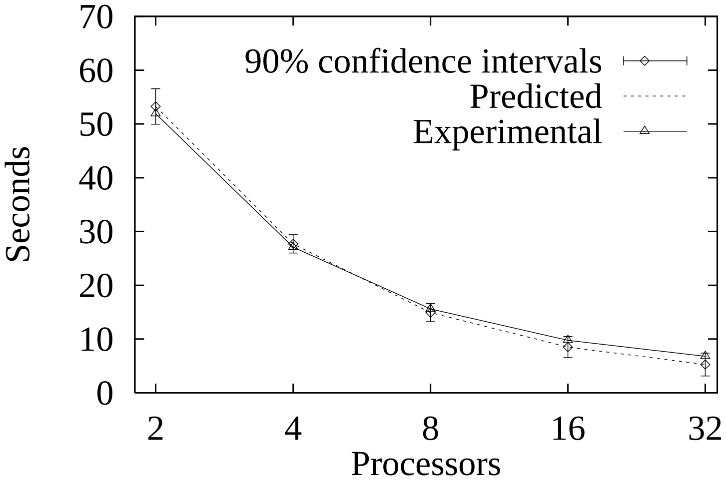


Figure 3: Predicted and experimental values for the Shallow Water Model with a 64×64 grid and 1200 iterations for an iPSC/860 multicomputer.

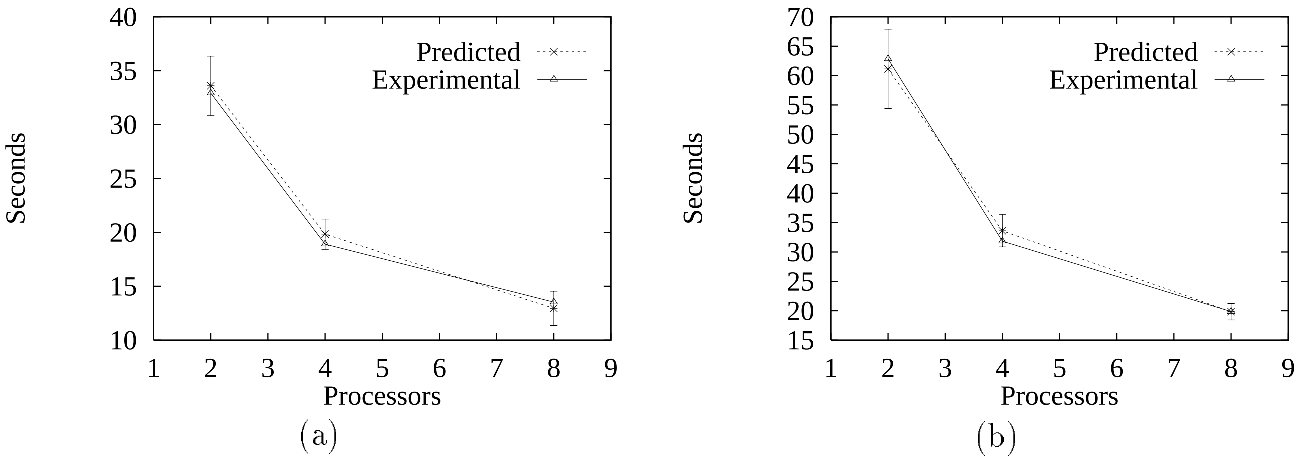


Figure 4: Predicted and experimental execution time in seconds for the Ocean Circulation Model on the Meiko CS-2 multicomputer with (a) 640 and (b) 1280 segments in the east-west direction. The error bars are 90% confidence intervals for the predicted values.

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 4 shows the results obtained for the Ocean Circulation Model with two different problem sizes. 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 iPSC/860 which use the NX message passing interface provided on the Meiko. This extra level of software 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.

If designers and users of parallel systems are to benefit from the results of performance prediction, they must be able to determine how accurate the predictions are. The statistical methods described here allow confidence intervals to be placed on predictions in order to fulfill this requirement. The statistical information can be obtained in an automated fashion using statistical software packages such as S-PLUS [26]. Additional information from the statistical analysis can also guide the development of more accurate models.

5 Evaluation of Dynamic Modeling Techniques

Several applications were analyzed during the course of this research in order to validate dynamic performance prediction techniques. Matrix multiplication, a linear system solver, an ocean circulation model and a shallow

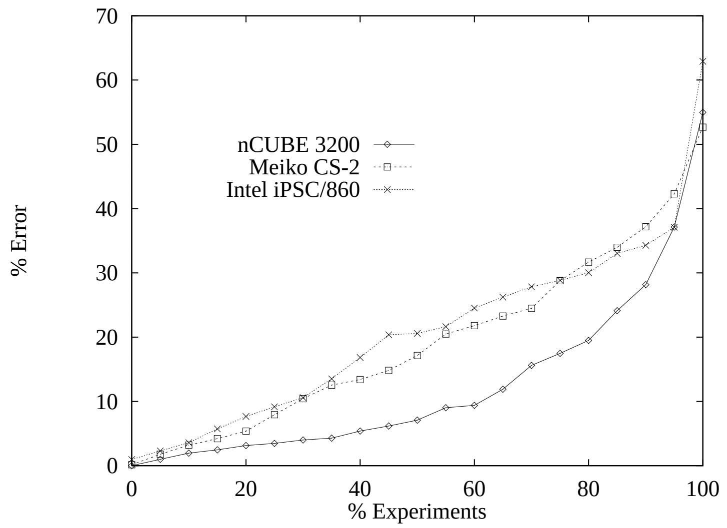


Figure 5: Percent error for experimental runs of applications in the validation suite.

water model were examined. Although the results would be strengthened by more significant applications, these kernels bear some similarity to other scientific problems.

For our error analysis we have used the classical error computation method:

$$Error = \frac{Predicted - Actual}{Actual}$$

A total of 246 executions of the validation suite applications were performed using different problem sizes and numbers of processors. The error contours shown in Figure 5 indicates that over 90% of the experimental runs achieve less than 40% error for all three machines. All of the experiments achieved less than 60% error.

The standard deviation of errors

$$\sigma = \sqrt{\frac{SSE}{n-3}}$$

where the sum of squared errors

$$SSE = \sum_{i=1}^{n} e_i^2$$

can be examined to determine the quality of the model. The weighted average value of σ is 5.4 seconds. The weighted average of the mean $\bar{t}$ is 53.7 seconds. The ratio of standard deviation to the mean, or the coefficient of variation (C.O.V.) can also be used as a unitless measure of error. For the nCUBE and Meiko the standard deviation was less than 9% of the mean experimental value. One problem with the standard deviation of error is that it tends to be influenced disproportionately by the relative error of long experimental runs because of the squared nature of SSE. The average error of all validation runs indicates that the model is able to predict results at an acceptable level for the uses we have targeted. A summary of the accuracy analysis is shown in Table 1.

System	Number of Experiments	σ	$\bar{t}$	C.O.V.	Average Error
nCUBE 3200	115	8.87 sec	105 sec	0.084	12%
Meiko CS-2	63	0.55 sec	6.36 sec	0.087	20%
iPSC/860	46	5.4 sec	12.7 sec	0.425	23%

Table 1: Experimental error values.

Given a validated model the performance of applications and architectures can be analyzed in detail. Our validation testing indicates that an accurate performance model can be obtained with minimal user intervention through leveraging existing software tools for symbolic computations and statistical analysis. Many aspects of parallel performance have been examined from a purely theoretical perspective. Additional insights can be obtained from a detailed analysis of performance using an analytical model derived from an actual application. The following examples show how this performance prediction system has been used to analyze performance as the problem size and architecture are scaled.

A fundamental use of performance prediction results is in the area of performance debugging. Figure 6 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. 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.

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 7, the execution time was differentiated with respect to message startup cost where

$$Sensitivity = \frac{dt}{d\beta_{St}}$$

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. Similar analysis can be performed to determine the effect of other system parameters.

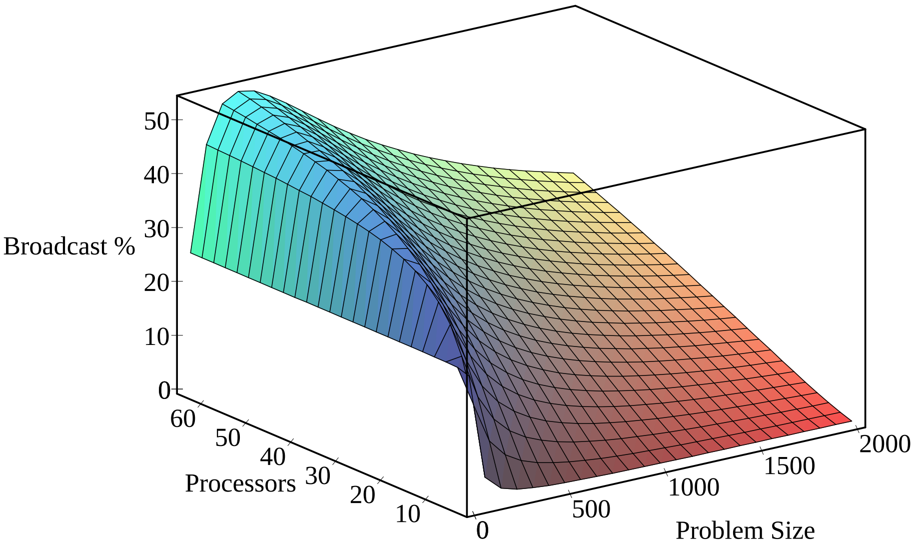


Figure 6: Percentage of execution time spent in broadcasting the pivot row for Gaussian elimination on an nCUBE 3200.

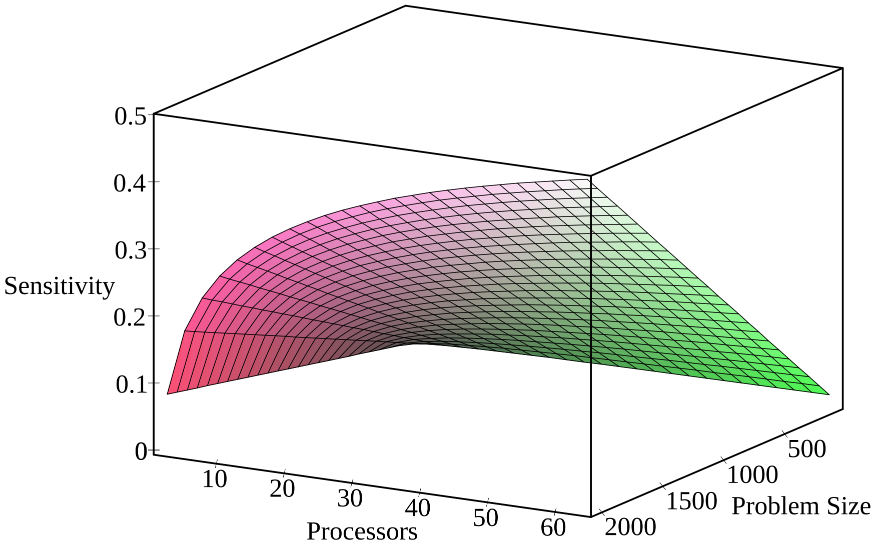


Figure 7: Sensitivity to changes in message startup cost for Gaussian elimination as a function of problem size and number of processors.

6 Conclusions

Performance prediction can be used in performance debugging of parallel applications and in making architectural improvements for multicomputer hardware and system software. Using predicted performance results for existing commercial MPP systems, purchasing agents can also make informed procurement decisions.

The dynamic 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. The simultaneous use of analytical and sampling techniques allows this technique to accomplish the goals of ease of use and accuracy which are important for performance prediction models.

If decisions are to be made on the basis of predicted behavior, the accuracy of the estimates must be specified. By using multivariate statistical analysis to determine critical system parameters, confidence intervals can be determined for predicted results. This statistical characterization can also assist in improving the accuracy of the model. By automatically determining the values for system parameters, the overall ease of use is also improved.

Future work will focus on improving the accuracy of the model and using the results to explore different parallel architectures. We also plan to develop portability measures similar to the isospeed metric [28] which can be quantified by analyzing the sensitivity of applications to changes in system parameters. Several alternatives are also being investigated to incorporate the effects of load imbalance into the performance prediction system.

The accuracy of this system is dependent upon extensive knowledge about how the compiler generates code. In order for debuggers and performance prediction tools to be effective, more information must be provided by the compiler about which optimizations were performed on which blocks of code.

Dynamic performance prediction has been shown to be effective in meeting the goals 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] J. M. Chambers and T. J. Hastie. *Statistical Models in S.* Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California, 1992.

[3] M. J. Clement. *Analytical Performance Prediction of Data-Parallel Programs.* PhD thesis, Oregon State University, 1994.

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

[5] 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.

[6] M. J. Clement, M. R. Steed, and P. E. Crandall. Network performance modeling for PVM clusters. In *Proceedings of Supercomputing '96*, Nov 17-20 1996.

[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. 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.

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

[10] T. Fahringer. *Automatic Perormance Prediction of Parallel Programs.* Kluwer Academic Publishers, The Netherlands, 1996.

[11] T. Fahringer. Toward symbolic performance prediction of parallel programs. In *Proceedings of the 10th International Parallel Processing Symposium*, April 15-19 1996.

[12] 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.

[13] G. Fox. What have we learnt from using real parallel machines to solve real problems? Technical Report C3P-522, Caltech, December 1989.

[14] M. Gupta and P. Banerjee. Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):179–193, March 1992.

[15] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, May 1988.

[16] S. Hackstadt and A. Malony. Next-generation parallel performance visualization: A prototyping environment for visualization development. In *Proceedings of Parallel Architectures and Languages Europe (PARLE), Athens, Greece*, July 1994.

[17] G. Haring and A. Ferscha. Performance oriented development of parallel software with capse. In *Proceedings of the Workshop on Environments and tools for Parallel Scientific Computing, Blackberry-Inn, Walland/Tennesee*, May 25-27 1994.

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

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

[20] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing.* Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.

[21] P. Mehra, M. Gower, and M. Bass. Automated modeling of message-passing programs. In *Proc. Int'l. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 94), Durham, NC*, pages 187–192, Jan 1994.

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

[23] C. M. Pancake. Why is there such a mis-match between user needs and tool products? Keynote address, 1993 Workshop on Parallel Computing Systems, Keystone, Colorado., April 1993.

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

[25] F. Sotz. A method for performance prediction of parallel programs. In *Proceedings of CONPAR 90 - VAPP IV, Zurich, Switzerland*, September 1990.

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

[27] M. R. Steed and M. J. Clement. Performance prediction of PVM programs. In *Proceedings of the 10th International Parallel Processing Symposium*, April 15-19 1996.

[28] X.-H. Sun and D. T. Rover. Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems*, 5(6):599–613, June 1994.