REFERENCES


Chapter-9

PERFORMANCE PREDICTION FOR PARALLEL SCIENTIFIC APPLICATION

INTRODUCTION

Message passing parallel programs that run on cluster of workstations are often used to solve problems that would take too long to run on conventional sequential machines. Understanding and predicting the performance of such programs is a complex task. It is important to understand the performance characteristics of computation, communication and message passing software used for parallelization [1].

One of the keys to making useful, accurate performance predictions in such situations is to have a clear picture of the communication characteristics of the application. There are many hurdles to obtaining useful, accurate performance measurements of communications operations for parallel computers and application.

Performance prediction is an important tool for the performance analysis and debugging of scalable parallel applications [2]. Performance prediction system models the program Performance as a function of the hardware and software parameters of the system. The execution time of a program can be predicted by changing the software and hardware parameters in the model on a variety of platforms and configurations. Performance prediction tools have been implemented for several high-level parallel languages [3, 4].

Due to the popularity of workstation clusters 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 Message Passing Interface (MPI). Although there are some performance debugging tools that work with MPI programs, but to our best knowledge there are no tools capable of accurate performance prediction.

This chapter presents our developed Automatic Parallel Application Prediction System (APAPS) model as a performance prediction tool designed to solve many MPI programs. Performance prediction tools and performance debuggers are designed to solve different types of problems. For example, a debugging tool such as Paradyn [5] would be well suited to discovering deadlocks in a parallel algorithm, while a performance prediction tool such as APAPS would be better at assessing an application’s sensitivity to network latency.
9.1. RELATED WORK

There is several performance debugging tools that support MPI. The Paradyn [5] performance tools, which were designed for runtime performance monitoring and bottleneck detection, have been ported to the MPI platform. Other debuggers are available, including [6], [7], and [8].

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 a performance prediction tool such as similar to APAPS would be better at assessing an application’s sensitivity to network latency.

9.2. PERFORMANCE MODELS

Performance Models are used to represent the performance aspects based on the hardware and software of the system. This section describes the performance models used to represent different performance aspects of different parallel programs. In our designed system, performance models were used which are discussed in the following subsections:

9.2.1. Computation Model

In our computational model computational costs are assigned to each operation. 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 to obtain values and confidence intervals for each $C_i$ on hardware platforms of interest. It is observed that, on the workstations 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 $C_i$, the predicted computation time $T_{comp}$ spent during program execution can be obtained by the equation given below:

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

(1)

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

9.2.2. Communication Model

The communication model developed is used to represent the performance of MPI communications over Ethernet. Similar models could be developed for other networking technologies such as Asynchronous Transfer Mode (ATM). Many researchers who have modeled the performance of inter-processor
communication report that a simple model that accounts for message latency and network bandwidth gives adequate results [10]. According to this model, the predicted communication time $T_{comm}$ spent during program execution can be expressed in the equation given below:

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

(2)

where $N_c$ is the total number of communications per processor, $\alpha$ is the message latency, $\beta$ 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 MPI communications over Ethernet, contention factor is added to Equation (2),

$$T_{comm} = \sum_{i=1}^{N_c} \left( \alpha + \frac{\lambda \times B_i}{\beta} \right)$$  

(3)

Where contention factor of $\lambda = P$, in which $P$ is the number of processors, gives a good approximation of Ethernet contention, assuming that all $P$ processors are communicating simultaneously.

9.2.3. Parallel Application Model

Adding together equations 1 and 3 provides a general equation for predicted execution time.

$$T_{total} = T_{comp} + T_{comm} = \sum_{i=1}^{N_{ops}} C_i \times N_{op(i)} + \sum_{i=1}^{N_c} \left( \alpha + \frac{\lambda \times B_i}{\beta} \right)$$  

(4)

In this equation $N_{op(i)}$, $N_c$ and $B_i$ vary with the application, the problem size $N$, and number of processors $P$. An MPI application model consists of equations for these parameters expressed as functions of $N$ and $P$.

Setting $\lambda$ to 1.0 effectively provides the communication Equation (2) for local area connections. By adjusting the value of $\lambda$ Equation (4) may be used for a variety of network connections.

9.3. SYSTEM STRUCTURE

The APAPS system is designed based on some of the assumptions. First, the system is restricted to the set of programs that conform to the SPMD (single program, multiple data) model. As many of the scientific problems are amenable to SPMD model, so this is not a serious limitation [11]. Another assumption concerns the homogeneity of the workstations used in the cluster of parallel machines. In our model homogeneous environment is considered to reduce the complexity of performance prediction. A block diagram of the APAPS system is shown in Fig 1.
The performance prediction process consists of three main phases: static analysis, dynamic analysis, and prediction. These are explained in details in the following subsections.

Fig. 1: Structure of the APAPS Performance Prediction System.

9.3.1. Static Analysis

In the static analysis phase, the source-to-source compiler constructs a call graph of the MPI 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. For the example code of a sample C++ program, the call graph is shown in Fig. 2.

```c++
#include <stdlib.h>

class A {
public:
    void a();
    void b();
};

int main() {
    A t;
t.a();
t.b();
exit(0);
}
```

Fig. 2: Sample C++ code and the call graph for that code.
In addition to the construction of a call graph of the program, the source-to-source compiler produces an instrumented version of the MPI 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 MPI communication routines that record the size of each message transmitted, and additional library routines used during dynamic analysis.

9.3.2. Dynamic Analysis

During the dynamic analysis phase, the instrumented MPI 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.

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. The equations, which conform to the syntax of the Maple Symbolic Computation System, constitute the "Software Description" indicated in Fig. 1.

The "Hardware Description" referred to in Fig. 1 consists of the cost factors $C_i$ in Equation (1), and the $\alpha$, $\beta$, and $\lambda$ terms describing the network characteristics in Equation (3). As mentioned above, we use benchmark programs to obtain values and confidence intervals for these parameters.

9.3.3. 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.

9.4. EXPERIMENTAL RESULTS

This section illustrates the use of the APAPS system for predicting the execution time of programs implementing parallel matrix multiplication. Here we will be discussing two parallel applications, one is the Matrix Multiplication and other is the Linear Equation Solver. First we discuss matrix multiplication in which our implementation of parallel matrix multiplication computes $C = A \times B$, where $A$ and $B$ are $N \times N$ matrices. To parallelize the computation, we distribute $N/P$ contiguous rows of $A$ and $N/P$ contiguous columns of $B$ to each processor. Each processor computes the appropriate sub block of $C$ and then passes its columns of $B$ to the neighboring processor. After $P$ iterations, the computation is complete, with the resulting $C$ matrix distributed across the processors. Fig. 3(a) and 3(b)
shows the predicted execution time (in seconds) and percent error of the time predicted by APAPS for the matrix multiplication program respectively. The predictions generated by the APAPS system are quite good with less than 15% prediction error. Fig. 3(a) shows some non-linear transition in the middle; this is because of some non-homogeneous nature of the cluster of workstations used and the communication overhead between a numbers of processors.

The second parallel application tested with APAPS system was a Linear Equation Solver, which is shown in below:

\[
\begin{align*}
  a_{n-1,0}x_0 + a_{n-1,1}x_1 + a_{n-1,2}x_2 + \cdots + a_{n-1,n-1}x_{n-1} &= b_{n-1} \\
  a_{2,0}x_0 + a_{2,1}x_1 + a_{2,2}x_2 &= b_2 \\
  a_{1,0}x_0 + a_{1,1}x_1 &= b_1 \\
  a_{0,0}x_0 &= b_0
\end{align*}
\]

where the a’s and b’s are constants and the x’s are unknowns to be found. The method used to solve for the unknowns \(x_0, x_1, \ldots, x_{n-1}\) is simple repeated “back” substitution. First, the unknown \(x_0\) is found from the last equation; i.e.,

\[
x_0 = \frac{b_0}{a_{0,0}}
\]

The value obtained for \(x_0\) is substituted into the next equation to obtain \(x_1\); i.e,

\[
x_1 = \frac{b_1 - a_{0,0}x_0}{a_{1,1}}
\]

The values obtained for \(x_1\) and \(x_0\) are substituted into the next equation to obtain \(x_2\), and is implemented with the following steps:

The above Linear Equation Solver using pipelining with MPI is used to get the resultant values of \(X_0\) to \(X_n\). In our approach the following steps are being used:

1. Initialize random values to arrays \(a, b\) from 0 to \(n\).
2. Calculate \(X_0\) and send value of \(X_0\) from process \(P_0\) to \(P_1\).
3. Receive \(X_0\) from \(P_0\) and send this \(X_0\) to \(P_2\) and then compute \(X_1\) and send this to \(P_2\).
4. Receive \(X_0\) and \(X_1\) from \(P_1\) and send this to \(P_3\), then compute \(X_2\) and send it to \(P_3\).
5. Receive \(X_0, X_1\) and \(X_2\) from \(P_2\) and calculate \(X_3\) to get the resultant \(X_0 X_1 X_2 X_3\).
6. Repeat the steps 1 to 5 for the problem size (set of equations) taken in the graph.
Fig. 3(c) and 3(d) shows the predicted execution time (in seconds) and percent error of actual times for the Linear Equation Solver program respectively. The predictions generated by the APAPS system are quite satisfactory, and the prediction error is between 10 to 15%.

9.4.1. Performance Debugging

The APAPS system can be used to characterize programs for performance debugging purposes. For example, Fig. 4 shows the percentage of time spent in computation and communication 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.

![Graph Showing the Percentange Error](image)

Fig. 3(a): Execution time for matrix multiplication Predicted by APAPS.

Fig. 3(b): % Error of the actual time for Matrix Multiplication Program.
Fig. 3(c): Execution time for Linear Equation Solver Predicted by APAPS.

Fig. 3(d): % error of the actual time for Linear Equation Solver.
9.5. CONCLUSIONS AND FUTURE WORK

In this chapter we studied performance predictions for parallel scientific applications on homogeneous cluster of workstations. Performance prediction is important for analyses of the scalability of parallel applications and the estimation of the processing time for the application in loaded/unloaded environment. We developed Automatic Parallel Application Prediction System (APAPS) for cluster computing environment. Here we are reporting the accuracy of the APAPS using two scientific applications. The measured result shows that APAPS have high prediction accuracy.

In conclusion some of the experimental results obtained with the APAPS performance prediction system for MPI programs were shown. The results were shown by with two parallel applications, Matrix Multiplication and Linear
Equation Solver using MPI with the system showing good performance with lower error rate. In the future, further studies can be made to improve the accuracy of the system and refine the performance models as necessary. APAPS system can be used, 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, the developed system can be used as a basis for performance debugging for MPI programs. The APAPS system allows performance models to be constructed for each basic block of a MPI program. This information can be used to explore the predicted program performance of different parts of an application for varying system configurations.
REFERENCES