Appeared in "Proceedings of the 10th International Parallel Processing Symposium"

Performance Prediction of PVM Programs[1]

Michael R. Steed and Mark J. Clement
Department of Computer Science
Brigham Young University
Provo, UT 84602-6576
{steed,clement}@cs.byu.edu

Abstract

As workstation clusters gain popularity as a parallel computing platform, there is an increasing need for performance tools that support these platforms. Performance prediction is important for performance analysis of scalable parallel applications. Although there are several performance tools available that work with PVM programs, none are capable of performance prediction. We have developed the APACHE (Automated PVM Application Characterization Environment) performance prediction system for PVM programs running on workstation clusters. We review the system implementation and present some experimental results obtained with the system.

1. Introduction

Performance prediction is an important tool for performance analysis and debugging of scalable parallel applications [5]. Performance prediction involves modeling program performance as a function of the hardware and software characteristics of a system. By changing these characteristics in the model, program execution time can be predicted for a variety of platforms and configurations. Performance prediction tools have been implemented for several high-level parallel languages [1, 5, 7, 9].

With workstation clusters becoming more popular as a parallel computing platform, there is a growing need for performance tools that support these platforms. One of the most common ways to write parallel programs for a workstation cluster environment is to use a sequential language augmented with a message passing library such as Parallel Virtual Machine (PVM) [11]. Although there are some performance debugging tools that work with PVM programs, to our knowledge there are no tools capable of performance prediction.

In this paper we present the APACHE (Automated PVM Application Characterization Environment) performance prediction system for C programs written to use the PVM message passing library.

2. Related Work

Several existing performance debugging tools support PVM. Recent versions of PVM include instrumented libraries for recording trace data, which the XPVM graphical interface [12] can use to provide runtime and post-mortem visualization of program execution. Because the trace data is stored in Pablo's Self-Defining Data Format [16], it can be used with any visualization system that accepts SDDF files.

The Paradyn [13] performance tools, which are designed for runtime performance monitoring and bottleneck detection, have been ported to the PVM platform. Other debuggers are available, including Aims [18], Xab [2], and Xmdb [8].

Some graphical programming environments are available for PVM, including HeNCE [3] and VPE [14]. In addition to visual programming facilities, these environments also provide some debugging capabilities.

Performance prediction tools and performance debuggers are designed to solve different types of problems. For example, a debugging tool such as Paradyn would be well suited to discovering deadlocks in a parallel algorithm, while

[1]This research was supported by Sprint Corporation

a performance prediction tool such as APACHE would be better at assessing an application's sensitivity to network latency. To our knowledge, there are no tools capable of performance prediction of PVM programs.

3. Performance Models

In this section we describe the models we developed to represent the performance aspects of the hardware and software of a system.

3.1. Computation Model

To model computation costs, we assign a cost factor C_i, measured in execution time, to each of the N_{ops} different operations that may be performed during program execution. We use simple benchmark programs and linear regression techniques [6] to obtain values and confidence intervals for each C_i on hardware platforms of interest. We found that, on the workstations we used, some operations have significantly different costs than others. For example, a division operation takes over twice as long as a multiplication, and some integer operations take significantly longer than their floating-point counterparts.

Given the cost factors C_i, we express the predicted computation time T_{comp} spent during program execution as

$$T_{comp} = \sum_{i=1}^{N_{ops}} C_i N_{op(i)} \tag{1}$$

where $N_{op(i)}$ is the number of times operation i is performed during program execution.

Other performance prediction systems are based on the *uniform cost assumption*, which is that all floating point operations have the same cost (measured in execution time) [15]. In contrast, the APACHE system accounts for the actual cost of each type of operation. The difference in execution time for a division compared to a multiplication, for instance, can be significant, particularly when a large fraction of the total execution time is spent in computation.

3.2. Communication Model

We have developed a model to represent the performance of PVM communications over Ethernet. Similar models could be developed for other networking technologies such as Asynchronous Transfer Mode (ATM).

Other researchers who have modeled the performance of interprocessor communication report that a simple model that accounts for message latency and network bandwidth gives adequate results [17]. According to this model, the predicted communication time T_{comm} spent during program execution can be expressed as

$$T_{comm} = \sum_{i=1}^{N_c} (\alpha + \frac{B_i}{\beta}) \tag{2}$$

where N_c is the total number of communications per processor, α is the message latency, β is the network bandwidth, and B_i is the size of message i.

With shared-medium networks such as Ethernet, contention for bandwidth can significantly affect network throughput. Therefore, to model the performance of PVM communications over Ethernet, we add a contention factor, γ, to Equation (2):

$$T_{comm} = \sum_{i=1}^{N_c} (\alpha + \frac{\gamma B_i}{\beta}) \tag{3}$$

According to our experiments, a contention factor of $\gamma = P$, where P is the number of processors, provides a good approximation of Ethernet contention, assuming that all P processors are communicating simultaneously.

Our experiments show that Equation (3) predicts the performance of PVM communications quite accurately for different message sizes and numbers of processors, assuming that all the processors communicate simultaneously and that the load due to communication is balanced across the processors. We found that the variance in actual communication time was highest for four or more processors sending small messages, and lowest for any number of processors sending large messages. We attribute this behavior to variations in message startup cost, which constitute a higher percentage of the total communication time for small messages.

4. System Structure

We have made some simplifying assumptions in the design of the APACHE system. First, the system is restricted to the set of programs that conform to the SPMD (single program, multiple data) model. This is not a serious limitation; Fox [10] has reported that 90 percent of scientific problems are amenable to SPMD solutions.

Another assumption concerns the homogeneity of the virtual parallel machine. Although one of the most important features of PVM is its ability to combine heterogeneous computers and network connections into a single parallel machine, this heterogeneity can make performance prediction arbitrarily complex. Therefore, we assume that the nodes of the parallel machine and their network connections are homogeneous.

A block diagram of the APACHE system is shown in

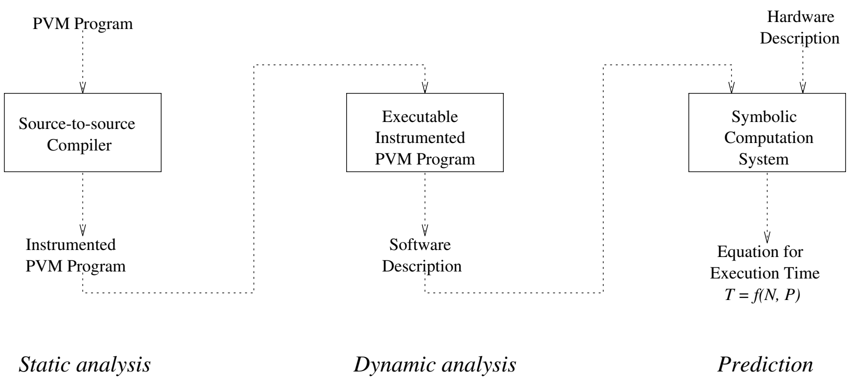

Figure 1. Logical structure of the APACHE performance prediction system.

Figure 1. The performance prediction process consists of static analysis, dynamic analysis, and prediction phases, which we explain in the remainder of this section.

4.1. Static Analysis

In the static analysis phase, the source-to-source compiler constructs a call graph of the PVM program. For each basic block, the compiler records the number of times that each type of operation occurs in that block, loop initialization expressions and termination conditions, and the number of calls to message passing routines.

In addition to construction a call graph of the program, the source-to-source compiler produces an instrumented version of the PVM program for use during the dynamic analysis phase. The instrumentation code consists of statements to count the number of times each basic block of the program is executed, instrumented versions of the PVM communication routines that record the size of each message transmitted, and additional library routines used during dynamic analysis.

4.2. Dynamic Analysis

During the dynamic analysis phase, the instrumented PVM program produced by the source-to-source compiler is executed, and the instrumentation code counts the number of times each basic block is executed and the size of each message transmitted. Block iteration counts are used to determine true loop iteration counts and branch ratios of conditional constructs.

Given the particular problem size, N, and the number of processors, P, used for the execution of the instrumented program, simplified fits are performed to express per-block execution counts as functions of N and P. If a simple fit cannot be performed for a particular block (e.g., because a loop iterates an unusual number of times), an error message is displayed and dynamic analysis is interrupted. In these cases, the time complexity of the block can be specified manually by inserting a compiler pragma of the form `#pragma complexity <exp>` somewhere inside the block. Here `<exp>` is a Maple expression potentially involving N, P, and iteration variables of enclosing loops, that describes the number of times the block is executed. Simple fits are also performed to express the size of each message as a function of N and P (or as a constant).

The information gathered during dynamic analysis is combined with the per-block operation counts gathered during static analysis to produce equations for total computation and communication requirements of the program. These equations, which conform to the syntax of the Maple symbolic computation system [4], constitute the "Software Description" indicated in Figure 1.

The "Hardware Description" referred to in Figure 1 consists of the cost factors C_i in Equation (1), and the α, β and γ terms describing the network characteristics in Equation (3). As mentioned above, we use benchmark programs and multiple regression to obtain values and confidence intervals for these parameters.

4.3. Prediction

The equations for computation and communication requirements are combined with the values constituting the hardware description to produce an equation for total execution time. This equation can be used to predict program execution time for different problem sizes and numbers of processors and examine other aspects of program performance. In addition, the factors in the equation can be changed to explore the performance impact of different types of processors and network media.

5. Experimental Results

5.1. Predicting Execution Time

In this section we illustrate the use of the APACHE system for predicting the execution time of programs implementing parallel matrix multiplication and the Jacobi algorithm.

Our implementation of parallel matrix multiplication computes $\mathbf{C} = \mathbf{A} \times \mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are $N \times N$ matrices. To parallelize the computation, we distribute $\frac{N}{P}$ contiguous rows of $\mathbf{A}$ and $\frac{N}{P}$ contiguous columns of $\mathbf{B}$ to each processor. Each processor computes the appropriate subblock of $\mathbf{C}$ and then passes its columns of $\mathbf{B}$ to the neighboring processor. After P iterations, the computation is complete, with the resulting $\mathbf{C}$ matrix distributed across the processors.

Figure 2 shows the predicted execution time (in seconds) and prediction error for the matrix multiplication program. The predictions generated by the APACHE system are quite accurate, with 90% of the experimental runs exhibiting less than 15% prediction error.

The Jacobi algorithm is an iterative method for solving the linear system $\mathbf{Ax} = \mathbf{b}$ for the unknown vector $\mathbf{x}$ [15]. The algorithm can be used to solve the two-dimensional steady state temperature distribution problem over an $N \times N$ grid. In this problem, the cells of the grid are initialized with initial temperature values, some of which are fixed (i.e., as heat sources or heat sinks). During each iteration of the algorithm, a new temperature value for each cell is computed with the formula

$$x'_{i,j} = \frac{x_{i-1,j} + x_{i,j-1} + x_{i+1,j} + x_{i,j+1}}{4}$$

The values on the right side of the equation represent the temperature of the four-neighbors of cell $(i,\ j)$ during iteration t, and the value on the left is the new temperature for that cell for iteration $t+1$.

We parallelize the Jacobi algorithm by mapping $\frac{N}{P}$ contiguous rows of the grid onto each processor. During each iteration, processors exchange values of neighboring rows. The Jacobi algorithm iterates until the maximum change across the cells of the grid is smaller than some value ε.

Figure 3 shows the predicted execution time and prediction error of the Jacobi programs. All of the observed execution times for this program are within about 7% of the predicted times.

5.2. Performance Debugging

The APACHE system can be used to characterize programs for performance debugging purposes. For example, Figure 4 shows the percentage of time spent in communication and computation for the matrix multiplication program as N and P vary. This type of graph is useful for determining when communication becomes the limiting factor in program performance.

6. Conclusions and Future Work

We have presented some experimental results obtained with the APACHE performance prediction system for PVM programs. In the future, we plan to make further studies of the accuracy of our system and refine the performance models as necessary. We also plan to use the APACHE system as a tool for identifying applications that can benefit from high-speed LAN technology such as ATM. Our performance prediction tool is well suited to this problem because it allows the hardware characteristics in the performance model to be changed easily.

Finally, we are interested in using the APACHE system as the basis for a performance debugging tool for PVM programs. The APACHE system allows performance models to be constructed for each basic block of a PVM program. This information can be used to explore the predicted program performance of different parts of an application for varying system configurations.

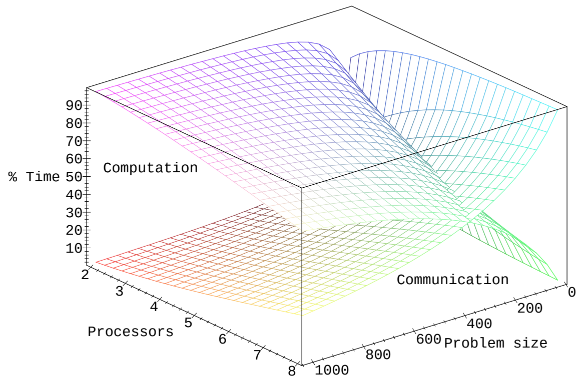

Figure 4. Predicted percent time spent in communication and computation for matrix multiplication program.

References

[1] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer. A Static Performance Estimator to Guide Data Partitioning Decisions. In *Proceedings of PPoPP 91*, pages 213–223. ACM SIGPLAN Notices, 1991.

[2] A. L. Beguelin. Xab: A Tool for Monitoring PVM Programs. Tech. Rep. CMU-CS-93-164 (June 1993), Carnegie Mellon University, School of Computer Science.

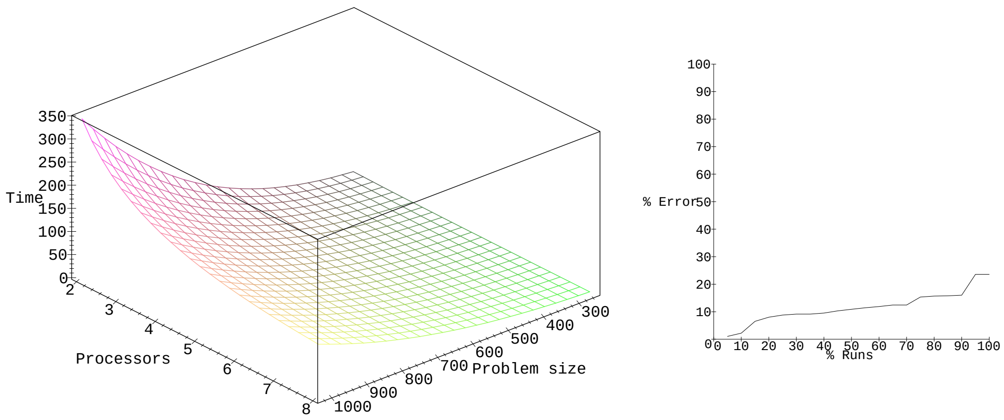

Figure 2. Predicted execution time and percent error of actual times for matrix multiplication program.

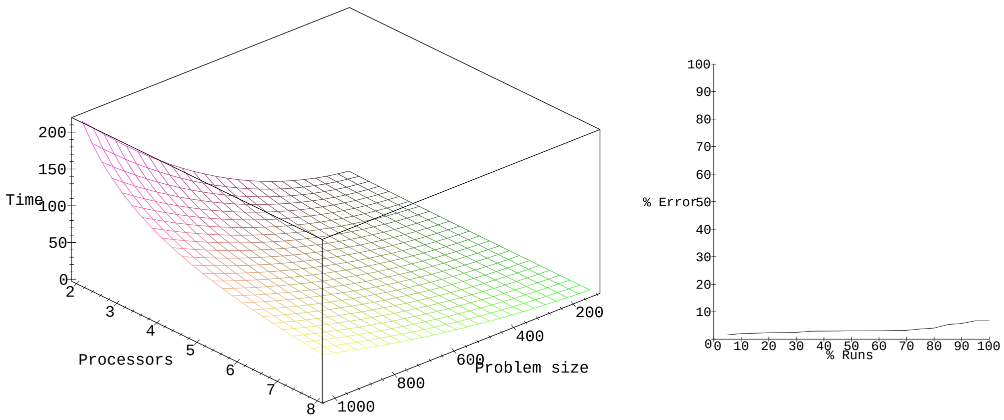

Figure 3. Predicted execution time and percent error of actual times for Jacobi program.

[3] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and K. Moore. HeNCE: A Heterogeneous Network Computing Environment. Tech. Rep. CS-93-205 (August 1993), University of Tennessee.

[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. *Analytical Performance Prediction of Data-Parallel Programs*. Ph.D. thesis, Oregon State University, 1994.

[6] 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 1995.

[7] M. E. Crovella. *Performance Prediction and Tuning of Parallel Programs*. Ph.D. thesis, University of Rochester, 1994.

[8] S. K. Damodaran-Kamal and J. M. Francioni. Mdb: A Semantic Race Detection Tool for PVM. In *Proceedings of the 1994 Scalable High Performance Computing Conference*, May 1994.

[9] T. Fahringer and H. P. Zima. A Static Parameter Based Performance Prediction Tool for Parallel Programs. In *Proceedings of International Conference on Supercomputing*, pages 207–219. ACM SIGARCH, ACM Press, 1993.

[10] G. C. Fox. What Have We Learnt From Using Real Parallel Machines to Solve Real Problems?. Tech. Rep. C3P-522 (December 1989), California Institute of Technology.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Networked Computing*. MIT Press, 1994.

[12] J. A. Kohl and G. A. Geist. XPVM 1.0 User's Guide. Tech. Rep. 12981 (April 1995), Computer Science and Mathematics Division, Oak Ridge National Laboratory.

[13] B. P. Miller, J. K. Hollingsworth, and M. D. Callaghan. The Paradyn Parallel Performance Tools and PVM. In *Environments and Tools for Parallel Scientific Computing*. SIAM Press, 1994.

[14] P. Newton and J. Dongarra. Overview of VPE: A Visual

Environment for Message-Passing Parallel Programming. Tech. Rep. CS-94-261 (1994), University of Tennessee.

[15] M. J. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill, 1994.

[16] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. Schwartz, and L. F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proceedings of the Scalable Parallel Libraries Conference.* IEEE Computer Society, 1993.

[17] H. Wabnig and G. Haring. Performance Prediction of Parallel Systems with Scalable Specifications—Methodology and Case Study. *Performance Evaluation Review* **22** (2–4), 46–62 (1995).

[18] J. Yan. Performance Tuning with AIMS: An Automated Instrumentation and Monitoring System for Multicomputers. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, January 1994.