3-18-1992

Mapping Programs to Parallel Architectures in the Real World

Dezheng Tang
Portland State University

Let us know how access to this document benefits you.
Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds
Part of the Electrical and Computer Engineering Commons

Recommended Citation

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.
Mapping an application program to a parallel architecture can be described as a multidimensional optimization problem. To simplify the problem, we divide the overall mapping process into three sequential substeps: partitioning, allocating, and scheduling, with each step using a few details of the program and architecture description. Due to the difficulty in accurately describing the program and architecture and the fact that each substep uses incomplete information, inaccuracy is pervasive in the real-world mapping process. We hypothesize that the inaccuracy and the use of suboptimal, heuristic mapping methods may greatly affect the mapping or submapping performance and lead to a non-optimal solution.
We do not discard the typical approach used by most researchers in which total execution time or speedup is the criterion to evaluate the quality of the mapping. However, we improve on this approach by including the effects of inaccuracy. We believe that, due to the presence of inaccuracy in the mapping process, investigating the impact of inaccuracy on the mapping quality is crucial to achieving good mappings. The motivation of this work is to identify the various inaccuracies during the mapping procedure and explore the sensitivity of mapping quality to the inaccurate parameters.

To conduct the sensitivity examination, the Global Cluster partitioning algorithm and some models were used. The models use some program and architecture characteristics, or lower-level meters, to characterize the mapping solution space. The algorithm searches the solution space and makes the decision based on the information provided by the models. The experiments were implemented on a UNIX LAN of Sun workstations for different data flow graphs. The graphs use three parallel programming paradigms: fine-grained, coarse-grained, and pipelined styles, to represent some high-level application programs: vector inner product calculation, matrix multiplication, and Gaussian elimination respectively. The experimental results show that varying system behavior affects the accuracy of lower-level meters, and the quality of the mapping algorithm is very sensitive to the inaccuracies.
MAPPING PROGRAMS TO PARALLEL ARCHITECTURES IN THE REAL WORLD

by

DEZHEN TANG

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1992
TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of Dezheng Tang presented March 18, 1991.

Michael A. Driscoll, Chair

W. Robert Daasch

Warren Harrison

APPROVED:

Rolf Schaumann, Chairman, Department of Electrical Engineering

C. William Savery, Interim Vice Provost for Graduate Studies and Research
I would like to express my sincere thanks to Dr. Michael Driscoll, my advisor, for providing invaluable guidance and constant encouragement throughout my graduate studies at Portland State University. His ability to grasp key aspects of complex problems has greatly influenced my approach to research. His gracious criticism of my writing has been educational and his enthusiasm for research has been inspirational. I am indebted to him for the long hours he has spent to revise my thesis. I am also very grateful to Dr. Robert Daasch and Dr. Warren Harrison for giving their precious and sensible suggestions for improving my thesis.

I would like to extend my appreciation to the fellow graduate students, Jingsong Fu, Kiswanto Thayib, Chintamani Patwardhan, Satish Maruti Pai, and Liono Setiowijoso, who shared the unusual ideas and delightful success in the Parplum project. Many thanks also go to the faculty and staff of the Electrical Engineering department who were always willing to help me whenever I needed it.

I would also like to thank two special friends in my life, Lowell Ericsson and Nancy Ericsson, for helping make my dream come true. Without their encouragement and generous support, this thesis would not have been possible. I am indebted to them for all this.

Finally, a special gratitude and respect goes to my parents, Xiaoxuan Tang and Junrong Shen, for enabling me to be more than I thought possible. Their patience, endurance, and their own academic achievement always provide me with motivation through various endeavors.
# TABLE OF CONTENTS

| ACKNOWLEDGEMENTS                                      | iii  |
| LIST OF TABLES                                       | vi   |
| LIST OF FIGURES                                      | vii  |
| LIST OF SYMBOLS AND EQUATIONS.                       | ix   |

## CHAPTER

### I INTRODUCTION .................................................. 1

I.1 Research Motivation. ........................................ 1

I.2 Key Research Focus .......................................... 3

I.3 Our Interests and Objectives ............................... 5

I.4 Thesis Overview ............................................. 6

### II MAPPING METHODOLOGY AND SENSITIVITY ............... 7

II.1 Overview of Mapping Problem .............................. 7

II.2 Background .................................................. 8

II.3 General Definitions and Notations ....................... 10

II.4 Inaccuracies and Their Impact on Mapping ............. 11

II.4.1 Impact of Search Strategy in Mapping

II.4.2 Cost Function and Its Inaccuracy

II.4.3 Impact of $E$ Using $S_O$

II.4.4 Impact of $E$ Using $S_S$

II.4.5 $\eta(P)$ and $\theta(A)$ and Their Inaccuracies

II.4.6 Impact of $\eta(P)$ and $\theta(A)$ on $E$

II.4.7 Impact of $\eta(P)$ and $\theta(A)$ Using $S_O$

II.4.8 Impact of $\eta(P)$ and $\theta(A)$ Using $S_S$

II.5 Summary of Inaccuracies and Sensitivity Analysis ... 23
### III GENERAL APPROACH AND EXPERIMENTAL ENVIRONMENT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III.1 General Approach.</td>
<td>31</td>
</tr>
<tr>
<td>III.2 The Experimental Environment: Parplum</td>
<td>32</td>
</tr>
<tr>
<td>III.3 Search Strategy Description</td>
<td>33</td>
</tr>
<tr>
<td>III.3.1 Mapping at Partitioning Stage</td>
<td></td>
</tr>
<tr>
<td>III.3.2 Global Cluster Partitioning Algorithm</td>
<td></td>
</tr>
<tr>
<td>III.4 Model Description</td>
<td>37</td>
</tr>
<tr>
<td>III.5 Effect of Other Submappings</td>
<td>40</td>
</tr>
<tr>
<td>III.6 Programs and Architectures</td>
<td>41</td>
</tr>
<tr>
<td>III.6.1 Program Description</td>
<td></td>
</tr>
<tr>
<td>III.6.2 Architecture Description</td>
<td></td>
</tr>
</tbody>
</table>

### IV TESTING AND DATA ANALYSIS.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV.1 Overview of Testing Procedure</td>
<td>49</td>
</tr>
<tr>
<td>IV.2 System Stability Testing</td>
<td>50</td>
</tr>
<tr>
<td>IV.3 Sensitivity Analysis</td>
<td>52</td>
</tr>
<tr>
<td>IV.4 General Comments on Sensitivity Testing</td>
<td>63</td>
</tr>
</tbody>
</table>

### V CONCLUSION AND FUTURE WORK.

REFERENCES.                                                           69
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Description</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Results of Partitioning with IPC Time Error for Gaussian Elimination Graph</td>
<td>53</td>
</tr>
<tr>
<td>II</td>
<td>Results of Partitioning with Node Execution Time Error for Gaussian Elimination Graph</td>
<td>56</td>
</tr>
<tr>
<td>III</td>
<td>Results of Partitioning with IPC Time Error for Matrix Multiplication Graph</td>
<td>59</td>
</tr>
<tr>
<td>IV</td>
<td>Results of Partitioning with IPC Time Error for Vector Inner Product Graph</td>
<td>61</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>18.</td>
<td>Execution Time for Matrix Multiplication Graph.</td>
<td>60</td>
</tr>
<tr>
<td>19.</td>
<td>Result of GC Algorithm for Vector Inner Product Graph.</td>
<td>62</td>
</tr>
<tr>
<td>20.</td>
<td>Execution Time for Vector Inner Product Graph</td>
<td>62</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS AND EQUATIONS

\( \eta(P) \) A vector of program characteristics (lower-level meters).
\( \theta(A) \) A vector of architecture characteristics (lower-level meters).
\( M \) Entire mapping solution space.
\( m \) A particular mapping in the solution space, \( m \in M \).
\( S \) The search strategy (mapping algorithm).
\( C \) The cost function in the ideal world.
\( C_m \) The cost of \( m \). \( C_m = C(\eta(P), \theta(A), m) \).
\( m_a \) A mapping which is found by \( S \). \( m_a = S(C_m), m_a \in M \).
\( Q \) The quality of mapping \( m_a \). \( Q = C(\eta(P), \theta(A), m_a) \).
\( S_O \) The optimal search strategy.
\( S_S \) The suboptimal search strategy.
\( m_O \) The optimal mapping found by \( S_O \) using \( C \). \( m_O = S_O(C_m) \).
\( m_S \) A mapping found by \( S_S \) using \( C \). \( m_S = S_S(C_m) \).
\( C_{m_o} \) The cost of \( m_O \). \( C_{m_o} = C(\eta(P), \theta(A), m_O) \).
\( Q_O \) The quality of \( m_O \). \( Q = C(\eta(P), \theta(A), m_O) \).
\( Q_S \) The quality of \( m_S \). \( Q_S = C(\eta(P), \theta(A), m_S) \).
\( \Delta Q_S \) The error caused by \( S_S \) using \( C \). \( \Delta Q_S = Q_S - Q_O \).
\( E \) The cost estimator (model).
\( E_m \) The estimate of \( C_m \). \( E_m = E(\eta(P), \theta(A), m) \).
\( \delta_E \) The error caused by \( E \). \( \delta_E = E_m - C_m \).
\( m_{EO} \) A mapping found by \( S_O \) using \( E \). \( m_{EO} = S_O(E_m) \).
\( Q_{EO} \) The quality of \( m_{EO} \). \( Q_{EO} = C(\eta(P), \theta(A), m_{EO}) \).
\(\Delta Q_{EO}\) The error caused by \(\delta_E\) using \(S_O\). \(\Delta Q_{EO} = Q_{EO} - Q_O\).

\(\frac{\partial \Delta Q_{EO}}{\partial \delta_E}\) The sensitivity of \(\Delta Q_{EO}\) to \(\delta_E\).

\(m_{ES}\) A mapping found by \(S_S\) using \(E\). \(m_{ES} = S_S(E_m)\).

\(Q_{ES}\) The quality of \(m_{ES}\). \(Q_{ES} = C(\eta(P), \theta(A), m_{ES})\).

\(\Delta Q_{ES}\) The error caused by \(\delta_E\) and \(S_S\). \(\Delta Q_{ES} = Q_{ES} - Q_O\).

\(\frac{\partial \Delta Q_{ES}}{\partial \delta_E}\) The sensitivity of \(\Delta Q_{ES}\) to \(\delta_E\).

\(\hat{\eta}(P)\) Inaccurate description of \(\eta(P)\).

\(\hat{\theta}(A)\) Inaccurate description of \(\theta(A)\).

\(\delta_\eta\) The error in \(\hat{\eta}(P)\). \(\delta_\eta = \hat{\eta}(P) - \eta(P)\).

\(\delta_\theta\) The error in \(\hat{\theta}(A)\). \(\delta_\theta = \hat{\theta}(A) - \theta(A)\).

\(\delta_{\eta\theta}\) The error caused by \(E\), \(\hat{\eta}(P)\), and \(\hat{\theta}(A)\).

\(\delta_{\eta\theta} = E(\hat{\eta}(P), \hat{\theta}(A), m) - C_m\).

\(\delta_{\eta\theta}\) The error in \(E\) which is caused only by \(\delta_\eta\) and \(\delta_\theta\).

\(\frac{\partial \delta_{\eta\theta}}{\partial \delta_\eta}\) The sensitivity of \(\delta_{\eta\theta}\) to \(\delta_\eta\).

\(\frac{\partial \delta_{\eta\theta}}{\partial \delta_\theta}\) The sensitivity of \(\delta_{\eta\theta}\) to \(\delta_\theta\).

\(m_{\eta\theta EO}\) A mapping found by \(S_O\) using \(E\), \(\hat{\eta}(P)\), and \(\hat{\theta}(A)\).

\(m_{\eta\theta EO} = S_O(E(\hat{\eta}(P), \hat{\theta}(A), m))\).

\(Q_{\eta\theta EO}\) The quality of \(m_{\eta\theta EO}\). \(Q_{\eta\theta ES} = C(\eta(P), \theta(A), m_{\eta\theta EO})\).

\(\Delta Q_{\eta\theta EO}\) The error caused by \(\delta_\eta\), \(\delta_\theta\), and \(\delta_E\). \(\Delta Q_{\eta\theta EO} = Q_{\eta\theta EO} - Q_O\).

\(\frac{\partial \Delta Q_{\eta\theta EO}}{\partial \delta_E}\) The sensitivity of \(\Delta Q_{\eta\theta EO}\) to \(\delta_E\).

\(\frac{\partial \Delta Q_{\eta\theta EO}}{\partial \delta_\eta}\) The sensitivity of \(\Delta Q_{\eta\theta EO}\) to \(\delta_\eta\).

\(\frac{\partial \Delta Q_{\eta\theta EO}}{\partial \delta_\theta}\) The sensitivity of \(\Delta Q_{\eta\theta EO}\) to \(\delta_\theta\).
\[ m_{\eta \theta ES} \] A mapping found by \( S_S \) using \( E, \hat{\eta}(P) \), and \( \hat{\theta}(A) \).

\[ m_{\eta \theta ES} = S_S ( E ( \hat{\eta}(P), \hat{\theta}(A), m ) ). \]

\[ Q_{\eta \theta ES} \] The quality of \( m_{\eta \theta ES} \). \( Q_{\eta \theta ES} \equiv C ( \eta(P), \theta(A), m_{\eta \theta ES} ) \).

\[ \Delta Q_{\eta \theta ES} \] The error caused by \( \delta_{\eta}, \delta_{\theta}, \delta_E \), and \( S_S \). \( \Delta Q_{\eta \theta ES} = Q_{\eta \theta ES} - Q_O \).

\[ \frac{\partial \Delta Q_{\eta \theta ES}}{\partial \delta_E} \] The sensitivity of \( \Delta Q_{\eta \theta ES} \) to \( \delta_E \).

\[ \frac{\partial \Delta Q_{\eta \theta ES}}{\partial \delta_{\eta}} \] The sensitivity of \( \Delta Q_{\eta \theta ES} \) to \( \delta_{\eta} \).

\[ \frac{\partial \Delta Q_{\eta \theta ES}}{\partial \delta_{\theta}} \] The sensitivity of \( \Delta Q_{\eta \theta ES} \) to \( \delta_{\theta} \).
CHAPTER I

INTRODUCTION

1.1 RESEARCH MOTIVATION

Due to the demand for fast computers for complicated computational problems in applications such as robotics and space technology, distributed computing systems have become more and more attractive and important in recent years. This is because the required processing capacity for many computational problems cannot be achieved on a single processor. A distributed system provides the facility for utilizing remote computer resources and parallel processing to improve throughput. Furthermore, it increases the flexibility, efficiency, and reliability [1].

Good mapping of programs to distributed systems is crucial to maximize the efficiency of their use. A serious problem that limits the efficiency of parallel processing is the degradation in speedup caused by the saturation effect [2]. We would expect the speedup to increase linearly as the number of processors increases. However, in practice, because of the data and control signals transferred from one processor to another, also known as inter-processor communication (IPC), and unbalanced loading in the distributed system, the speedup begins to decrease with each additional processor after the number of processor increases to a certain point.

The purpose of mapping is to decompose the application program into smaller components and allocate them to the multiprocessor system in such a way that the components can be executed simultaneously and the total execution time of the program can
Mapping programs to target architectures to improve execution time is a complicated problem. First, in most cases, no optimal solution can be found in polynomial time. Thus, some simple, efficient heuristic approaches are suggested for dealing with most of the mapping problems [3][4]. Second, the diversity of programs and architectures makes finding general mapping procedures and evaluating them extremely difficult. To balance the conflicts between optimality and generality is the fundamental goal of most researchers [5][6].

In the past, most approaches have ignored one or more key factors during the mapping process and have dealt with the mapping problems for some special performance requirement or application environment [7][8]. Their approaches can not deal with a wide range of programs and target architectures. To evaluate the quality of mapping, most researchers have used execution time of a program on a target architecture [9][10] or some benchmark mapping algorithms [11] as the criteria. The best solution is usually determined by comparing the current mapping algorithms with these benchmark algorithms. However, even a benchmark algorithm using an "optimal" mapping method may not guarantee an optimal solution because of the diversity of programs and architectures. In addition, many approaches can map application programs to architectures, but they only deal with some subset of the mapping problem.

To deal with the complexity of the problem while still exploiting fine-grained parallelism, we use three steps, including:

1. investigating program and architecture characteristics at the level of fine-grained parallelism, since coarse-grained program and architecture characteristics hide the possible parallelism inside the program and architecture and make the load balancing problem difficult to solve.
(2) subdividing the overall mapping process into several sequential steps to simplify the complex mapping problem.

(3) exploring accurate models of system performance, or cost functions, for mapping search decision making and evaluation. It is obvious that a good characterization of the mapping solution space is a basic requirement for obtaining good mappings.

We divide the overall mapping procedure into three substeps: partitioning, allocating, and scheduling. The partitioning step makes use of fine-grained program structure and coarse estimates of the architecture description to partition the program into a set of modules. It is primarily concerned with minimizing data dependence and balancing the load among partitioned modules. The allocating step assigns the partitioned modules to processors. It uses the details of the architecture description but ignores the fine-grained structures of the program to minimize the IPC overhead and improve the efficiency of resource utilization. The scheduling step orders operations inside each processor so that the precedence relationship among the operations assigned to a processor can be exploited and data dependency among the operations in different processors will be minimized. (Most previous works can be cast into one or more of these categories.)

Each step can be viewed as a search strategy making use of a few fine-grained details of the programs or the architectures to search the solution space described by some models to find a particular mapping to optimize some aspect of performance.

1.2 KEY RESEARCH FOCUS

Due to the fact that each submapping procedure simplifies the problem by ignoring some details and because of the difficulty in accurately generalizing the features of the programs and architectures, a problem arises in that the incomplete or inaccurate mapping information may affect the mapping quality, which in turn makes the evaluation
of the mapping or submapping quality extremely difficult. The inaccuracy comes from two main sources:

(1) The inaccuracies in describing the program and architecture characteristics. For example, the data dependence constraints inside the program and the system behavior are difficult to characterize. Thus, errors in characterizing these parameters will occur.

(2) The exact model of system performance is hard to find. A model which only estimates the cost of mapping may thus be inaccurate. The model making use of inaccurate program and architecture characteristics will also be in error.

In a word, we view inaccuracy as pervasive in the entire mapping process. We hypothesize that the inaccuracy is a first order effect in the mapping process, i.e., it may seriously affect the performance of the mapping. We identify several sources of non-optimality:

(1) We believe that each submapping, ignoring some details of the program and architecture description, may result in non-optimal solution.

(2) Even the optimal mapping or submapping procedure making use of inaccurate models may not guarantee an optimal solution.

(3) For each mapping or submapping procedure, the use of heuristic techniques may lead to non-optimal solution.

(4) The combination of each optimal submapping may lead to a non-optimal overall mapping.

(5) Most of the distributed systems are user-sharing systems. The variation of the system load may affect the accuracy of the performance measurement.

Every inherent inaccuracy in the mapping can contribute to a non-optimal solution. However, the inaccuracy and its effect on the mapping performance have not drawn
much attention in the literature. A few papers presented the impact of the programs [3] and the impact of the systems [12] on mapping performance. But no work has examined the inaccuracies in the model and the impact of inaccurate models on the mapping performance. Our approach to solving mapping problems differs from previous works in at least one key aspect. In addition to evaluating mapping quality by using execution time, we focus on evaluating the sensitivity of the mapping quality to the inaccuracies. We are concentrating on finding key factors which may impact mapping quality rather than optimal mapping methods.

1.3 OUR INTERESTS AND OBJECTIVES

To prove the hypothesis that inaccuracies during the mapping process may affect the evaluation of the mapping quality, our major interest is to explore the key factors of non-optimal solutions in the mapping such as the program and architecture characteristics, the models, and the search strategies, to investigate the effects of ignoring details of program and architecture characteristics at various points during the mapping process, and to examine the sensitivity of the mapping performance to the inaccuracies of these key factors.

The evaluation of the sensitivity of mapping quality brings several benefits. First, it can identify the key factors on which mapping quality depends. Second, it can help to improve the accuracy of the models for the mapping evaluation. Finally, it will identify some mapping methods which are insensitive to inaccurate models. Our ultimate goal is to find some general mapping methods which are relatively insensitive to various inaccuracies, and thus can find good mappings for certain classes of programs and architectures under real world constraints.

Due to the complexity of quantifying the inaccuracy inherent in the mapping problem and validating our approach, the research presented here will focus on the
following objectives:

1. Identify the key factors upon which the mapping procedure may depend.

2. Test system run-time behavior in Parplum [13] to show the existence of the inaccuracies.

3. Evaluate the sensitivity of mapping quality to some inaccurate parameters via the Parplum system to prove our hypothesis.

I.4 THESIS OVERVIEW

Chapter II presents the history of mapping programs to architectures and sensitivity analysis in the mapping process, discusses the parameters which may impact the quality of mapping and the relationship among these parameters, explains why the inaccuracies in these parameters may lead to non-optimal solutions in mapping, and emphasizes the importance of sensitivity analysis in the mapping evaluation. Chapter III describes our approach to the sensitivity problem and introduces the experimental environment, the construction of the mapping method, the cost functions or models which are used in the mapping method, the programs, and the architectures in the Parplum system. The design of mapping methods focuses on a heuristic approach, the Global Cluster partitioning algorithm [14]. Chapter IV shows the experimental results of inaccuracies in describing the architecture characteristics. It also demonstrates the sensitivity of the mapping quality to these inaccuracies. The final chapter discusses the goals which have been achieved and future work which is planned.
CHAPTER II

MAPPING METHODOLOGY AND SENSITIVITY

II.1 OVERVIEW OF MAPPING PROBLEM

Mapping programs to target architectures can be viewed as a multidimensional optimization problem. Multidimensional refers to the variables which characterize the programs and the architectures, the various mapping methods, and the various criteria for the evaluation of the mapping quality.

The program characteristics include: the grain size of the program, the precedence relation between the instructions in the program, the data types, the operation types, the loop iteration count, etc. The instructions of the program can be represented as the functional nodes in a dataflow graph (DFG) and the immediate precedence relation between the instructions can be represented as the arcs between the nodes. The architecture characteristics include: the number of processors, the memory limit, the topology of the communication links among the processors, the processor speed, the communication speed between the processors, etc.

Section II.2 discusses the background of the mapping process and points out the deficiencies of current approaches. Section II.3 presents the definitions and the notations of the mapping methods and the mapping quality. Section II.4 starts with the discussion on the mapping problem in the ideal world, and then introduces the non-optimal solution caused by the suboptimal search strategy, the inaccuracy of the cost functions, and the inaccuracies in describing program and architecture characteristics. It also describes the
sensitivity of the mapping quality to the inaccuracies of various parameters. Section II.5 summarizes the inaccuracies in the mapping and sensitivity problem.

II.2 BACKGROUND

The diversity of programs and architectures makes general mapping methods hard to find, which in turn causes a diversity of evaluation procedures for mapping methods. The common cost function such as minimum execution time can be used in mapping evaluation, but, an "optimal" solution is usually found in a comparative form; i.e., the best solution is selected only from the finite number of mapping methods which have an excellent response time [2][9].

Most previous research has focused on finding a combination of mapping methods, cost functions, program characteristics, and architecture characteristics to obtain good mappings for some programs and architectures. Many of them dealt with one or more submapping problems as defined earlier. These works can be roughly divided into two categories:

(1) Given a program and an architecture, fix one or a few submapping problems (e.g., assuming the architecture structure is ideal and no allocating is necessary.), find a submapping method (e.g., partitioning) using some cost functions (e.g., IPC time, task completion time) to evaluate the mapping quality so that some aspect of cost can be minimized. Shen and Tsai [15] presented a minimax criterion to partition a graph into modules so that the intermodule communication link is minimized and the load in each module is balanced. The work of Girkar [3] described a Branch and Bound partitioning algorithm using a heuristic method and adjusted the cost function for the mapping search for different graph topologies. The work done by Pinter and Wolfstahl [16] is also an example of this category. This approach is prevalent in the past research. Furthermore, some
researchers developed several submapping methods of the same type (e.g., allocating) and identified one submapping method which resulted in the best performance among those submapping methods for certain programs and architectures. Chu, Holloway, et al. [2] presented two approaches, integer 0-1 programming and heuristic techniques, to model the task allocation in the distributed system for different logical interconnection structures in the task. The examples in this category either investigated programs at module level (coarse-grain) [7] and ignored the parallelism inside the module or exploited program topology but ignored the possible mismatch between the program modules' topology and architecture topology [17][18].

(2) Given a program and a few classes of architectures or given a few classes of programs and an architecture, find a mapping method which is a combination of all the submappings, and improve some aspect of performance by minimizing a cost function. The Pre-P project [19] divided the mapping problem into several components: partitioning, layout, and multiplexing, and focused on finding general automatic mappings of algorithms into architectures. El-Rewini and Lewis [8] presented a heuristic scheduler to schedule program modules onto an arbitrary machine topology. This approach examines the impact of either program characteristics or architecture characteristics on the mapping performance and finds good mapping methods for fixed programs and architectures.

A great deal of effort has been expended in creating and improving various mapping methods for different programs and architectures. However, most approaches concentrate on finding an optimal mapping or submapping for a few classes of programs and architectures, and the approach to evaluate the mapping quality is limited to the improvement of the task response time [16][9]. The effects of various parameters during the mapping process and the inaccuracies in characterizing these parameters to the mapping qual-
ity, i.e. the sensitivity of the mapping quality, have not received much attention in the literature. Very few papers considered the various parameters during the mapping process and their impact on performance while evaluating mapping procedures. Iyer and Sholl [20] mentioned the sensitivity of different operational environments to the selected solutions, but no results have been seen in their research. The work by Huang [17] is an exception which focuses on the sensitivity of performance to the application environment as well as the application requirement.

Due to the complexity of the programs and the architectures, the current approach is not sufficient for the evaluation of mapping quality. Improving task response time is the ultimate criterion in evaluation of the mapping quality. Our investigation in the sensitivity of the mapping quality does not reject the current evaluation method but demonstrates the failings of this criterion in the presence of the inaccuracies and explores a new approach in supporting the ultimate goal of improving the mapping performance.

II.3 GENERAL DEFINITIONS AND NOTATIONS

We denote a particular mapping \( m \) of a program \( P \) to an architecture \( A \) as a point in a multidimensional solution space \( M \), i.e., \( m \in M \). The cost \( C_m \) of the mapping (i.e. at point \( m \)) can be defined as:

\[
C_m = C( \eta(P), \theta(A), m )
\]

where \( C \) is the cost function which models the cost of the mapping in the ideal mapping solution space, \( \eta(P) \) represents a vector of program characteristics which can be denoted as:

\[
\eta(P) = (\eta_1, \eta_2, \ldots, \eta_n)
\]

where \( \eta_n \) is the \( n \)th element in the vector \( \eta(P) \).

\( \theta(A) \) represents a vector of architecture characteristics which can be denoted as:
\( \theta(A) = (\theta_1, \theta_2, \ldots, \theta_n) \).

where \( \theta_n \) is the \( n \)th element in the vector \( \theta(A) \).

The optimization problem is to find a point \( m_O \) in \( M \) such that:

\[
C_{m_O} \leq C_m \quad \forall \ m \in M.
\]

In general, the mapping problem can be characterized as: a mapping method takes program characteristics \( \eta(P) \) and architecture characteristics \( \theta(A) \) as inputs, and then searches the solution space using a search strategy \( S \) that is guided by some cost function \( C \) to find the optimal solution. Figure 1 shows the relationship among \( S, C, \eta(P) \), and \( \theta(A) \). The \( C \) is influenced by \( \eta(P) \) and \( \theta(A) \). \( S \) selects a mapping \( m_a \) based on the information provided by the \( C \). Formally, an \( m_a \) can be defined as:

\[
m_a = S(C_m) = S(C(\eta(P), \theta(A), m)).
\]

Thus, the quality of mapping \( m_a \), \( Q \), can be defined as:

\[
Q = C(\eta(P), \theta(A), m_a)
\]

where \( C \) describes the performance of mapping selected by \( S \) in the ideal world. \( Q \) is usually obtained by measuring the run-time of the program on a particular architecture.

**II.4 INACCURACIES AND THEIR IMPACT ON MAPPING**

To validate the existence of error in the mapping process and the necessity of sensitivity analysis, we start with a discussion of the mapping problem in the ideal world and then investigate the various errors occurring during the real-world mapping process.

We assume, in the ideal world, that programs and architectures have complete and accurate descriptions, and the accurate cost function \( C \) can be found making use of these accurate descriptions. The mapping \( m_O \) obtained by optimal search strategy, \( S_O \),
using $C$ is:

$$m_0 = S_O (C_m)$$

The quality of the optimal mapping $m_0$, $Q_0$, is:

$$Q_0 \equiv C_{m_0} = C (\eta(P), \theta(A), m_0).$$

$C_{m_0}$ is the optimal point in the mapping solution space. Thus, it indicates that the optimal search strategy, $S_O$, is the one that is guaranteed to find an optimal solution in the ideal world.
However, due to the difficulty in describing the exact program and architecture characteristics, the investigation of the mapping problem in the ideal world is unrealistic and dangerous. For example, it is hard to generalize the precedence relationship between the modules inside the program when loop structures are used. Consequently, the error in describing program and architecture characteristics can somehow contribute to the evaluation of the mapping quality. Thus, we move our discussion from the ideal world to the real world. Under our definition, a mapping problem in the real world can be denoted as one in which the search strategy, either optimal or suboptimal, makes use of some cost function to search the optimal solution in the solution space. The cost function can be either accurate or inaccurate. In addition, the program and architecture characteristics are not ideal but evaluated through simulation and empirical measurement which may cause error.

The remainder of this section characterizes the non-optimal solution caused by the search strategy, the inaccurate cost function, and the inaccurate program and architecture characteristics, highlights the errors in the cost function and in describing program and architecture characteristics, and then explores the sensitivity of the mapping quality to these errors.

II.4.1 Impact of Search Strategy in Mapping

Most mapping problems are NP-hard, and require the use of heuristic search techniques in the mapping search [9][21]. The heuristic search techniques may or may not search the whole solution space and may not guarantee an optimal solution even if an accurate cost function has been used.

Even though the heuristic technique does not guarantee finding the optimal solution, it is faster, and simpler than the optimal solution technique. It is important that the mapping procedure must execute reasonably quickly because there is a trade-off between the execution cost of the mapping procedure and the execution cost of the program. It
makes little sense to use a search strategy which spends an hour to find an optimal mapping for a program that only takes few minutes to execute with poor mapping. As a result, the heuristic technique is more acceptable to researchers. But, the deviation from the optimal solution caused by the heuristic technique must be investigated.

We define suboptimal search strategy $S_S$ as the one which may or may not find an optimal solution. Let $m_S$ be the mapping obtained by $S_S$ using accurate cost function $C$, giving:

$$m_S = S_S (C_m).$$

Then, the quality of mapping using $S_S$ is:

$$Q_S = C(\eta (P), \theta (A), m_S).$$

We characterize the error from the optimal solution, $\Delta Q_S$, caused by the $S_S$ as:

$$\Delta Q_S = Q_S - Q_O.$$

A key research topic is to find the $S_S$ such that $\Delta Q_S$ is low.

Heuristic search techniques require prior knowledge of the cost function [2][9]. Finding an accurate cost function usually is not practical because of the difficulty in characterizing the features of the programs and the architectures.

II.4.2 Cost Function and Its Inaccuracy

For several reasons, the cost function is one of the major factors which affect the mapping quality. First, the performance of the mapping is evaluated by the cost function. Second, the performance of the mapping is subject to the search strategy. The mapping search strategy needs the cost function to guide the search. The cost function gives a quantitative description of the mapping solution space and characterizes the constraints in mapping such as the task precedence relationship, memory limit, etc. Commonly used cost functions are the execution time of the program and the resource utilization.
However, the exact value of a cost function is hard to determine. For example, the actual execution time of the module in a program is impossible to obtain before this module is allocated to a processor. Instead some secondary cost functions are often substituted.

There are two types of secondary cost functions in common use: cost estimators and cost indicators. The cost estimator uses characteristics of the mapping to estimate the actual cost function. The total execution time of a program on a target architecture, for example, can be estimated as the execution time of the partitioned module on the most heavily loaded processor. The cost indicator provides information somehow correlated to the cost function. A lower ratio of the communication overhead to the computation cost, for example, may indicate higher speedup. In the following, the cost estimator $E$ will be denoted as:

$$E_m = E (\eta(P), \theta(A), m).$$

The error of $E_m$ compared with the actual cost $C_m$ for a particular mapping $m$, $\delta_E$, can be characterized as:

$$\delta_E = E_m - C_m.$$  

Various cost estimators have been used in mapping algorithms. Girkar [3] modeled the total computation time of the program on the system as the sum of the computation time of the module on the most heavily loaded processor and the total IPC time among the processors. He used this model in his Branch-Bound heuristic partitioning algorithm. He also pointed out the two criticisms of this model. The first is that the model ignores the dependence constraints among the modules. The second is the fact that the model fails to account for the possible overlap of the communication time with the computation time. Indurkhya and Stone [22] demonstrated several estimated models for the optimal task-assignments of randomly generated distributed program.
To improve the accuracy of the cost function, researchers evaluate the cost function through simulation or empirical measurement. Driscoll [9] describes the execution time as a function of several variables, including the mapping methods, the number of processors, the number of functional units with different function type, the execution time for each functional unit, the global communication time, the local communication time, and a vector of functions describing the characteristics of the program.

The error in the cost function may contribute to the evaluation of the mapping quality. The next two sections discuss the impact of error in cost function on the mapping quality.

II.4.3 Impact of $E$ Using $S_O$

The cost estimator, $E$, estimates the mapping solution space. Thus, finding the optimal mapping solution may not be guaranteed even though the optimal search strategy $S_O$ has been applied.

A mapping $m_{EO}$ obtained by $S_O$ using $E$ is:

$$m_{EO} = S_O (E_m) = S_O (C_m + \delta_E)$$

The quality of mapping $Q_{EO}$ becomes:

$$Q_{EO} = C (\eta (P), \theta (A), m_{EO}) = C (\eta (P), \theta (A), S_O (C_m + \delta_E)).$$

We define $\Delta Q_{EO}$ as the error in the mapping quality caused by $E$ using $S_O$:

$$\Delta Q_{EO} = Q_{EO} - Q_O.$$

The notation suggests that the error in $E$ contributes to the mapping quality. Our interest is to find those $S_O$, such that $\Delta Q_{EO}$ is small and $\Delta Q_{EO}$ is insensitive to $\delta_E$. It can be characterized as:
\[
\frac{\partial \Delta Q_{EO}}{\partial \delta_E} \to 0 \quad \text{and} \quad \Delta Q_{EO} \to 0.
\]

\(\Delta Q_{EO} \to 0\) is the obvious goal of the researchers [3]. Based on this notation, we are also looking for \(\frac{\partial \Delta Q_{EO}}{\partial \delta_E} \to 0\), i.e., for search strategies that are less affected by the inaccuracy of \(E\).

Figure 2 shows the expected impact of \(\delta_E\) on \(\Delta Q_{EO}\), which means that the best solution has been found by using accurate cost estimates. While \(\delta_E\) changes, the error of mapping quality \(\Delta Q_{EO}\) may also change. Thus, in the presence of inaccuracy in \(E\), even \(S_O\) does not guarantee finding the optimal solution. \(\Delta Q_{EO}\) may or may not increase while \(\delta_E\) increases depending on the characteristics of the program and architecture. To obtain \(\Delta Q_{EO} \to 0\), we need to find \(E\) so that \(\delta_E = 0\). But, if we can identify that \(\frac{\partial \Delta Q_{EO}}{\partial \delta_E} \to 0\), we may just quickly use an estimate instead of consuming time to find the exact \(C\). The cost estimate may be used by the mapping search at various points. Thus, the precise estimate is expensive for the mapping process. The sensitivity analysis can help us to identify the tradeoff between time and accuracy. A good search strategy must be the one in which the quality of mapping is less affected by the inaccurate cost function.

**II.4.4 Impact of \(E\) Using \(S_S\)**

We define \(m_{ES}\) as the mapping obtained by \(S_S\) using \(E\):

\[
m_{ES} = S_S \left( E_m \right)
\]

\[
= S_S \left( C_m + \delta_E \right)
\]

The quality of mapping \(Q_{ES}\) can be characterized as:

\[
Q_{ES} \equiv C \left( \eta \left( P \right), \theta \left( A \right), m_{ES} \right)
\]

\[
\equiv C \left( \eta \left( P \right), \theta \left( A \right), S_S \left( C_m + \delta_E \right) \right).
\]
The error of the mapping quality $\Delta Q_{ES}$ is:

$$\Delta Q_{ES} = Q_{ES} - Q_O$$

The source of non-optimal solutions come from two components: the cost estimator $E$ and the suboptimal search strategy $S_S$. It is hard to examine the exact $\Delta Q_{ES}$ because of the difficulty in characterizing the correlation between $\Delta Q_{EO}$ and $\Delta Q_S$. Thus, we isolate each individual inaccurate source and identify which one is the critical source affecting the mapping quality $Q_{ES}$. To examine the effect of inaccurate sources to the $Q_{ES}$, we use crude estimates by varying one inaccurate parameter at a time, i.e., test $\Delta Q_{EO}$ by using $S_O$ and $E$ and test $\Delta Q_S$ by using $S_S$ and $C$. The overall error of the mapping quality $\Delta Q_{ES}$ is thus approximated as two independent parts:

$$\Delta Q_{ES} = \Delta Q_{EO} + \Delta Q_S$$

It is worth noting that $\Delta Q_{ES}$ may or may not be greater than $\Delta Q_{EO}$, i.e. $S_O$ using $E$ may not guarantee a better performance than $S_S$ using $E$. This is because $S_O$ may be more affected by $\delta_E$ than $S_S$.

To obtain a good mapping for a particular mapping problem, revising $S_S$ is a
possible solution. However, as long as the error exists, it may not be the way to solve the general problems. Thus, we ignore the possible non-optimal solution by using $S_S$ and only concentrate on examining the impact of $\delta_E$ on $\Delta Q_{ES}$. A good mapping should be:

$$\frac{\partial \Delta Q_{ES}}{\partial \delta_E} \rightarrow 0 \quad \text{and} \quad \Delta Q_{ES} \rightarrow 0.$$  

As in II.4.3, we are looking for not only a good combination of $E$ and $S_S$ but also for those $S_S$ which are insensitive to the inaccurate $E$.

**II.4.5 $\eta(P)$ and $\theta(A)$ and Their Inaccuracies**

The vectors of the program characteristics $\eta(P)$ and the architecture characteristics $\theta(A)$ are the two fundamental elements in the mapping process. The performance of mapping is directly affected by the inaccuracies of $\eta(P)$ and $\theta(A)$.

To analyze the inaccuracies of $\eta(P)$ and $\theta(A)$ quantitatively, we define some lower-level meters to estimate $\eta(P)$ and $\theta(A)$. We denote $\delta_\eta$ and $\delta_\theta$ as the vectors of error to $\eta(P)$ and $\theta(A)$ measured in the real world. We have:

$$\delta_\eta = \hat{\eta}(P) - \eta(P)$$

$$= (\hat{\eta}_1 - \eta_1, \hat{\eta}_2 - \eta_2, \ldots, \hat{\eta}_n - \eta_n)$$

$$= (\delta_{\eta_1}, \delta_{\eta_2}, \ldots, \delta_{\eta_n})$$

where $\hat{\eta}(P)$ is the estimate of $\eta(P)$, $\hat{\eta}_n$ is the $n$th element in $\hat{\eta}(P)$ which estimates $\eta_n$, and $\delta_{\eta_n}$ is the error in estimating $\eta_n$.

$$\delta_\theta = \hat{\theta}(A) - \theta(A)$$

$$= (\hat{\theta}_1 - \theta_1, \hat{\theta}_2 - \theta_2, \ldots, \hat{\theta}_n - \theta_n)$$

$$= (\delta_{\theta_1}, \delta_{\theta_2}, \ldots, \delta_{\theta_n})$$

where $\hat{\theta}(A)$ is the estimate of $\theta(A)$, $\hat{\theta}_n$ is the $n$th element in $\hat{\theta}(A)$ which estimates $\theta_n$, and $\delta_{\theta_n}$ is the error in estimating $\theta_n$. The inaccuracies of $\eta(P)$ and $\theta(A)$ can be attributed to things like measurement error and system load variation, etc.
The next three sections discuss the impact of $\delta_\eta$ and $\delta_\theta$ on the cost estimator $E$ and on the mapping quality.

11.4.6 Impact of $\delta_\eta$ and $\delta_\theta$ on $E$

The cost estimator $E$ is used to estimate the performance of a mapping of program to architecture. Thus, the accuracy of $E$ depends on the accuracies of the characteristics of the programs and the architectures, i.e., the lower-level meters.

For example, one cost estimator shows that the execution time of a program is the sum of execution time of each functional node. But, for a program with loop structure, the nodes may be executed several times. Thus, the actual execution time of the loop program should be the sum of the execution time of each node inside the loop multiplied by the number of loop iterations plus the sum of the execution time of all the nodes outside the loop. To identify if a given node is in the loop and how many times this node will be executed is always difficult because it depends on the characteristics of the program and input data.

From the architectural aspect, the process load and the network server load will affect the inaccuracy of the lower-level meter. The $E$ making use of inaccurate lower-level meters as input to model the mapping performance, in turn, will be in error.

We characterize the error in $E$ as $\delta_{E\eta\theta}$ caused by the inaccurate $\hat{\eta}(P)$ and $\hat{\theta}(A)$:

$$\delta_{E\eta\theta} = E(\hat{\eta}(P), \hat{\theta}(A), m) - C(\eta(P), \theta(A), m)$$

$$= E(\eta(P) + \delta_\eta, \theta(A) + \delta_\theta, m) - C(\eta(P), \theta(A), m)$$

$$= \delta_E + \delta_{E\eta\theta}.$$

The error of $E$ comes from two components: first, $E$ using estimates instead of the actual value of $C$ causes error. $\delta_E$ is the error in estimating $C$ assuming that $\hat{\eta}(P)$ and $\hat{\theta}(A)$ are accurate. Second, the accuracy of $E$ is affected by the accuracies of $\hat{\eta}(P)$ and $\hat{\theta}(A)$. $\delta_{E\eta\theta}$ is the error in $E$ caused only by $\delta_\eta$ and $\delta_\theta$. When $\delta_\eta$ and $\delta_\theta$ are zero, $\delta_{E\eta\theta}$
becomes zero. The correlation between $\delta_{\eta\theta}$ and $\delta_{E}$ is hard to define because of the complexity of $\eta(P)$ and $\theta(A)$. For some programs and architectures, $\delta_{E}$ may tend to zero. For some other programs and architectures, $\delta_{\eta\theta}$ may tend to zero. It is important to note that $\delta_{\eta\theta E}$ may or may not be worse than either $\delta_{E}$ or $\delta_{\eta\theta}$, and $\delta_{\eta\theta E}$ may depend on some elements of vectors $\delta_{\eta}$ and $\delta_{\theta}$ but not all of them. This approximation aids in the initial exploration of the correlation between the two components of inaccuracy.

To identify an $E$ that is relatively insensitive to inaccurate $\hat{\eta}(P)$ and $\hat{\theta}(A)$, we study the impact of $\delta_{\eta}$ and $\delta_{\theta}$ on $E$ and intend to find $E$ such that:

$$\frac{\partial \delta_{\eta\theta E}}{\partial \delta_{\eta}} \rightarrow 0 \quad \text{and} \quad \frac{\partial \delta_{\eta\theta E}}{\partial \delta_{\theta}} \rightarrow 0$$

and certainly, $\delta_{\eta\theta E} \rightarrow 0$ is also desired.

It indicates that we need to find not only an accurate $E$ comparing with an accurate $C$ but also an $E$ that is less affected by the error of $\eta(P)$ and $\theta(A)$.

II.4.7 Impact of $\eta(P)$ and $\theta(A)$ Using $S_{O}$

Now, assuming $E$ takes $\hat{\eta}(P)$ and $\hat{\theta}(A)$ as inputs to describe the mapping solution space, a mapping obtained by using $S_{O}$ and $E$ is defined as:

$$m_{\eta\theta E O} = S_{O} \left( E \left( \hat{\eta}(P), \hat{\theta}(A), m \right) \right) = S_{O} \left( C_{m} + \delta_{\eta\theta E} \right)$$

Thus, the quality of mapping $Q_{\eta\theta E O}$ using $S_{O}$ is:

$$Q_{\eta\theta E O} = C \left( \eta(P), \theta(A), m_{\eta\theta E O} \right).$$

Then, the error of the mapping quality $\Delta Q_{\eta\theta E O}$ can be characterized as:

$$\Delta Q_{\eta\theta E O} = Q_{\eta\theta E O} - Q_{O}.$$  

$\Delta Q_{\eta\theta E O}$ is similar to $\Delta Q_{E O}$. The difference is that $\Delta Q_{E O}$ is the result of mapping that $S_{O}$ makes use of $E$ assuming $\eta(P)$ and $\theta(A)$ are accurate.
To measure the sensitivity of the mapping quality to the inaccurate sources, we divided the overall procedure into two steps: first, given an $E$, assuming $\delta_E$ is fixed, vary $\delta_\eta$ and $\delta_\theta$, and examine $\Delta Q_{\eta \theta E}$. In practice, we hope for:

\[
\frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_\eta} \to 0, \quad \frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_\theta} \to 0, \text{ and } \Delta Q_{\eta \theta E} \to 0
\]

It indicates that a good mapping method should be insensitive to the inaccuracies in describing program and architecture characteristics.

Second, given $\hat{\eta}(P)$ and $\hat{\theta}(A)$, assuming $\delta_\eta$, $\delta_\theta$ are fixed, vary $\delta_E$, and find $\Delta Q_{\eta \theta E}$. In practice, we hope for:

\[
\frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_E} \to 0 \quad \text{and} \quad \Delta Q_{\eta \theta E} \to 0
\]

It indicates that, even for a given program, a given architecture, and an $S_O$, we need to examine the sensitivity of the mapping quality to inaccurate $E$ so we can use $E$ instead of $C$ while $\frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_E}$ is small.

**II.4.8 Impact of $\eta(P)$ and $\theta(A)$ Using $S_S$**

As in section II.4.7 except using suboptimal search strategy $S_S$ instead of $S_O$, we have:

\[
m_{\eta \theta ES} = S_S \left( E(\hat{\eta}(P), \hat{\theta}(A), m) \right)
\]

\[
= S_S \left( C_m + \delta_{\eta \theta E} \right)
\]

where $m_{\eta \theta ES}$ is the mapping obtained by using $S_S$ associated with $E$, $\hat{\eta}(P)$, and $\hat{\theta}(A)$.

Thus, the quality of mapping $Q_{\eta \theta ES}$ becomes:

\[
Q_{\eta \theta ES} = C(\eta(P), \theta(A), m_{\eta \theta ES})
\]

Then, the overall error of the mapping quality $\Delta Q_{\eta \theta ES}$ is:

\[
\Delta Q_{\eta \theta ES} = Q_{\eta \theta ES} - Q_O
\]


\[ \approx \Delta Q_{\theta E} + \Delta Q_S \]

To measure the sensitivity of the mapping quality using \( S_S \) to the inaccurate sources, the procedure is the same as in \( \Pi.4.7 \). First, given an \( E \) with fixed \( \delta_E \), vary vectors \( \delta_\eta \) and \( \delta_\theta \). We hope for:

\[
\frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_\eta} \rightarrow 0, \quad \frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_\theta} \rightarrow 0, \quad \text{and} \quad \Delta Q_{\eta \theta E} \rightarrow 0.
\]

Second, fix \( \hat{\eta}(P) \) and \( \hat{\theta}(A) \), vary \( \delta_E \). We hope for:

\[
\frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_E} \rightarrow 0 \quad \text{and} \quad \Delta Q_{\eta \theta E} \rightarrow 0
\]

This section defines the mapping problem in the real world, i.e., a suboptimal mapping method makes use of inaccurate cost estimator and inaccurate program and architecture descriptions to find a mapping in the solution space. It contains the overall objective of our research. To validate our approach to the mapping problem, we evaluate the sensitivity of the mapping quality to the inaccuracies of various parameters used during the mapping process. These parameters include cost functions, search strategies, and the vectors of the program and architecture characteristics.

\section*{\( \Pi.5 \) SUMMARY OF INACCURACIES AND SENSITIVITY ANALYSIS}

Summarizing, we have found three major sources of inaccuracy that may alter the mapping quality, which are:

1. The inaccuracy in characterizing program and architecture, \( \delta_\eta \) and \( \delta_\theta \). The errors are primarily from the variation of the system load and the network server load. These errors affect the accuracy of the cost function directly.

2. The inaccuracy in the cost function, \( \delta_{\eta \theta E} \). \( \delta_{\eta \theta E} \) is composed of two parts: \( \delta_E \) and \( \delta_{\eta \theta} \). The boundary between \( \delta_E \) and \( \delta_{\eta \theta} \) is unclear. These errors are the key sources of the non-optimal solution in the mapping process.
(3) The nature of the search strategy. Both optimal search strategy \( S_0 \) and suboptimal search strategy \( S_s \) making use of an inaccurate cost function, \( E \), may not guarantee an optimal solution. The suboptimal, heuristic-based technique may not search the whole solution space and does not guarantee optimality even if an accurate cost function is applied.

Every possible inaccurate source may affect the performance. Figure 3 shows the combination of each possible path to an optimal mapping. We found only one path that can reach to the top of the pyramid with the guaranteed optimal solution \( (\Delta Q = 0) \). That is: \( S_0 \) using accurate \( C \), \( \eta(P) \) and \( \theta(A) \). However, it only happens in the ideal world. Each contribution of error may lead mapping to a non-optimal solution.

An example, see Figure 4, given here shows the impact of inaccurate cost functions and lower-level meters on the mapping performance. In Figure 4.a, a DFG has 5 modules A, B, C, D, E. Assume the execution time of A, B, C, D, E are 5, 5, 10, 5, 5 units respectively, and the communication time between A and D, B and D, D and E is 5 units, between C and E is 10 units. Both communication time and execution time have a 20% error. A critical-path search strategy is applied to find the path where the total execution time is maximum. The total execution time is denoted as the sum of the execution time of the modules and the communication time between the modules on this path. Thus, due to the inaccurate measurement, three possible paths have been found, as shown in Figure 4.b, 4.c and 4.d, if the communication time and execution time have different errors. Which one is the "real" critical path is unknown. This simple example shows that the inaccuracy of the various parameters during the mapping process can have a significant impact on the mapping quality.

Figure 5 gives the details of variables in the overall mapping process. All of these variables are relative to the optimization of the mapping. To evaluate the sensitivity of the mapping quality to all these variables needs a great deal of further research. To sim-
Figure 3. Road to the Optimal Solution.

To simplify our investigation of the problem, we define four sensitivity problems of interest in the real-world mapping process:

1. For some classes of programs and architectures, what is the sensitivity of the inaccurate cost function $\delta_{\eta \theta E}$ to the inaccurate lower-level meters, $\delta_{\eta}$ and $\delta_{\theta}$? It is denoted as $\frac{\partial \delta_{\eta \theta E}}{\partial \delta_{\eta}}$ and $\frac{\partial \delta_{\eta \theta E}}{\partial \delta_{\theta}}$. Note that:

$$\frac{\partial \delta_{\eta \theta E}}{\partial \delta_{\eta}} = \frac{\partial (E(\hat{\eta}(P), \hat{\theta}(A), m) - C(\eta(P), \theta(A), m))}{\partial \delta_{\eta}}$$

$$= \frac{\partial E(\hat{\eta}(P), \hat{\theta}(A), m)}{\partial \delta_{\eta}}$$
because $C(\eta(P), \theta(A), m)$ is defined as the cost function in the ideal world and will not be sensitive to the errors in the parameters it uses, i.e.,

$$\frac{\partial C(\hat{\eta}(P), \hat{\theta}(A), m)}{\partial \delta_{\eta}} = 0.$$  

As for $\delta_{\theta}$, we have:

$$\frac{\partial \delta_{\eta\theta E}}{\partial \delta_{\theta}} = \frac{\partial E(\hat{\eta}(P), \hat{\theta}(A), m)}{\partial \delta_{\theta}}.$$  

(2) For some classes of programs and architectures and a given search strategy, either $S_O$ or $S_S$, what is the sensitivity of $Q_{\eta\theta E O}$, or $Q_{\eta\theta E S}$, to the inaccurate cost function $\delta_{\eta\theta E}$? It is denoted as $\frac{\partial \Delta Q_{\eta\theta E O}}{\partial \delta_{\eta\theta E}}$ and $\frac{\partial \Delta Q_{\eta\theta E S}}{\partial \delta_{\eta\theta E}}$. Note that:
Figure 5. Sources of inaccuracies.
\[
\frac{\partial \Delta Q_{\eta \theta E}}{\partial \delta_{\eta \theta E}} = \frac{\partial (Q_{\eta \theta E} - Q_O)}{\partial \delta_{\eta \theta E}} = \frac{\partial Q_{\eta \theta E}}{\partial \delta_{\eta \theta E}}
\]

because \(Q_O\) can only be found in the ideal world where no error will occur. Thus,

\[
\frac{\partial Q_O}{\partial \delta_{\eta \theta E}} = 0.
\]

The same approach can be used with \(S_S\) to give:

\[
\frac{\partial \Delta Q_{\eta \theta E S}}{\partial \delta_{\eta \theta E}} = \frac{\partial Q_{\eta \theta E S}}{\partial \delta_{\eta \theta E}}
\]

(3) For some classes of programs and architectures and a given search strategy, either \(S_O\) or \(S_S\), what is the sensitivity of \(Q_{\eta \theta E O}\), or \(Q_{\eta \theta E S}\), to the inaccurate lower-level meters \(\delta_{\eta}\) and \(\delta_{\theta}\)? As in problem (1) and (2), the notation can be simplified as:

\[
\frac{\partial \Delta Q_{\eta \theta E O}}{\partial \delta_{\eta}} = \frac{\partial Q_{\eta \theta E O}}{\partial \delta_{\eta}} , \quad \text{and} \quad \frac{\partial \Delta Q_{\eta \theta E O}}{\partial \delta_{\theta}} = \frac{\partial Q_{\eta \theta E O}}{\partial \delta_{\theta}} \quad \text{for} \quad S_O;
\]

\[
\frac{\partial \Delta Q_{\eta \theta E S}}{\partial \delta_{\eta}} = \frac{\partial Q_{\eta \theta E S}}{\partial \delta_{\eta}} , \quad \text{and} \quad \frac{\partial \Delta Q_{\eta \theta E S}}{\partial \delta_{\theta}} = \frac{\partial Q_{\eta \theta E S}}{\partial \delta_{\theta}} \quad \text{for} \quad S_S.
\]

(4) For some classes of programs and architectures and various search strategy \(S\), what is the sensitivity of the \(Q_S\) or \(Q_{ES}\) to the various \(S\) guided by either accurate cost functions \(C\) or estimators \(E\)? (Assuming the error is fixed if \(E\) is used.)

The extensive use of heuristic mapping methods makes sensitivity analysis more and more important. Most researchers confront the problem that the accurate cost function is hard to obtain because of the diversity of the programs and architectures [23][24][25]. Sensitivity analysis remedies the weakness of the traditional approach and exploits the new approach to evaluate the mapping quality systematically. The benefits of sensitivity analysis are many, including:

(1) Quantifying the inaccuracy of the cost estimate used by a particular search strategy to achieve good mappings. The impact of the inaccurate estimates on the
mapping quality can be examined and an appropriate value of cost estimator can be chosen by the search strategy. This approach can help to improve the accuracy of the cost estimators.

(2) Identifying the search strategy which is insensitive to the inaccurate cost estimates. We can ignore the inaccuracy of the cost estimates and use rough estimates to allow mappings to be generated more rapidly.

(3) Identifying the inaccuracies in describing program and architecture characteristics and examining the impact of these inaccuracies on the mapping quality. This can help to find the search strategies that result in good mappings for various programs and architectures.

(4) Examining the inaccuracy in estimating the quality of mapping so the complicated simulation procedure can be omitted.

It is important to note that \( \frac{\partial}{\partial \delta_\eta} \Delta Q_{\eta\Theta ES} \rightarrow 0 \), \( \frac{\partial}{\partial \delta_\Theta} \Delta Q_{\eta\Theta ES} \rightarrow 0 \), or \( \frac{\partial}{\partial \delta_E} \Delta Q_{\eta\Theta ES} \rightarrow 0 \) and \( \Delta Q_{\eta\Theta ES} \rightarrow 0 \) are conflicting goals. One of the primary motivations of sensitivity analysis is to balance the trade-off between time and accuracy and speedup in the mapping process.

At present, our goals are twofold: First, to show the existence of inaccuracies during the mapping process. Second, to prove that the inaccuracies greatly affect the evaluation of the mapping quality and lead to non-optimality.

To examine the inaccuracies of various parameters during the mapping process and evaluate the sensitivity of the mapping quality, the next chapter describes our approach and the experimental environment. An important feature of our system is that it allows experimenters to vary the inaccuracies of the parameters we mentioned before. This feature can help us to evaluate the sensitivity of the mapping quality to these inaccuracies. The Parplum pipeline [13] simplifies conducting our experiments. A heuristic
search method, the Global Cluster partitioning algorithm, makes use of some cost estimates to demonstrate the mapping performance. The creation of the models allows experimenters to select cost estimates to guide the algorithm to a mapping solution. Different fine-grained programs have been created in a scalar DFG form. Our experiments focus on the sensitivity of the partitioning quality using suboptimal search strategy to the inaccurate lower-level meters.
CHAPTER III

GENERAL APPROACH AND EXPERIMENTAL ENVIRONMENT

III.1 GENERAL APPROACH

Due to the complex interrelationship of inaccuracies during the mapping process, our experiments focus on a small portion of the inaccuracies and on investigating the impact of these inaccuracies on mapping performance. We start from observing the inaccuracies in the lower-level meters which are fundamental components during the mapping process. In our scheme, models or cost estimators use the values of lower-level meters as inputs to provide information to the mapping search. Thus, the mapping quality is influenced by the lower-level meters.

We examine two primitive lower-level meters in the architecture characteristics \( \theta(A) \), the execution time of the functional nodes and the interprocessor communication (IPC) time. These two meters give a quantitative description of the major constraint in distributed system, i.e., that load balancing tends to split tasks evenly while minimizing IPC cost tries to merge them into one. Thus, the mapping search strategy may use the information provided by the models and the lower-level meters to decide if the functional nodes in the program should be merged into one processor or should be allocated separately so that the performance can be improved. The execution time of the functional nodes depicts the load information while the IPC time reflects the communication overhead in the system. We measure the execution time of the nodes and the IPC time at run-time and use the average of the measurements as inputs to the mapping process. Note that inaccuracy may be introduced by this measurement process.
To verify our hypothesis that inaccuracy is a primary factor affecting mapping performance, we examine the variation of the mapping quality by adding additional errors to the lower-level meters. For a given program and an architecture, we vary the inaccuracies in one lower-level meter at a time and fix the other possible variables. For instance, we add extra error to the IPC time and use the measured execution time of the nodes and a particular cost estimator. Then, we measure the execution time of the program which is used as the criterion of the mapping quality and observe the impact of the inaccuracies on the mapping quality. Due to the use of a suboptimal, heuristic search strategy \( S_S \) and cost estimator \( E \), in our definition, the sensitivity of the mapping quality \( Q_{\hat{\Theta}E} \) to the inaccurate lower-level meters \( \hat{\Theta}(A) \) we examined can be denoted as

\[
\frac{\partial \Delta Q_{\hat{\Theta}E}}{\partial \delta_\theta}
\]

which is equal to

\[
\frac{\partial Q_{\hat{\Theta}E}}{\partial \delta_\theta}.
\]

Note that \( \delta_1 \) is a constant while \( \frac{\partial \Delta Q_{\hat{\Theta}E}}{\partial \delta_\theta} \) is measured.

To achieve our sensitivity examination, we require: 1) an environment to read the programs and the architecture information into the system, perform mapping, and demonstrate the performance of the mapping at run-time; 2) a mapping search strategy using some models to find a mapping solution; 3) a model which uses the values of the lower-level meters to provide information to the search strategy; and 4) some programs and architectures possessing different parallelism features for testing. In the rest of the chapter, we will give detailed descriptions of the components we used to satisfy the requirements.

III.2 THE EXPERIMENTAL ENVIRONMENT: PARPLUM

To conduct our experiments, an environment, Parplum [13], was used to allow independent control of the programs, the architectures, the models, the search strategies, and the accuracies of various parameters. It is able to map programs to the distributed...
system in such a way that some aspect of performance variation can be seen when some values of the parameters have been changed.

Parplum is composed of a collection of C programs and libraries. It provides three major features to aid in our experiments: First, it is able to handle fine-grained structure programs and architectures. Second, it allows experimenters to select algorithms from extensible submapping libraries and link them together to obtain an overall mapping solution. Finally, it provides different models to estimate the cost of a particular mapping solution. The experimenter can select the models from an extensible model library and introduce errors to the models or to the parameters of the models. To actually execute the mapping, Parplum uses the results from the mapping pipeline [13] to allocate the partitioned program modules to the processors. The performance of the mapping can be measured at run-time by executing the program on the system under the control of an interpreter.

A core library handles the internal data structures of the programs and the architecture information, and manages the coordination among the submappings via some data structures. We translated some high-level language programs into annotated DFGs for testing. The DFG performs the operations at the level of scalar functions, such as additions and multiplications. The architecture used was a UNIX LAN of Sun3 workstations.

III.3 SEARCH STRATEGY DESCRIPTION

To find a good mapping of the program to the architecture, the search strategy should have several characteristics. First, it must take advantage of the fine-grained parallelism in the programs and the architectures. Second, it should be general, so that different programs and architectures can be used. Finally, it should be of low complexity so that the execution of the algorithm can be finished in a reasonable amount of time.
III.3.1 Mapping at Partitioning Stage

We use a heuristic search strategy at the partitioning level. The algorithm partitions the nodes of a DFG into modules based on some knowledge of mapping cost such as the task execution time and the communication requirement. The cost information can be obtained from the lower-level meters and the models. The inaccuracies in the lower-level meters may contribute to the inaccuracies of the models. Thus, the result of the partitioning algorithm may be altered by using the inaccurate meters and models. Since partitioning is the first step in the overall mapping process, exploring the mapping sensitivity problem at the partitioning step can help us to find a good solution at an early stage which will benefit the overall mapping evaluation. Also, partitioning making use of different parallel programs and some estimates of the architecture allows us to find the impact of the diversity of the programs and the inaccurate cost on the mapping performance.

The objective of the partitioning algorithm is to minimize the intermodule communication, exploit potential concurrency in the program, and limit the size of the processes. The Global Cluster partitioning (GC) algorithm [9][14] is based on these characteristics. The GC algorithm takes advantage of the fine-grained parallelism inherent in the program and maps this parallelism using some lower-level meters. It uses typical heuristic strategies to make some assumptions about the process. Given a set of clusters, or modules, the execution time of the cluster, and the intercluster communication time, the heuristic technique compares the execution time of the cluster pairs on the same processor and on the different processors and merges the clusters so that the local intercluster communication cost can be minimized.

III.3.2 Global Cluster Partitioning Algorithm

The GC algorithm is composed of two steps: fine-grain partitioning and global partitioning. Figure 6 illustrates the procedures of the GC algorithm. First, the fine-grain
The global partitioning algorithm partitions the graph into a set of initial clusters so that each cluster contains only one functional node. This is a prerequisite of the global partitioning algorithm, since the global partitioning algorithm transforms a partition of the DFG into a set of clusters with each cluster assigned to an individual processor in the ideal system. This approach allows the search strategy to make use of the details of the program structure and the coarse architecture information. It also prevents the possibility of unbalanced load in the initial partitioning and will be beneficial to the overall procedure. Second, the global partitioning algorithm makes use of the results from the previous step. Some heuristics are used by the global partitioning algorithm to minimize the IPC time and improve the execution time in the ideal system. Heuristic search requires some prior knowledge of cost such as the IPC time and the execution time of a cluster which is provided by the lower-level meters and models. The global partitioning algorithm makes use of the heuristic approach associated with the cost information to search for a good solution. The result of the global partitioning algorithm is a set of clusters which will be allocated to the processors in the real system.

The global partitioning algorithm starts with the clusters created by the fine-grain partitioning algorithm and attempts to merge clusters to improve execution time. Note that, in this situation, each cluster is assigned to an ideal processor. The main loop of the algorithm executes until no improvement can be made by merging cluster pairs. The result of the global partitioning algorithm produces a new set of clusters.

The global partitioning algorithm operates in two phases. One major heuristic is used in each phase. The first heuristic, for a given cluster, uses a distance function to select a candidate cluster responsible for the highest percentage of intercluster communication in the DFG, i.e. the greatest possible IPC cost will be eliminated if this pair is allocated to the same processor. The details of the distance function will be introduced in the next section.
The second heuristic makes use of an execution time model to determine whether a cluster pair should be located in one processor. If the execution time would be improved when a cluster pair is assigned to the same processor, this pair must be merged into one cluster. Otherwise, a new cluster pair will be selected by the first heuristic. The process repeats until all the clusters have been considered. The overall procedure of the global partitioning algorithm [14] is presented here.

Step 1. Choose one cluster from a partition list, and select the candidate cluster which has the largest value of the distance function with the cluster under consideration. If the chosen cluster communicates with no other clusters, set a flag to this cluster, so it will not be chosen again next time. If no candidate cluster is chosen, go back to step 1 to choose the next cluster for consideration, else go to step 2.
Step 2. After a pair of clusters has been identified, the execution time for a given cluster pair will be obtained from the model. If the execution time will improve when this pair are in the same processor, this pair of clusters should be merged together. Go to Step 3. Otherwise, another cluster will be chosen from the partition list for consideration. Go to Step 4.

Step 3. The partition list is updated after two clusters are merged together, and the communication structure of the clusters in the new partition list is also modified, go to Step 4.

Step 4. If the cluster has been chosen before and no candidate cluster communicates with this chosen cluster or only one cluster is left in the reconstructed partition list, return the updated partition list and exit. Otherwise, go to Step 1.

The GC algorithm has many properties that can be applied to some general mapping problems. The use of fine-grained structure program and architecture allows parallelism to be fully exploited. The nature of the two heuristics reduces the complexity of the algorithm because the search limits its attention to a pair of clusters at a time, while ignoring all the other clusters and their precedence relationship in the program. On the other hand, the mapping search deals with clusters and their interconnections but ignores their internal structure. Thus, the complexity is reduced after each iteration that merges a cluster pair. However, note that the heuristics may not search the entire solution space. Therefore, the algorithm may be trapped in local optimality even when accurate models are used.

III.4 MODEL DESCRIPTION

Based on the need of the search strategy, we adopt some models. The models use some lower-level meters as inputs to characterize the mapping performance in the solution space and guide the mapping search.
To accurately characterize the mapping performance, a model should include several pieces of information. First, the model must be able to represent any lower-level parallelism features of mapping the program to the architecture. Second, the model should reflect the constraints on the order of program execution. Third, the model should be general so that it can be applied in different environments or under different requirements. Finally, the values of the model should be quickly generated so that the execution time of the mapping process can be reduced.

In each submapping step, Parplum uses a model library. The model library includes all the information needed to estimate mapping cost including accurate values or estimates of parameters such as the execution time of the partitioned module, the communication links between the processors, the total execution time of the DFG, the number of successors or predecessors of a given node, and the size of the graph (the number of functional nodes in the graph). The models can be in hierarchal form. For example, the total execution time of the DFG can be modeled by some lower-level models such as the execution time of the partitioned modules, the number of modules, and the IPC time.

A distance function between two clusters, or partitioned modules, is introduced here [14]. Assuming cluster $M_k$ communicates with cluster $M_j$ with volume of data $V_{k,j}$ transferred between them. If $M_k$ and $M_j$ also communicate with other clusters in the program, the distance between $M_k$ and $M_j$ is defined as:

$$d(k,j) = \frac{V_{k,j}+V_{j,k}}{\sum_{i=1}^M (V_{k,i}+V_{j,i})}$$

where $M$ is the number of clusters in the program and $d(k,j)$ is the distance function. The numerator of this equation is the sum of the communication cost between $M_k$ and $M_j$. The denominator is the sum of the communication cost between $M_k$, $M_j$ and all the
other clusters in the DFG. The equation compares the costs of two kinds of communications: the communication cost between $M_k$ and $M_j$ and the communication cost from $M_k$ and $M_j$ to the other clusters. The pair of clusters with the highest rating, i.e., shortest distance, indicates that this pair has maximum communication among the clusters in the program. This distance function has been used as the first heuristic of the global partitioning algorithm.

Another model estimates the execution time of a cluster pair in a two-processor system and has been used in the second heuristic of the global partitioning algorithm. The total execution time $T$ of a cluster pair $i$ and $j$ is modeled as:

$$T = \max (T_i, T_j) + T_{\text{comm}}$$

where $T_i$ and $T_j$ are the execution times of cluster $i$ and $j$ respectively. $T_{\text{comm}}$ is the communication time between $i$ and $j$. Thus, the total execution time of a cluster pair on separate processors is the sum of the maximum cluster execution time and the communication time between the clusters assuming that the cluster pair has perfect overlap of computation time and no overlap of the communication time. The total execution time of the cluster pair on the same processor is the sum of the execution time of each cluster, assuming no communication time is needed inside the processor ($T_{\text{comm}} = 0$). $T_{\text{comm}}$ is obtained from the lower-level meter directly. $T_i$ and $T_j$ are estimated as the sum of the execution time of the nodes in the cluster $i$ and $j$ respectively. The execution time of the node is obtained from the lower-level meters. Note that, if a node is in the loop of the program, the actual execution time of the node is estimated as the product of the loop iteration count and the execution time of the node. If two connected nodes in the loop are in different clusters, the intercluster communication time between the nodes is estimated as the product of the loop iteration count and the IPC time retrieved from the meter. It is important to note that this model has its own error because of the assumptions made. In the real world, perfect overlap of the computation time is difficult to obtain. Thus, the
error is inevitable. This model also ignores the data dependence constraints between the clusters.

The distance function and the execution time model used in the GC algorithm are obtained from the model library. In our experiments, we examine the effect of the inaccuracies in the lower-level meters, $\delta_0$, on the quality of partitioning. However, due to the difficulty in characterizing the data dependence constraints, we did not try to characterize the inaccuracy inside the model $\delta_E$ and its impact on the overall mapping performance.

### III.5 EFFECT OF OTHER SUBMAPPINGS

Our major interest focuses on finding the impact of inaccurate lower-level meters on the quality of partitioning. However, the result of the partitioning is a set of modules, and the quality of partitioning can not be evaluated at run-time before the modules are allocated to the multiprocessor system. Thus, some allocating algorithm is needed to allocate the modules to the processors. An important limitation of the partitioning algorithm is that it does not guarantee that the number of modules will be exactly the same as the number of processors. If there are $n+k$ modules but only $n$ processors, the extra $k$ modules have to be reassigned to the processors. This reassignment will alter the quality of partitioning. To minimize the impact of the mismatch between the number of modules and the number of processors, we take advantage of the fully-connected homogeneous network system and use the "modulo" allocating algorithm. If the number of modules is less than the number of processors, the $i$th module goes to the $i$th processor. If the number of modules is greater than the number of processors $N$, the $(N+1)$th module is assigned to the first processor, module $(N+2)$ is assigned to second processor, and so on, until all the modules have been assigned to the processors.

We pointed out the effect of allocating on the evaluation of partitioning quality. Increasing the number of processors is a possible way to minimize the reassignment
effect in allocating. However, this approach can not be applied in our testing because of
the limit of our system. It is obvious that the effect of allocating on the evaluation of par­
titioning quality will be significant if the number of modules is much larger than the
number of processors. Due to the difficulty in characterizing the interaction between the
effect of inaccuracies of the lower-level meter and the effect of the allocating on the
quality of partitioning, we fix the number of processors. Some results of the allocating
effect will be discussed in the next chapter.

No scheduling algorithms have been used in the sensitivity examination. The
goal of scheduling is to explore the precedence relationship within the processor so that
the idle time of functional nodes waiting to be executed is minimized. As the execution
order in our programs is determined by the graph precedence relationship, the priorities
of the nodes in the program are irrelevant to the execution order. Thus, any scheduling
step makes no difference but increases the complexity of the mapping process.

III.6 PROGRAMS AND ARCHITECTURES

III.6.1 Program Description

A DFG description format has been developed to represent our programs for the
evaluation of mapping quality. The directed DFG is translated into our format where
nodes represent the functional units and arcs represent data transfer between the units.
The arrow on the arc shows the data dependence relationship. In data flow computation, a
node can be executed only when all the data arrive at its input arcs. The output arcs
represent the results of the executed nodes.

Some high-level application programs: vector inner product calculation, matrix
multiplication, and Gaussian Elimination, have been represented in three parallel pro­
gramming styles: fine-grained, coarse-grained, and pipelined DFG respectively. All these
Figure 7 is a binary tree graph which represents the calculation of the inner product of 8 element vectors.

Figure 8 is an example showing a 2x2 matrix multiplication DFG. The algorithm for matrix multiplication is described by the following high-level language fragment:

```c
for (i=1; i<=2; i++)
    for (j=1; j<=2; j++)
        for (k=1; k<=2; k++)
            C[i][k] += A[i][j]*B[j][k];
```

The whole graph for 2x2 matrix multiplication computation is composed of two branches, and each branch computes each column of the result matrix C and ends with the END node. The data comes through a stream via a GDATA node which assigns an iteration number to each element for a row in matrix A and a column in matrix B. An ICT node counts how many times a functional unit has been fired. A LGR node compares the
iteration number of the loop and outputs a Boolean signal to control the continuation of the loop execution. A SEL node uses the boolean signal from a LGR node to control the flow of its output. If the signal is FALSE, the input goes to the left of the output arc and exits the loop, otherwise, it goes to the right and continues the loop execution. An INC node increments the iteration number by 1. A DCRE node resets the iteration number for
the next step.

While it is true that the vector inner product is part of the computation of the matrix product, the topologies of these two graphs are totally different. The former is a tree graph with the entire loop in the high-level language representation unwrapped and uses complete fine-grained data flow, while the latter loop-branch graph is only partially unwrapped and uses a coarse-grained loop structure to reduce the size of the graph.

Another application program used in our testing is Gaussian Elimination without pivoting. The linear equations are written in matrix form as a single matrix equation:

\[
\begin{bmatrix}
  a_{11} & a_{21} & a_{31} \\
  a_{21} & a_{22} & a_{32} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix}
=
\begin{bmatrix}
  b_1 \\
  b_2 \\
  b_3
\end{bmatrix}
\]

or simply \( Ax = b \), where \( A \) represents the matrix, \( x \) represents the variables, and \( b \) represents the right-hand side of the equations. The initial step in Gaussian Elimination is to transform all the elements below the diagonal in matrix \( A \) to zero, so all the variables can be obtained by backward substitution. This process can be described by the following fragment in a high-level language:

```c
for ( i = 1; i <= 3; i++ )
  for ( j = i+1; j <= 3; j++ )
    for ( k = 4; k <= i; k-- )
      a[j][k] -= a[i][k] * a[j][i] / a[i][i];
```

To simplify the calculation and avoid division by zero, two assumptions are made here: first, that all the elements in \( A \) are non-zero; second, that a unique solution can be found. The DFG represented in Figure 9 uses a pipelined process. Each branch in the graph has a data stream of all the elements in each row of \( A \) and the element in the corresponding row in vector \( b \). The result of the DFG comes through data streams with some zero values which eliminates the elements below the diagonal of \( A \). The END node
Figure 9. Graph of 3x3 Gaussian Elimination.

is used to terminate the Parplum interpreter but does not reflect the dependence relationship among the pipelines.

The three DFGs have been created using different graph topologies to depict the different characteristics of parallel programming paradigms. The first graph uses a tree graph so the parallelism can be found through fine-grained structure. The second graph
uses the loop branch topology, in which the branch structure shows good parallelism while the loop structure shows coarse-grained structure with heavy data dependence. The third graph gives a pipelined structure and no data dependence among the branches. These examples make it possible to test the impact of the different program characteristics on the quality of mapping.

A high-level program language can be translated into a DFG in several ways. If the graph cannot reflect the fine-grained parallelism, it is hard for the mapping search to generate a good mapping because the data dependence inside the grain cannot be fully exploited. On the other hand, the decrease of the grain size increases the complexity of the mapping algorithm. The improvement of total execution time taking advantage of fine-grained structure may not be seen because of the longer mapping process time. There is a trade-off between the execution time of mapping process using fine-grained program structure and the execution time of a program on a given architecture.

III.6.2 Architecture Description

The experimental environment used in our sensitivity examination is a UNIX LAN of Sun workstations. Our system can be described as a centrally-controlled distributed system with a parent processor which is connected to at most 16 child processors. All the processors are fully connected. The topology of the architecture is showed in Figure 10.

The Parplum interpreter takes advantage of the fully connected distributed system so that the mismatch of topology between the partitioned modules and processors can be ignored. If each module is allocated to one processor and no data is transferred between two modules, no physical connection between these two processors will be set up by the interpreter.

In Parplum, the function of the parent processor is to set up the connections
between itself and all the child processors, read and write the DFG format and the architecture information, along with the results of the mapping or submappings, allocate all the partitioned modules to the child processors based on the results of mapping, and terminate execution after one of the child processors sends a "done" signal back. The function of a child processor is to read the module and some data transfer information from the parent processor, execute each functional node in the module, send data to or receive data from other child processors based on the data transfer information, and send a "done" signal to the parent if the END node in this processor is executed.

The lower-level meters have been defined in the architecture information. The architecture information includes the processor id, the processor name, the number of processors, the execution time of a functional node, the IPC time, etc. Below is the example of our architecture information:

Machine: SUN
Numprocs: 2
proc_name( goofy, huey );
exec_time( ADD, INT, a, b, (0,1));
comm_time( 0, 1, A, B );

Figure 10. Architecture of the Parplum System.
exec_time shown here represents that the execution time of the addition (ADD) node on processors 0 and 1 is equal to $a \times x + b$ where $a$ and $b$ are the parameters obtained from run-time measurement and $x$ is the number of input arcs on this node whose data type is integer (INT). comm_time represents that the IPC time from processor 0 to 1 and is equal to $A \times X + B$ where $A$ and $B$ are the parameters obtained from run-time measurement. $X$ is defined as the message size transferred between the processors.

In the next chapter, we will present the results of our experiment, the inaccuracies in the lower-level meters caused by the system load, and the measurement error. We will also investigate the impact of the inaccuracies in the lower-level meters on the quality of partitioning for different programs.
CHAPTER IV

TESTING AND DATA ANALYSIS

IV.1 OVERVIEW OF TESTING PROCEDURE

To evaluate mapping quality, one commonly used criteria is the execution time of a program on the distributed system. To evaluate the sensitivity of the mapping quality, our approach measures the execution time of the program while inaccuracies in various parameters are introduced in a controlled manner. Our testing concentrates on the source of inaccuracy and the degree of inaccuracy which may influence the mapping quality. The overall testing procedure can be divided into two steps. The first step is measurement error testing. Varying system behavior may produce different results at different times even if the same program, architecture, and mapping method are used. This measurement error may affect the inaccuracies of the lower-level meters directly and also may make mapping evaluation difficult. The second step is the evaluation of the sensitivity of the partitioning quality to the inaccuracies of the lower-level meters for different programs.

The total execution time of program executed using Parplum can be divided into three periods: the setup time to set up the connections among the processors, the parser time to read the program into the system, and the direct execution time to execute all the functional nodes [13]. For a given graph and number of processors, the setup time and the parser time are constants in the ideal situation. Also, they may dominate the overall performance of the mapping while the graph size is small. Thus, the impact of the inaccuracy to the actual mapping performance may not be seen because the variation of the
actual mapping quality is concealed by the inaccuracy of the setup time and the parser time. To minimize the effect of overhead inaccuracy to our sensitivity analysis, in our experiments, we have ignored the setup time and the parser time and only used the direct execution time of the program. The direct execution time of the program is denoted as the time starting from when the program is ready to execute until all the nodes have been executed. In the rest of this chapter, the execution time of the program refers to the direct execution time unless stated specifically.

IV.2 SYSTEM STABILITY TESTING

To implement sensitivity analysis, the inaccuracy of our system timing measurement is the first major concern. Most distributed systems are multi-user systems. The number of users on the system and the number of processes running are unpredictable. The increase of the number of processes will slow down the execution speed of each individual process. Thus, the variation of the system load may affect the accuracy of the performance measurement. All the parameters of the lower-level meters such as the execution time of the functional node and the IPC time are obtained from run-time measurement. Thus, they must be in error. To measure the execution time of each functional node, we take advantage of the fine-grained program structure to insert some timers around nodes, and subtract the timing difference between starting and finishing execution. The timer is a functional node providing the starting time for a given node. Then, we measured the total execution time of a addition (ADD) node on a given processor every half hour with 145 trials. Figure 11 displays the distribution of the measured execution time of an ADD node with two floating data inputs.

As can be seen, even with the same functional node and the same architecture, the measurement of the node execution time has about 10% error from the mean value. The reasons for such fluctuation in the distributed system are various including: the process
Figure 11. Measurement of the Execution Time of ADD Node.

To get the communication time between the processors, we use a two-processor system, build the connection between them, start the timer at $t_s$, send a message from one processor to another $N$ times, then stop the timer at $t_e$. Thus, the average of the IPC time, $T_{comm}$ is:

$$T_{comm} = \frac{(t_s-t_e)}{N}.$$  

Note that in the architecture information defined in the previous chapter, if message size $X$ is zero, i.e., no messages pass between the processors, $T_{comm}$ is zero. Thus, $B$ is zero. We set message size $X$ to one in the testing. Thus, $A$ is equal to $T_{comm}$.

Because of the unpredictable system behavior, all the lower-level meters includ-
ing the execution time of the nodes and the IPC time have been estimated using over one hundred trials, and the parameters for these lower-level meters have been installed in the architecture information archive by averaging the experimental results. To identify the inaccuracy of the lower-level meter, the IPC time was measured once every half hour with 206 trials. Figure 12 shows the distribution of measured IPC times [26]. As can be seen, the IPC time fluctuates as much as 100% from its mean value.

![Figure 12. Measurement of the IPC Time.](image)

**Figure 12.** Measurement of the IPC Time.

### IV.3 SENSITIVITY ANALYSIS

Our sensitivity testing observes the variation of the quality of the GC partitioning algorithm while the magnitude of the lower-level meter changes. We use the GC partitioning algorithm to partition the graph into modules and use the \"modulo\" allocating algorithm to assign the modules to the processors. Three different DFG programs, Gaus-
sian elimination (GE), matrix multiplication, and vector product computation, are used in testing.

The performance of mapping a 12x12 GE graph with 199 nodes on the 4-processor system has been examined while the inaccuracies in the IPC time are varied by adding different values of errors to the measured average. At the same time, the execution time of the nodes used is acquired from the architecture information archive with no extra error.

**TABLE I**

RESULTS OF PARTITIONING WITH IPC TIME ERROR
FOR GAUSSIAN ELIMINATION GRAPH

<table>
<thead>
<tr>
<th>IPC time(sec)</th>
<th>0</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.16</th>
<th>0.18</th>
<th>0.25</th>
<th>0.35</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td># of partitions</td>
<td>199</td>
<td>199</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td># of trails</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Mean of exec(sec)</td>
<td>80.68</td>
<td>78.90</td>
<td>14.56</td>
<td>14.60</td>
<td>14.30</td>
<td>10.60</td>
<td>14.20</td>
<td>21.70</td>
<td>21.74</td>
</tr>
<tr>
<td>Max. of exec(sec)</td>
<td>97.80</td>
<td>96.94</td>
<td>17.48</td>
<td>17.54</td>
<td>16.16</td>
<td>12.12</td>
<td>16.80</td>
<td>25.30</td>
<td>26.02</td>
</tr>
<tr>
<td>Min. of exec(sec)</td>
<td>67.80</td>
<td>63.42</td>
<td>12.90</td>
<td>12.72</td>
<td>12.86</td>
<td>9.32</td>
<td>12.76</td>
<td>18.34</td>
<td>18.34</td>
</tr>
</tbody>
</table>

For each value of IPC time, Table I gives the number of partitions produced, the number of trials, the mean value of the measured execution time, the maximum measured value, and the minimum measured value of the execution time. In Table I, the intervals between successive values of IPC time are non-equidistance. This is because the results of the partitioning do not change for intervals of IPC time. To identify end points of these intervals, we first quantized the IPC time into equal size intervals and measured the results of the partitioning algorithm. If the results changed in an interval, we split this interval and so on, until the point causing a change in the partitioning was found. Note
that we may still have missed some points since it is impossible to split the time intervals infinitely.

Figure 13 shows the results of the GC algorithm partitioning the GE graph while the magnitude of the IPC time changes. As shown in Figure 13, if the IPC time is underestimated, the number of partitions increases and results in more communication. On the other hand, if the IPC time is overestimated, the number of partitions decreases and results in less communication. If the IPC time is zero, each node can reside in one partition because of the free communication cost. The GC algorithm exploits the constraints between maximizing parallel execution and reducing communication overhead such that when the IPC time is relatively short compared with the execution time of the partitions in the processor, many processors are used so that maximum parallelism can be achieved. When the IPC time is relatively long compared with the execution time of the partitions in the processor, fewer processors will be used so that the communication overhead can be minimized.

Figure 14 shows the execution time of the GE graph with different value of IPC time after the partitions are allocated in the system. The test run includes 5 trials for each value of the IPC time for a total of 45 trials. The mean value is given by averaging the results of 5 trials and the error bar shows the maximum and minimum values of the results. The figure shows that some run-time results (mean value of execution time) are different even if the results of the partitioning (number of partitions) are the same. Note that, in our experiments, the interconnection topologies of the partitions are the same while the numbers of partitions are the same. We also observed that there are many overlaps between the error bars indicating that the actual performance may be hidden by the inaccuracy. The error at run-time caused by the varying system behavior hurts the evaluation of the mapping quality. The use of the mean value compensates for some measurement error but may not be a good solution especially when the quality of
Figure 13. Result of GC Algorithm for GE Graph with IPC Time Error.

Figure 14. Execution time for GE Graph with IPC Time Error.
mapping is very sensitive to the errors.

In the ideal situation, it is easy to understand that the best performance (minimum execution time) should be seen while the IPC time has no extra error and the performance will get worse while the error of the IPC time increases. However, the curve did not show exactly as we expected. The minimum occurred when the IPC time used was slightly above the mean value. The reason for this could be: First, the use of the heuristic GC algorithm cannot find the optimal solution even if the IPC time is accurate. Second, the execution time of the partition is possibly underestimated along with the underestimation of the node execution time by measurement which, in turn, causes the overestimation of the IPC time. Third, the impact of allocating causes the inaccurate measurement. Finally, the system behavior contributes to the error.

TABLE II
RESULTS OF PARTITIONING WITH NODE EXECUTION TIME ERROR
FOR GAUSSIAN ELIMINATION GRAPH

<table>
<thead>
<tr>
<th>Node exec error(%)</th>
<th>-100</th>
<th>-75</th>
<th>-50</th>
<th>-20</th>
<th>0</th>
<th>20</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td># of partitions</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td># of trials</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Mean of exec(sec)</td>
<td>55.02</td>
<td>54.68</td>
<td>25.24</td>
<td>10.70</td>
<td>15.10</td>
<td>15.34</td>
<td>17.90</td>
<td>18.20</td>
<td>20.00</td>
</tr>
<tr>
<td>Max. of exec(sec)</td>
<td>63.00</td>
<td>61.20</td>
<td>30.06</td>
<td>12.84</td>
<td>16.02</td>
<td>18.72</td>
<td>20.58</td>
<td>21.28</td>
<td>22.00</td>
</tr>
<tr>
<td>Min. of exec(sec)</td>
<td>45.09</td>
<td>44.80</td>
<td>20.83</td>
<td>8.94</td>
<td>12.82</td>
<td>13.00</td>
<td>15.08</td>
<td>15.68</td>
<td>18.30</td>
</tr>
</tbody>
</table>

Another experiment indicates the sensitivity of the quality of the GC partitioning algorithm to the inaccurate node execution time while the IPC time is fixed. The same GE program is mapped on the 4-processor system.
Table II gives the results of the mapping while the execution time of each individual node has error ranging from -100% to 100%. Note that the GE graph has different nodes with different node execution time. Thus, we use error percentage instead of actual value of node execution time to indicate the error we added. The execution time of the partition is estimated based on the execution time of the nodes in this partition. The tiny inaccuracy in the execution time of the node may cause significant error in the execution time of the partition if the partition size is large. When the execution time of the partition is underestimated, the algorithm tends to merge the partitions because the communication cost is relatively overestimated. On the other hand, when the execution time of the partition is overestimated, it results in more partitions because the communication cost is underestimated correspondingly.

Figure 15 shows the number of partitions while the execution time of the partition is either underestimated or overestimated. To test the execution time of the program, 5 trials are conducted for each value of node execution time. A total of 45 trials were conducted. The execution time shown on Figure 16 is the mean of the measured results from 5 trials. The error bar shows the measurement error at run-time.

The matrix multiplication application is the graph with loop structures. The loop structure of the graph hides the concurrency of the program execution but presents coarse-grained parallelism. Table III shows the results of the partitioning quality with different errors in the IPC time for the matrix multiplication program. A 15x15 matrix multiplication graph with 181 nodes was tested on a 4-processor system. The graph has 15 loops and each loop calculates one column result of the matrix product.

Figure 17 illustrates the number of partitions with different IPC time. Figure 18 illustrates the impact of the inaccurate IPC time on the execution time of the program. The testing approach is the same as we used in the GE graph experiment.
Figure 15. Result of GC Algorithm for GE Graph with Node Execution Time Error.

Figure 16. Execution time for GE Graph with Node Execution Time Error.
TABLE III

RESULTS OF PARTITIONING WITH IPC TIME ERROR FOR MATRIX MULTIPLICATION GRAPH

<table>
<thead>
<tr>
<th>IPC time (sec)</th>
<th>0</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.138</th>
<th>0.20</th>
<th>0.35</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td># of partitions</td>
<td>181</td>
<td>151</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td># of trials</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Mean of exec (sec)</td>
<td>2212.0</td>
<td>1770.0</td>
<td>252.5</td>
<td>268.7</td>
<td>255.3</td>
<td>256.7</td>
<td>263.5</td>
<td>258.0</td>
</tr>
<tr>
<td>Max. of exec (sec)</td>
<td>2434.0</td>
<td>1902.0</td>
<td>283.3</td>
<td>300.0</td>
<td>296.9</td>
<td>273.3</td>
<td>297.0</td>
<td>282.0</td>
</tr>
<tr>
<td>Min. of exec (sec)</td>
<td>1998.0</td>
<td>1693.0</td>
<td>240.7</td>
<td>230.0</td>
<td>243.7</td>
<td>210.0</td>
<td>210.0</td>
<td>226.9</td>
</tr>
</tbody>
</table>

Three notable features can be seen from these results: First, the result of the GC algorithm is insensitive to the IPC time when the IPC time is not close to zero. This is because the execution time of each loop is much longer than the communication time between the loop branches. The result of the partitioning keeps each loop branch in a separate module rather than merging them. Second, for each loop branch, if the loop iteration number is large (In 15x15 matrix multiplication graph, the loop iteration number is 225.), it puts all the nodes inside the loop into one module rather than separating them because of the expensive communication cost. Mapping using the loop structure takes each loop as an entity instead of an individual node. From the above it can be seen that the quality of partitioning is relatively insensitive to the inaccuracies for the loop graph. Last, when the IPC time is less than 0.01 second but greater than zero, the result of the partitioning puts some nodes inside the loop into separate modules and, when IPC time is zero, puts all the nodes into separate modules. In this case, allocating may dominate the performance of the GC algorithm because the number of partitions is much larger than the number of processors. The "modulo" allocating algorithm merges partitions to remove the mismatch between the ideal architecture used in partitioning and the actual
Figure 17. Result of GC Algorithm for Matrix Multiplication Graph.

Figure 18. Execution time for Matrix Multiplication Graph.
architecture. It is important noting that, with an increase in number of processors, the total execution time may or may not be improved because more interprocessor communication may be needed.

The tree graph gives another example to show the effect of the inaccuracies of the lower-level meters on the mapping quality with fine-grain representation. A 256-node tree graph calculating the vector product of two 256 element vectors was mapped to a 4-processor system. We measured the execution time of the tree graph while the value of the IPC time was varied. The results are presented on Table IV.

**TABLE IV**

RESULTS OF PARTITIONING WITH IPC TIME ERROR FOR VECTOR INNER PRODUCT GRAPH

<table>
<thead>
<tr>
<th>IPC time (sec)</th>
<th>0</th>
<th>0.005</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.138</th>
<th>0.16</th>
<th>0.25</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td># of partitions</td>
<td>256</td>
<td>255</td>
<td>255</td>
<td>46</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td># of trials</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Mean of exec (sec)</td>
<td>12.14</td>
<td>9.67</td>
<td>10.30</td>
<td>2.26</td>
<td>2.56</td>
<td>2.37</td>
<td>1.99</td>
<td>3.77</td>
<td></td>
</tr>
<tr>
<td>Max. of exec (sec)</td>
<td>14.40</td>
<td>11.90</td>
<td>11.00</td>
<td>3.99</td>
<td>2.50</td>
<td>2.67</td>
<td>2.67</td>
<td>2.30</td>
<td>4.41</td>
</tr>
<tr>
<td>Min. of exec (sec)</td>
<td>10.99</td>
<td>9.01</td>
<td>8.26</td>
<td>2.55</td>
<td>2.10</td>
<td>2.33</td>
<td>2.01</td>
<td>1.87</td>
<td>3.02</td>
</tr>
</tbody>
</table>

Figure 19 gives the number of partitions with different IPC time error. Figure 20 shows the execution time of the tree graph with different values of IPC time. It is clear from observing Figure 19 and 20 that the GC algorithm produces more partitions with underestimated IPC time and results in fewer partitions when the IPC time is overestimated. The performance of the mapping gets worse while the IPC time is either underestimated or overestimated. However, as shown in Figure 20, the minimal execution time is not at the point that the IPC time has no extra error. The best performance is
Figure 19. Result of GC Algorithm for Vector Inner Product Graph

Figure 20. Execution time for Vector Inner Product Graph
obtained while the IPC time is about 0.25 second. This is possibly caused by the same reasons as described in GE graph testing. At this point, the allocating effect is less than that while the IPC time is relatively accurate because of the smaller number of partitions. Due to the limits of our system, unfortunately, we could not precisely characterize the interactive effects among the use of suboptimal technique, the allocating, and the inaccurate lower-level meters to the mapping quality.

IV.4 GENERAL COMMENTS ON SENSITIVITY TESTING

This chapter has investigated the source of error and the impact of the error on the quality of mapping. We characterized the measurement error caused by the varying system behavior. We then described our approach for measuring the lower-level meters. We found that the measurement error affects the accuracy of the lower-level meters directly. Finally, we tested the quality of the mapping with different programs while error in the lower-level meters was introduced. We observed the results of the partitioning first by looking at the change in the number of partitions when the error in the IPC time and node execution time varies. The results showed that accurate IPC time and node execution time are important to the mapping search. Then, we allocated the partitions to the processors and observed the variation of the execution time of the program with different errors. The interesting results conclude that inaccuracy in the various parameters lead to non-optimality, a point which has been neglected by most researchers.

The fine-grained, coarse-grained, and pipelined feature programs reflect different program characteristics. The use of various programs in the testing indicates that the mapping algorithm may choose different models for different programs to reduce the execution time of the mapping process. For example, rough estimates can be used in the model for matrix multiplication application program because it is relatively insensitive to the inaccuracies in IPC time.
Due to the time constraint, I was not able to test the effects of the error in node execution time to the quality of the GC algorithm for matrix multiplication and vector inner product graph.

To improve the quality of the mapping, further research should investigate the source of the inaccuracies and improve the model rather than refining the algorithm. One approach to improve the accuracy of the model is that we can use a probabilistic model instead of specifying a single value for the lower-level meters we are using. We may also conduct our experiments on a system with more processors to minimize the allocating effect to the mapping performance. Of course, the effect of the system behavior to the measurement of mapping quality can not be eliminated easily. Any attempt to correct this drawback is doomed to failure.
CHAPTER V

CONCLUSION AND FUTURE WORK

To deal with the complexity of the mapping problem while still exploiting fine-grained parallelism, we perform mapping in several substeps: partitioning, allocating, and scheduling, with each step using only a few details of the program and the architecture. Thus, the effect of incomplete and inaccurate information on the performance of the overall mapping must be investigated. We used the traditional approach to develop and evaluate a mapping procedure and remedied its drawback by evaluating the sensitivity of the mapping quality to inaccuracies in various parameters.

We noticed that non-optimality comes from three major sources: the use of heuristic, suboptimal search strategies, the inaccurate lower-level meters used to characterize the mapping problem, and the inaccurate models used by the search strategies. We found that this problem is significant because even optimal mapping methods using inaccurate parameters may lead to non-optimality.

To validate our approach, the Global Cluster partitioning algorithm was used to minimize the intermodule communication, exploit data dependence constraints, and balance the load. The use of a heuristic approach reduces the complexity of the mapping procedure. Two heuristics are used in the algorithm, with each heuristic being guided by an analytical model. These models use the descriptions of the program and architecture characteristics, lower-level meters, to influence the mapping search.

To implement our experiments, a multiprocessor interpreter, Parplum, was used. The experimental environment allows experimenters to interchange each submapping
method, select different models for different search methods, and add desired error on the models and the lower-level meters they use. To explore fine-grained parallelism in the mapping process, we develop some programs showing different features of parallel programming paradigms.

Through extensive measurement, we found the results of the IPC time and the execution time of the program measurement at different time are different because of the process load and network server load. Our hypothesis was that the quality of partitioning is greatly influenced by these inaccuracies. Our experiments presented the sensitivity of the partitioning quality to the inaccuracies in a few parameters: inaccurate execution time of the functional nodes and the IPC time. These two lower-level meters characterize the key constraints between balancing the load and minimizing communication overhead. The experimental result proved our hypothesis that the inaccuracies during the mapping process are the first-order effects on the performance of mapping and may lead to non-optimal solution.

We are at the initial stage of sensitivity exploration. Our approach helps to identify the characteristics of the mapping in the real world and verify the existence of the problem. Our primary goal has been achieved. However, we now see several shortcomings to remedy in the future:

(1) To reexamine our notational framework for sensitivity analysis in light of our experimental results. This will result in a refined understanding of the problem, which will help guide the design of further experiments.

(2) To identify the interaction between the effect of allocating and the effect of meter inaccuracies on the quality of the partitioning.

(3) To improve the accuracy of the models because of the difficulty in characterizing the data dependence constraints.
(4) To reduce the measurement error caused by the variable system behavior. The current approach using the average value of the measured results at run-time conceals the actual performance of the mapping.

(5) To deal with large size programs due to the current limitation of the Parplum system.

Furthermore, we have not developed an optimal search strategy using relatively accurate models. Thus, we can not examine the quality of mapping or submapping using a heuristic technique by comparing with the optimal solution. Based on the notation defined in Chapter II, there are several large tasks for future study, including:

(1) Identifying more parameters that may affect the quality of mapping and concentrating on the key factors which may impact the performance. Thus, we may choose the approach to either improve the algorithms or correct the inaccuracies in the models or the lower-level meters.

(2) Developing more accurate models and reducing the error in the existing models for different mapping or submapping search strategies and examining the sensitivity of the mapping quality to these models. For instance, we may try to use probabilistic models for the cost information instead of the model we are using. We may also use measured IPC time and execution time of the nodes as the direct inputs of the model which helps in making correct decision for mapping search.

(3) Investigating the impact of inaccuracies on the quality of some other submapping search strategies. It includes two possibilities: First, develop more partitioning algorithms using the same models we applied so that we can compare these algorithms and choose the one which has the least sensitivity to the inaccuracies and still results in a good solution. We can also use different models for different submapping algorithms. Thus, we may find an algorithm associated with one or more models which produces good results for a particular problem. Second,
develop some allocating or scheduling algorithms using different models or lower-level meters and find the impact of the inaccuracies in the models and the meters on the quality of these submappings.

Ignoring the details of programs and architectures during mapping procedure while still achieve good mapping is a difficult task. Our work provides new insight and encourages researchers to make further investigation.
REFERENCES


