Synthesis of Application-Specific Multiprocessor Architectures

BY

Shiv Prakash and Alice C. Parker

CEng Technical Report 90-25

Electrical Engineering Systems
University of Southern California
Los Angeles, CA. 90089-0781
Synthesis of Application-Specific Multiprocessor Architectures

Shiv Prakash and Alice C. Parker
Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-0781

November 5, 1990
Synthesis of Application-Specific Multiprocessor Architectures

Abstract

This paper describes a formal technique for automated synthesis of multiprocessor systems for given applications. The application task is specified in terms of a graph, and the architecture synthesized includes a set of processing elements and the interconnection architecture between them. The technique generates a task execution schedule along with the architecture. The technique involves creation of a Mixed Integer-Linear Programming (MILP) model and solution of the model. A primary component of the model is the set of relations that must be satisfied to ensure proper ordering of various events in the task execution as well as to ensure completeness and correctness of the system. Some example architectures have been synthesized, and these results are reported.

---

This work was supported in part by the Department of Air Force, the Department of Army and the Department of Navy, Contract No. N00039-87-C-0194 and in part by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under contract No. JFBI90092. The views and conclusions considered in this document are those of authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Defense Advanced Research Projects Agency or the U.S. Government.
1 Introduction

Application-specific systems have become so complex that system-level design decisions cannot be made without the aid of computer tools. As system complexity and size have increased, designers have relied increasingly on analysis techniques like simulation and queueing models for assistance during the design process. However, in most cases, system design decisions have been left to the human designer, who often uses a "generate and test" approach to confirm the validity of his or her decisions. There is a growing need to develop tools for system-level design. This paper addresses a technique for design of multiprocessor systems for given applications.

Our focus is on the design of the system architecture, which is the first step in the design of an application-specific multiprocessor system. We assume the application domain is specified in terms of a task data flow graph. The task data flow graph specifies a set of subtasks (nodes in the graph) that need to be performed and the data precedence between them (arcs in the graph). Given the task data flow graph, the goal is to synthesize a multiprocessor architecture which meets various cost and performance requirements and constraints. Synthesizing an architecture involves making decisions about the number and types of processing elements selected, the overall interconnection between the processing elements, and the scheduling of subtasks on the processing elements.

This paper describes a new approach which is aimed at producing a custom multiprocessor architecture, as well as mapping the subtasks onto the architecture and providing a schedule for the task execution. This research focuses on the automatic design of the multiprocessor architecture itself, not merely the mapping of tasks onto a given architecture. A distinguishing feature of the research is the fact that we are designing a truly heterogenous system, in terms of the functionality and the cost-speed characteristics of the processing elements, which allows a more precise tailoring of the synthesized architecture to a specific application. Also, our approach can be used to explore different interconnection styles; e.g., bus, point-to-point, ring, or a mixture of these. We assume there is no global clock and communications between subtasks are asynchronous. With the exception of some early work by Talukdar and Mehrotra [9], we believe this is the first publication to describe a solution to this problem.
2 Previous Related Research

Our research builds on past research done by others in the areas of multiprocessor task allocation and data path synthesis. Most of the previous related work on multiprocessors is directed at the problem of task allocation for a given architecture.

The assignment of tasks to a fixed multiprocessor system was inspired by Stone's work on the two-processor problem [10]. More recently, Chu et. al. [2] have described an integer 0-1 programming approach to the problem of task allocation in distributed data processing. The problem they have considered involves the allocation of a set of m sub-tasks to a set of p (fixed) processors already interconnected in some fashion. Their model does not consider the data precedence relations among the subtasks. Houstis [6] describes task allocation for real-time applications with concurrent selection of the optimal number of processing elements. She does consider data precedence, but assumes identical processing elements and does not consider timing constraints. Indurkhya et.al.[8] use random models for distributed programs, and confirm intuitive results for the 2-processor and n-processor cases. Haddad[3] described a load allocation problem solved with continuous partition sizes to minimize total execution time. An early study mapping algorithms such as the Fast Fourier Transform onto the CM* multiprocessor in order to meet real time constraints was undertaken by Brantley [1]. Talukdar and Mehrotra [9] describe a problem which is a simplified and similar version of our problem. They also model the problem using mathematical programming, though they use heuristics to solve it.

Mathematical programming has also been applied to the data path synthesis problem. Hafer and Parker [4] used a mixed-integer linear programming approach to automatically synthesize register-transfer level datapaths, given a data flow/control flow graph description of the hardware. The approach involves developing various timing relationships to be satisfied, but does not include interconnection styles or delays, and does not consider the detailed timing of multiple outputs. Hwang et. al. [7] have described an integer linear programming model for the scheduling problem in data path synthesis under resource constraints, and they present a heuristic technique called Zone Scheduling for solving large size problems.
3 The Problem Definition

We are addressing the problem of multiprocessor architecture synthesis for a given application task. Our approach involves creation of a formal model of the multiprocessor synthesis problem and the solution of this model, using mathematical programming.

The task consists of a set of subtasks. Each subtask requires certain input data and produces certain output data. Inputs to a subtask may come from other subtasks and outputs from a subtask may go to other subtasks. The set of subtasks and the input-output relationships among them can be expressed by a task data flow graph as shown in Figure 1. The subtask nodes are labeled $S_1$, $S_2$, etc. ($S_a$ in general). The input end of a data arc is labeled $i_{a,b}$ if it provides $b^{th}$ input to subtask $S_a$, and the output end is labeled $o_{a,c}$ if it transmits the $c^{th}$ output from the subtask $S_a$. Although we represent our task by a data flow graph, there is a subtle distinction between our meaning and the traditional meaning attached to a data flow graph. With the traditional meaning, a subtask would require all the inputs before starting its execution and none of the outputs would be available until after its execution is over. However, in our model subtasks do not require all the inputs before starting their execution and they may produce some outputs even before their completion. To express this possibility, each input $i_{a,b}$ has a parameter $f_R(i_{a,b})$ associated with it which specifies that up to $f_R(i_{a,b})$ fraction of the subtask $S_a$ can proceed without requiring the input $i_{a,b}$. Similarly, each output $o_{a,c}$ has a parameter $f_A(o_{a,c})$ associated with it which specifies that the output $o_{a,c}$ becomes available when $f_A(o_{a,c})$ fraction of the subtask $S_a$ is completed.

The multiprocessor architecture is specified in terms of the processors selected and the interconnection architecture between them. A simple example multiprocessor system is shown in Figure 2. The specific model described in this paper assumes point-to-point interconnection; i.e., if a processor $p_{d1}$ needs to send data to another processor $p_{d2}$, then there must be a direct communication link from $p_{d1}$ to $p_{d2}$. However, it must be emphasized that the approach is capable of handling other styles of interconnection.

For each subtask $S_a$, a set $P_a$ represents the set of processors capable of executing it. However, only one processor actually performs the subtask in the synthesized architecture, and the execution time for the subtask depends on the processor on which it is performed. A parameter, denoted as $D_{pS}(p_d,S_a)$, specifies the execution time for the subtask $S_a$ if the processor $p_d$ is selected to perform it.

A data arc from node $S_{a1}$ to node $S_{a2}$ implies that some data is transferred from
the subtask $S_{a1}$ to the subtask $S_{a2}$. The volume of data transferred varies from arc to arc, and a parameter $V_{a1,a2}$ specifying the volume is associated with each arc. The data transfer maybe a remote transfer (if the two subtasks are mapped to different processors in the synthesized system), where the data is transferred from a processor to another; or it may be a local transfer within the same processor (if the two subtasks are mapped to the same processor). Delay associated with a data transfer depends on whether it is a remote transfer or a local transfer. Local transfer delay could be negligible compared to the remote transfer delay. In any case, the local transfer delay is represented by the parameter $D_{CL}$ which specifies the time taken in transferring a unit volume of data locally. The remote transfer delay is represented by the parameter $D_{CR}$ which specifies the time taken in transferring a unit volume of data remotely.

A set $P$ represents the set of all the processors available for selection as part of the synthesized architecture, where

$$P = \bigcup_{a} P_{a}$$

Associated with each processor $p_{a} \in P$ is a parameter $C_{a}$ which specifies the cost of the processor. Another cost parameter is $C_{L}$ which specifies the cost of creating a communication link between two processors.

Certain constraints related to the cost of the system as well as timing of arbitrary events may also be specified. In summary, the following are the problem inputs and outputs:

- **Problem inputs:**
  - A task data flow graph specifying the overall application task, along with the associated parameters
  - A set of processing elements with varying functionality, cost and performance
  - Communication link characteristics: cost and performance
  - Constraints on total system cost
  - Constraints on timing of arbitrary events

- **Problem outputs:**
  - A multiprocessor architecture, including
    - the chosen set of processing elements
    - the interconnection style for the elements
  - A schedule for the subtasks
  - Detailed timing information for computation and transfer of data
4 The Problem Approach: Formal Model

Our approach is a natural outgrowth of the work described by Chu [2], Talukdar [9], and Hafer [4]. We are using mathematical programming to produce a formal model for the problem. Such a mathematical model allows us a deep understanding of the problem and allows us to verify our software more easily, even if future run-time problems with larger examples force us to resort to heuristics. Such an approach allows us to modify, extend and enhance the model to include more design possibilities and variations without significant reconstruction of existing code. Also, the approach offers a great degree of flexibility in handling arbitrary constraints.

4.1 The Model

A complete mathematical programming formulation of the problem requires specification of an objective function that has to be optimized and a set of constraints that have to be satisfied. The objective function can be whatever the designer wishes; e.g., the total system cost, or the overall system performance. The set of constraints consists of the constraints that must be satisfied for the overall task to be performed correctly as well as the arbitrary timing and cost constraints imposed by the designer.

4.1.1 The Constraints

The constraints that must be satisfied for the overall task to be performed correctly consist primarily of the relations that ensure proper ordering of the subtasks and the data transfers taking into account the timing involved in carrying them out and the relations that express the conditions for complete and correct system configuration. In order to express the various constraints, one needs to define certain variables related to the system. The necessary variables fall into two basic categories:

- **Timing variables**: These are real variables which represent timings of various critical events in the operation of the system. There are three classes of timing variables defined:
- Data availability timing variables:
  * Input data availability, $T_{IA}(i_{a,b})$: Time when the data required by input $i_{a,b}$ of subtask $S_a$ is available for use.
  * Output data availability, $T_{OA}(o_{a,c})$: Time when the output data value $o_{a,c}$ computed by subtask $S_a$ has become available.

- Subtask execution timing variables:
  * Subtask execution start, $T_{SS}(S_a)$: Time when the execution of subtask $S_a$ actually begins.
  * Subtask execution end, $T_{SE}(S_a)$: Time when the execution of subtask $S_a$ is completed.

- Data transfer timing variables:
  * Data transfer start, $T_{CS}(i_{a,b})$: Time when the communication/transfer of the data required by input $i_{a,b}$ of subtask $S_a$ actually begins.
  * Data transfer end, $T_{CE}(i_{a,b})$: Time when the communication/transfer of the data required by input $i_{a,b}$ of subtask $S_a$ ends.

- Binary variables: These are 0-1 variables which represent the implementation decisions regarding the system configuration. There are two types of binary variables defined:
  - Subtask-to-processor-mapping variable, $\sigma_{d,a}$: The variables of this type specify the mapping between the subtasks and the processors. $\sigma_{d,a} = 1$ indicates processor $p_d$ will implement subtask $S_a$.
  - Data-transfer-type variable, $\gamma_{a1,a2}$: The variables of this type specify the data transfer type for the various data arcs. $\gamma_{a1,a2} = 1(0)$ indicates that data transfer from subtask $S_{a1}$ to subtask $S_{a2}$ is a remote (local) transfer.

The necessary constraints have been classified into ten categories:

- Processor-selection constraint: For each subtask $S_a$, a set of processors $P_a$ is available to implement it. In order for the implementation to be correct, one and only one processor should be selected to implement the subtask. Thus, for each subtask $S_a$, the following must be satisfied:

  $$\sum_{d \in P_a} \sigma_{d,a} = 1 \quad (4.1.1)$$
- **Data-transfer-type constraint:** $\gamma_{a1,a2}$ is a variable which indicates whether the data transfer from the subtask $S_{a1}$ to the subtask $S_{a2}$ is a local transfer or a remote transfer. Now, if the subtasks $S_{a1}$ and $S_{a2}$ are mapped to the same processor (say $p_d$, where $p_d \in P_{a1}$ and $p_d \in P_{a2}$), then we know that it is a local transfer, and thus $\gamma_{a1,a2} = 0$. However, if they are mapped to different processors, then the data transfer is remote, and thus $\gamma_{a1,a2} = 1$. Thus, the defining equation for $\gamma_{a1,a2}$ is:

$$\gamma_{a1,a2} = 1 - \sum_{d | p_d \in P_{a1} \land p_d \in P_{a2}} \sigma_{d,a1}\sigma_{d,a2} \quad (4.1.2)$$

We will have such an equation for each pair of subtasks communicating with each other.

- **Input-availability constraint:** $T_{IA}(i_{a,b})$ is the time the data required at input $i_{a,b}$ will be available, which will be the time $T_{CE}(i_{a,b})$ when the data transfer has ended. So, for each input $i_{a,b}$, we have:

$$T_{IA}(i_{a,b}) = T_{CE}(i_{a,b}) \quad (4.1.3)$$

- **Output-availability constraint:** Once execution of the subtask $S_a$ begins, a certain time elapses before an output data value $o_{a,c}$ produced by the subtask becomes available. The time elapsed would be the time taken in executing $f_A(o_{a,c})$ fraction of the subtask; and so the time $T_{OA}(o_{a,c})$ must satisfy the following relation:

$$T_{OA}(o_{a,c}) = T_{SS}(S_a) + f_A(o_{a,c})(T_{SE}(S_a) - T_{SS}(S_a)) \quad (4.1.4)$$

We will have such a relation for each output.

- **Subtask-execution-start constraint:** $T_{SS}(S_a)$ is the time the subtask $S_a$ begins execution. There must be a certain relationship between the time a given subtask begins its execution and the times at which its various inputs become available. Since $f_A(i_{a,b})$ fraction of the subtask $S_a$ can proceed without requiring the input $i_{a,b}$, the following relation must be satisfied for all the inputs $i_{a,b}$ to the subtask:

$$T_{IA}(i_{a,b}) \leq T_{SS}(S_a) + f_A(i_{a,b})(T_{SE}(S_a) - T_{SS}(S_a)) \quad (4.1.5)$$

- **Subtask-execution-end constraint:** Once execution of a subtask begins, a time equal to the execution time of the subtask must elapse before the subtask is completed. Execution time of the subtask depends on the processor being used for it. A priori we do not know which processor a given subtask $S_a$ is going to be mapped to. Any processor from the set $P_a$ could be selected to execute the subtask $S_a$. The
uncertainty can be expressed by the following relation. The summation acts as a selection since only one \( \sigma_{d,a} = 1 \) for each \( a \):

\[
T_{SE}(S_a) = T_{SS}(S_a) + \sum_{d \in P_a} \sigma_{d,a} D_{PS}(p_d, S_a)
\]  

(4.1.6)

For each subtask \( S_a \), we need such a relation.

- **Data-transfer-start constraint**: The time at which transfer of data begins must be after the output data is produced. For each input data (except for external inputs) \( i_{a2,b2} \) (to the subtask \( S_a \)) being supplied by another subtask’s output, if the output supplying the data is \( o_{a1,c1} \), the following relation must be satisfied by \( T_{CS}(i_{a2,b2}) \):

\[
T_{CS}(i_{a2,b2}) \geq T_{OA}(o_{a1,c1})
\]  

(4.1.7)

- **Data-transfer-end constraint**: The time at which transfer of data ends depends on whether the transfer is remote or local. A priori we do not know which case will occur. However, the two possibilities can be combined into one single relation using the variable \( \gamma_{a1,a2} \). Thus, for each input data \( i_{a2,b2} \) being supplied by another subtask \( S_{a1} \), we have:

\[
T_{CE}(i_{a2,b2}) = T_{CS}(i_{a2,b2}) + \gamma_{a1,a2} D_{CR} V_{a1,a2} + (1 - \gamma_{a1,a2}) D_{CL} V_{a1,a2}
\]  

(4.1.8)

The next two categories of constraints ensure that the hardware resources (processors, communication links) are shared correctly. These constraints ensure that the same hardware resource is not scheduled to perform more than one function during any given time interval. In order to express these constraints concisely, we need to define a special function called an overlap function \( L \) (as defined in [4]). The function is defined on two closed intervals of time, \([t1, t2]\) and \([t3, t4]\) (where \( t1 < t2 \) and \( t3 < t4 \)), as:

\[
L([t1, t2], [t3, t4]) = \begin{cases} 
1, & \text{if the intervals overlap} \\
0, & \text{otherwise}
\end{cases}
\]

- **Processor-usage-exclusion constraint**: If two subtasks \( S_{a1} \) and \( S_{a2} \) are being executed by the same processor \( p_d \), then the two subtasks must not be scheduled to be executed at the same time. The situation that two subtasks \( S_{a1} \) and \( S_{a2} \) are being implemented by the same processor \( p_d \) implies \( \sigma_{d,a1} = \sigma_{d,a2} = 1 \). For each processor \( p_d \) and each pair of subtasks \( S_{a1} \) and \( S_{a2} \) such that the sets of processors \( P_{a1} \) and \( P_{a2} \) available to implement the subtasks contain the processor \( p_d \), the following relation ensures that the overlap in the usage of the processor by the two subtasks is prevented:

\[
\sigma_{d,a1} \sigma_{d,a2} L([T_{SS}(S_{a1}), T_{SE}(S_{a1})], [T_{SS}(S_{a2}), T_{SE}(S_{a2})]) = 0
\]  

(4.1.9)
• **Communication-link-usage-exclusion constraint:** If the data required by two inputs \( i_{a1,b1} \) and \( i_{a2,b2} \) are being transmitted over the same communication link, then the two data transfers must not be scheduled at the same time. Let us say the input data \( i_{a1,b1} \) is supplied by the subtask \( S_{a3} \) and the input data \( i_{a2,b2} \) is supplied by the subtask \( S_{a4} \). The two inputs \( i_{a1,b1} \) and \( i_{a2,b2} \) will be transmitted over the same communication link if the two subtasks \( S_{a1} \) and \( S_{a2} \) are mapped to the same processor, say \( p_{d2} \), and also the subtasks \( S_{a3} \) and \( S_{a4} \) are mapped to the same processor, say \( p_{d1} \) (in that case, both the inputs will be transmitted over the communication link from processor \( p_{d1} \) to processor \( p_{d2} \)). For each processor pair \( (p_{d1}, p_{d2}) \) and each pair of inputs \( i_{a1,b1} \) and \( i_{a2,b2} \) (to subtasks \( S_{a1} \) and \( S_{a2} \) respectively, and from subtasks \( S_{a3} \) and \( S_{a4} \) respectively) such that the sets of processors \( P_{a1} \) and \( P_{a2} \) available to implement the subtasks \( S_{a1} \) and \( S_{a2} \) contain the processor \( p_{d2} \) and the sets of processors \( P_{a3} \) and \( P_{a4} \) available to implement the subtasks \( S_{a3} \) and \( S_{a4} \) contain the processor \( p_{d1} \), the following relation ensures that the overlap in the usage of the communication link from processor \( p_{d1} \) to processor \( p_{d2} \) by the two data transfers is prevented:

\[
\sigma_{d2,a1} \sigma_{d2,a2} \sigma_{d1,a3} \sigma_{d1,a4} L([T_{CS}(i_{a1,b1}), T_{CE}(i_{a1,b1})], [T_{CS}(i_{a2,b2}), T_{CE}(i_{a2,b2})]) = 0 \tag{4.1.10}
\]

The set of constraints described here should be treated as an example set. The exact form of constraints used can be tailored to meet the characteristics of the design problem at hand. Our approach offers a great degree of flexibility in this regard.

4.1.2 **Objective Functions**

Two of the most important goals that the designer may wish to optimize are the overall system performance and the total system cost.

**Overall System Performance:** The performance is usually measured by how fast the system can perform the task. So, it can be represented by the time at which the task is completed (or all the subtasks are completed). If \( T_F \) is a real variable representing the time at which the task is completed, then the objective is to minimize \( T_F \).

To ensure that \( T_F \) represents the time at which all the subtasks are completed, we
need to introduce the following constraint in the model (for each subtask $S_a$):

\[ T_P \geq T_{SE}(S_a) \] (4.1.11)

**Total System Cost:** The total cost of the system can be expressed as the sum of the costs of the processors selected and the costs of the links created. In order to do so, we need to define two types of binary variables:

- **Processor-selection variable, $\beta_d$:** The variables of this type specify which processors have been selected in the synthesized architecture. $\beta_d = 1$ indicates the processor $p_d$ is being included in the system.

- **Communication-link-creation variable, $\chi_{d1,d2}$:** The variables of this type specify what communication links are present in the synthesized architecture. $\chi_{d1,d2} = 1$ indicates there exists a communication link from the processor $p_{d1}$ to the processor $p_{d2}$ in the designed system.

Using the variables defined above, the objective is to:

\[ \text{MINIMIZE} \sum_{d \in \mathcal{P}} \beta_d C_d + C_L \left( \sum_{d_1,d_2 \in \mathcal{P}: p_{d1}, p_{d2} \in \mathcal{P}} \chi_{d1,d2} \right) \]

where $C_d$ is the cost of a processor $p_d$ and $C_L$ is the cost of building a communication link between two processors, as defined in Section 3. The variables of type $\beta_d$ are related to the variables of type $\sigma_{d,a}$. A processor $p_d$ will be included in the system if and only if at least one of the subtasks $S_a$ ($p_d \in P_a$) is mapped to it, which implies that the variable $\beta_d$ is the logical OR of all the $\sigma_{d,a}$ variables. This can be expressed by introducing the following constraint in the model (for all $a$ such that $p_d \in P_a$):

\[ \beta_d \geq \sigma_{d,a} \] (4.1.12)

The variables of type $\chi_{d1,d2}$ are also related to the variables of type $\sigma_{d,a}$. A communication link is created from processor $p_{d1}$ to processor $p_{d2}$ if and only if at least one of the subtasks $S_{a1}$ ($p_{d1} \in P_{a1}$) mapped to the processor $p_{d1}$ needs to send data to at least one of the subtasks $S_{a2}$ ($p_{d2} \in P_{a2}$) mapped to the processor $p_{d2}$. So, the variable $\chi_{d1,d2}$ is the logical OR of all the product terms of the form $\left(\sigma_{d1,a1}\sigma_{d2,a2}\right)$, where the subtask $S_{a1}$ supplies some data to the subtask $S_{a2}$. This condition leads to the introduction of following constraint in the model (for all $a1,a2$ such that $p_{d1} \in P_{a1}$ and $p_{d2} \in P_{a2}$ and subtask $S_{a1}$ sends data to subtask $S_{a2}$):

\[ \chi_{d1,d2} \geq \sigma_{d1,a1}\sigma_{d2,a2} \] (4.1.13)
The essence of the model has been presented. It is easy to see that arbitrary constraints imposed by the designer can be expressed using the timing and binary variables defined in the model.

4.2 Synthesis Using the Model

Several constraints comprising the formulation presented in Section 4.1 are non-linear relations. These relations were linearized and the model was converted into a MILP (Mixed Integer-Linear Programming) formulation. Bozo [5], a branch-and-bound program to solve an MILP problem has been developed by L. J. Hafer of Simon Fraser Univ. Bozo invokes a commercial linear programming package, XLP, developed by XMP Software, Inc. We are using this program to solve our MILP model. Architectures for some small example tasks have been synthesized by solving the MILP problem. These results are discussed in Section 5.

5 Experiments and Results

We have considered two example task graphs. The first example consists of four subtask nodes, while the second consists of nine. Some of the data related to these examples is taken from [9].

5.1 Example 1: Four-Node Task Graph

This example data flow graph is shown in Figure 1. Associated $f_R$ and $f_A$ parameters are also given in the figure, constraining input/output timing for the subtasks.

We assume we have available three types of processors: $p_1$, $p_2$, $p_3$. The costs of these processors and the execution times of various subtasks on the processors are given in Table 1. An entry of '−' in the table implies that the particular processor is functionally not capable of performing the particular subtask. As is obvious from the table, different processors have different cost-speed-functionality characteristics.

In this example the volume of data that needs to be communicated is one unit at each
Figure 1: Four-Node Task Graph
Table 1: Execution Time and Cost Table for Four-Node Graph

<table>
<thead>
<tr>
<th>Proc.</th>
<th>Cost</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>$p_2$</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$p_3$</td>
<td>2</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Architectures for Four-Node Graph

<table>
<thead>
<tr>
<th>Design</th>
<th>Runtime</th>
<th>Cost</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>14</td>
<td>2.5</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

arc in the graph. Local transfer delay is given to be negligible; i.e., $D_{CL} = 0$. We are also given the communication link characteristics. The cost of a link, $C_L$, is one unit; and the remote transfer delay for a unit volume of data over a link, $D_{CR}$, is also one unit.

The MILP model for the example consists of 93 variables, 21 timing and 72 binary, and 174 constraints. Bozo was used to generate 4 non-inferior architectures. These different architectures were generated by changing the constraint value for the total cost of the system, and optimizing the overall performance of the system. Bozo’s runtime to generate each of these designs is of the order of a few seconds. These runtimes are on a system with CPU type Solbourne Series5e/900 (similar to Sun SPARCsystem 4/490) with 128 MB of memory. Cost, performance and runtime (in seconds) for the four designs are given in Table 2.

A brief discussion of these designs follows:

• *Design 1*: This design consists of 3 processors: $p_{1a}$ - a processor of type $p_1$, $p_{2a}$ - a processor of type $p_2$, and $p_{3a}$ - a processor of type $p_3$. Processor $p_{1a}$ performs
Figure 2: Synthesized Multiprocessor System I and Schedule for Four-Node Graph
subtask $S_1$, processor $p_{2a}$ performs subtasks $S_2$ and $S_4$ in that order, and processor $p_{3a}$ performs subtask $S_3$. There are three communication links: $l_{1a,2a}$, $l_{1a,3a}$, and $l_{2a,3a}$. Data $i_{4,1}$ gets transmitted on link $l_{1a,2a}$, data $i_{3,1}$ gets transmitted on link $l_{1a,3a}$, and data $i_{3,2}$ gets transmitted on link $l_{2a,3a}$. As an illustration, this architecture is shown in Figure 2. A detailed schedule for the various events is also shown in the figure.

- **Design 2**: This design is similar to design 1, and also consists of 3 processors: $p_{1a}$, $p_{2a}$, and $p_{3a}$. However, it has only two links: $l_{1a,2a}$, and $l_{1a,3a}$. Presence of fewer links forces a change in the mapping between the resources and the events. Processor $p_{1a}$ performs subtasks $S_1$ and $S_2$ in that order, processor $p_{2a}$ performs subtask $S_4$, and processor $p_{3a}$ performs subtask $S_3$. Data $i_{4,1}$ gets transmitted on link $l_{1a,2a}$, data $i_{3,1}$ and data $i_{3,2}$ get transmitted on link $l_{1a,3a}$ in that order.

- **Design 3**: This design consists of 2 processors: $p_{1a}$ - a processor of type $p_1$, and $p_{3a}$ - a processor of type $p_3$. Processor $p_{1a}$ performs subtasks $S_1$ and $S_4$ in that order, and processor $p_{3a}$ performs subtasks $S_2$ and $S_3$ in that order. There is a communication link: $l_{1a,3a}$. Data $i_{3,1}$ gets transmitted on link $l_{1a,3a}$.

- **Design 4**: This design consists of just 1 processor: $p_{2a}$ - a processor of type $p_2$. The processor performs the subtasks $S_2$, $S_1$, $S_3$, and $S_4$ in that order.

### 5.2 Example 2: Nine-Node Task Graph

The data flow graph is shown in Figure 3. For this example, we assumed that a subtask requires all the inputs before it can start and that none of the outputs from a subtask become available until its execution is over. Again, there are three types of processors, with the costs and the execution times given in Table 3. The volume of data is one unit for each arc. We are given: $D_{CL} = 0$, $D_{CR} = 1$. For this graph, we synthesized architectures for two different styles of interconnection.

#### 5.2.1 Point-to-Point Interconnection

Here, as before, if two processors need to communicate, then there must be a direct link between them; and the cost of building a link $C_L = 1$.

The MILP model consists of 272 variables, 47 timing and 225 binary, and 1081 constraints. We generated 5 non-inferior architectures by changing the constraint value for
Figure 3: Nine-Node Task Graph
## Execution Time

<table>
<thead>
<tr>
<th>Proc.</th>
<th>Cost</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_8$</th>
<th>$S_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$p_2$</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$p_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3: Execution Time and Cost Table for Nine-Node Graph

## Architectures for Nine-Node Graph (Point-to-Point)

<table>
<thead>
<tr>
<th>Design</th>
<th>Runtime</th>
<th>Cost</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62.2</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>445.17</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>538.67</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>75.18</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>6416.87</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 4: Architectures for Nine-Node Graph (Point-to-Point)

The total system cost, and optimizing the system performance. Bozo’s runtime for each of these designs is of the order of a few hours, except for design 5. Cost, performance and runtime (in minutes) for the five designs are given in Table 4. Discussion of the designs is omitted.

### 5.2.2 Bus-Style Interconnection

In this interconnection style, the system consists of a set of processors and a bus connecting all the processors to each other. So, the cost of the system is dominated by the costs of the processors selected. Our approach is capable of modeling such a system.

The MILP bus-architecture model consists of 200 variables, 47 timing and 153 binary, and 416 constraints. Three non-inferior architectures were generated by changing the constraint value for the total system cost, and optimizing the system performance. Runtime for each of these designs is of the order of a few hours. Table 5 gives the statistics
<table>
<thead>
<tr>
<th>Design</th>
<th>Runtime</th>
<th>Cost</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>107.3</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>89.53</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>61.52</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 5: Architectures for Nine-Node Graph (Bus-Style)

for the three designs (runtime in minutes). Discussion of the designs is omitted.

6 Conclusions and Future Work

In this paper, we have presented a formal model for the multiprocessor synthesis problem. It has been shown that the model can be solved fairly quickly for small size problems. Even for larger problems, the runtime is not prohibitive but there is much room for improvement. The model offers a great degree of flexibility. The model described in the paper should be treated as an example. The overall approach has potential to be applied to model more generalized design situations (e.g.; more design parameters, more design styles, etc.). Further research is required in two directions:

- Enhancing the model to handle a more generalized multiprocessor design problem.
- Designing techniques to improve the runtime for solution of the model.

Work is in progress at USC in both the directions. Memory design and mixed-style interconnect design are some of the aspects related to the multiprocessor synthesis problem that are being addressed. Incorporation of some heuristics for performing the branch-and-bound search seems to be a promising idea for improving the runtime.
7 Bibliography

References


