SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems*

Shiv Prakash and Alice C. Parker
Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-2562
shiv@eve.usc.edu (e-mail)
(213) 740-8284 (office)
(213) 740-4449 (fax)

December 4, 1991

*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.

SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems

Abstract

This paper describes a formal synthesis approach for design of optimal application-specific heterogeneous multiprocessor systems. The method generates a static task execution schedule along with the structure of the multiprocessor system and a mapping of subtasks to processors. The approach itself is quite general, but its application is demonstrated with a specific style of design. The approach 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. Several experiments and tradeoff studies have been performed using the approach. These results indicate that the approach can be a useful tool in designing application-specific multiprocessor systems.

1 Introduction

With VLSI systems becoming more and more commonplace, and with myriads of new applications for VLSI systems, there is a growing need for design of hardware systems for specific applications (i.e., hardware systems that will perform given tasks efficiently). This paper addresses a technique for design of *heterogeneous multiprocessor systems* for given applications. Example application tasks where heterogeneous multiprocessor systems are desirable can be found in many domains, including digital signal processing, robotics, and control of power systems.

We consider a heterogeneous multiprocessor system to be a system which makes use of several different types of processors, processing components, and/or connectivity paradigms to optimize performance and/or cost-effectiveness of the system. For example, different processor/processing component types could include vector processors, SIMD processors, MIMD processors, special purpose processors, and data-flow processors. Similarly, different connectivity paradigms could include bus, point-to-point, ring, or a mixture of these. The Purdue mixed-mode PASM system [35] is an example of a heterogeneous system.

1.1 Motivation for Synthesis

Some questions that come up during the design of application-specific multiprocessor systems are the following (the list is not exhaustive):

- how to consider all the relevant factors, costs, constraints, and objectives during the design,
- how to decide the number and types of processors to be included in the system,
- how to decide the interconnection between the selected processors,
- how to use the designed system effectively to perform the given application task, and
- how to map and schedule the subtasks onto the processors.

Obviously, the answers to these questions are difficult to obtain. It is important to devise systematic methods for designing such systems. As a matter of fact, answers to several design questions depend on the characteristics of the specific application task under consideration. Hence, it seems appropriate to consider such task characteristics while designing the system; i.e., the system should be specifically tuned to the application at hand. With

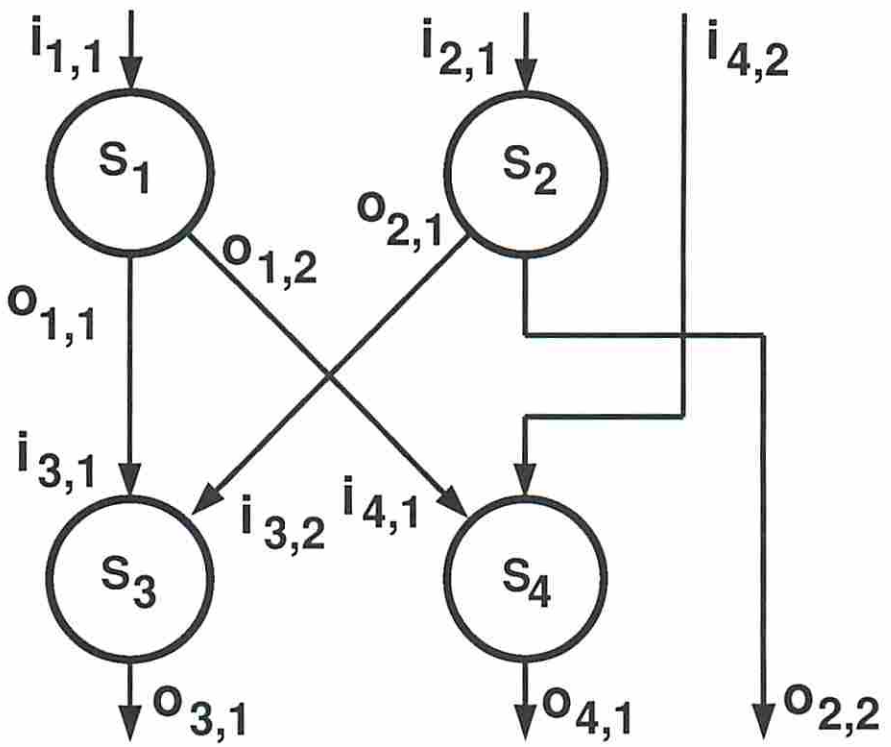
this in mind, we describe an "automatic synthesis approach" for design of application-specific multiprocessor systems. In such an approach, one starts with a given application and designs a complete system considering the application characteristics and all other relevant factors.

1.2 Overview of the SOS Approach

Some of the questions mentioned in Section 1.1 relate to the design of the system while others relate to how to effectively use the system. It is our view that in order to optimize overall performance and/or cost-effectiveness of the system, it is imperative to consider issues related to both the types of questions while designing the system. The SOS (Synthesis Of Systems) approach reflects this view. One of the prime concerns relating to the effective usage of the system is the issue of scheduling/mapping the application task onto the system. Indeed, the SOS approach is scheduling-driven and it deals with scheduling/mapping as a primary issue. The synthesis technique described here produces a custom multiprocessor system, maps the subtasks onto the system and provides a static schedule for the task execution.

SOS assumes the application domain is specified in terms of a task data flow graph (Figure 1). 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 system which meets various cost and performance constraints. The multiprocessor system is specified in terms of a set of processors and the interconnections between them. An example system is shown in Figure 2. Synthesizing a system involves making decisions about the number and types of processors, the overall interconnection between the processors, and the scheduling of subtasks on the processors.

The SOS approach involves creation of a formal model of the multiprocessor synthesis problem using mathematical programming and the solution of this model. This approach is a natural outgrowth of the work described by Chu [6], Talukdar [28], and Hafer [18]. Hafer's notation has been adopted wherever meaningful to do so. *Our research focuses on the automatic design of the multiprocessor system itself, not merely the mapping of tasks onto a given system.* The SOS approach can be used to explore different interconnection styles; e.g., bus, point-to-point, ring, or a mixture of these. SOS assumes there is no global clock and communications between subtasks are asynchronous at the task level. A distinguishing feature of the research is the fact that SOS designs a truly heterogeneous system, which allows a more precise tailoring of the synthesized system to a specific application. SOS is



$$f_R(i_{1,1}) = 0.25 \qquad f_A(o_{1,1}) = 0.50$$
$$f_R(i_{2,1}) = 0.25 \qquad f_A(o_{1,2}) = 0.75$$
$$f_R(i_{3,1}) = 0.25 \qquad f_A(o_{2,1}) = 0.50$$
$$f_R(i_{3,2}) = 0.50 \qquad f_A(o_{2,2}) = 0.75$$
$$f_R(i_{4,1}) = 0.25 \qquad f_A(o_{3,1}) = 0.75$$
$$f_R(i_{4,2}) = 0.50 \qquad f_A(o_{4,1}) = 0.75$$

Figure 1: Example 1 Task Graph

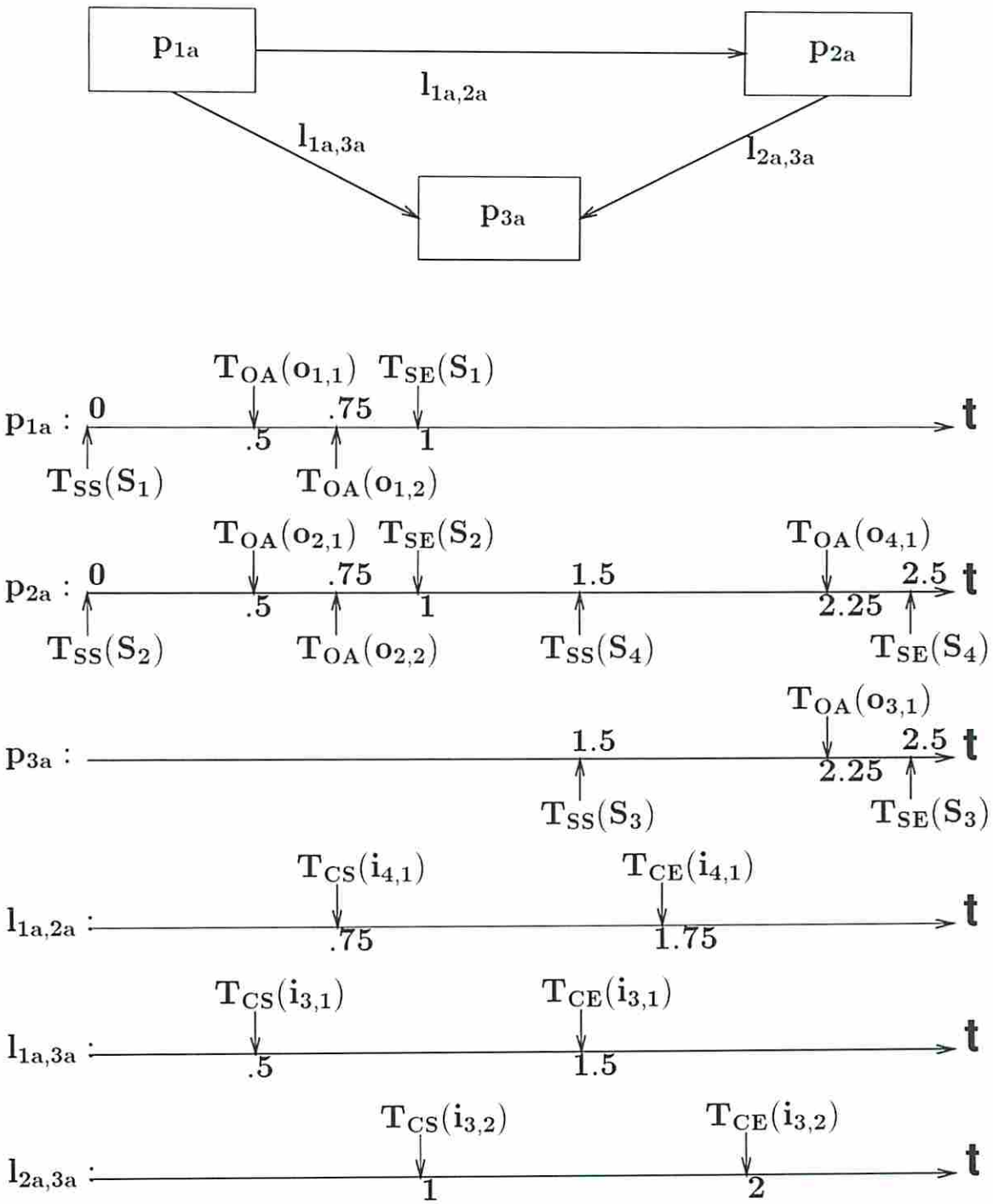


Figure 2: Synthesized Multiprocessor System I and Schedule for Example 1

capable of handling two types of heterogeneity:

- *Heterogeneity of Type-I:* This refers to heterogeneity in terms of the functionality of the processors. Two processors could be different in terms of the types of subtasks they are capable of performing. This notion allows inclusion of special-purpose processors in the system.
- *Heterogeneity of Type-II:* This refers to heterogeneity in terms of the cost-speed characteristics of the processors. Two processors could be capable of performing a given subtask, but the speeds of execution are different.

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 any function which can be linearized; e.g., the total system cost, or the overall system performance. The set of constraints consists of the correctness 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. The correctness constraints consist primarily of the relations that ensure proper ordering of the subtasks and the data transfers, taking into account the timing involved and the relations that express the conditions for complete and correct system configuration. In order to express the various constraints and the objective function, SOS defines certain variables related to the system. The necessary variables fall into two basic categories: the *timing variables* (real variables which represent timings of various critical events in the operation of the system) and the *binary variables* (0-1 variables which represent the implementation decisions regarding the system configuration). Several constraints comprising the mathematical programming model turn out to be non-linear relations. These relations are linearized and the model is converted into an MILP (Mixed Integer-Linear Programming) formulation. *Bozo*[1] [17] solves the MILP model by invoking a commercial linear programming package, XLP, developed by XMP Software, Inc.

SOS is being developed as part of the USC (Unified System Construction) Project [31]. The strength of SOS lies in the fact that it is quite general and flexible. The approach allows us to modify, extend and enhance the model to include more design possibilities and variations easily. The exact form of constraints used can be tailored to meet the characteristics of the design problem at hand. Also, the approach offers a great degree of flexibility in handling arbitrary constraints as they can be expressed using the timing and binary variables defined in the model.

[1] A branch-and-bound program to solve MILP problems which has been developed by L. J. Hafer of Simon Fraser Univ.

The organization of the rest of the paper is as follows. Section 2 briefly surveys some of the related research. Section 3 describes the SOS approach in detail by applying it to a specific style of design; Section 3.1 presents the computation model used to represent the application task to be executed on the system, Section 3.2 describes the architecture style chosen for the system to be designed, Section 3.3 describes the complete SOS mathematical programming model for the style under consideration, and Section 3.4 describes how to linearize the specific SOS model and solve it as an MILP problem. Section 4 describes some examples and experimental results in detail; Section 4.1 presents some synthesis experiments with an example of four subtasks, Section 4.2 describes some tradeoff studies using the same example, and Section 4.3 describes some synthesis experiments with another example (nine subtasks). Section 5 outlines possible applications of the SOS approach and how the specific SOS model can be extended to deal with more complex design problems. Finally, Section 6 summarizes the paper and concludes that SOS can be a useful tool.

2 Previous Related Research

SOS is a scheduling-driven synthesis approach for application-specific multiprocessor systems. Synthesis methodologies for such systems are not well researched yet. The related past research of others covers a broad range of topics, which we briefly review here.

The multiprocessor scheduling problem has been researched quite extensively [20, 11, 8, 1, 14, 24] (although no effort is directed towards synthesis). However, most of the solutions concentrate on homogeneous systems. Earlier solutions did not consider communication overhead. Many solutions revolve around the idea of *List Scheduling* (LS) heuristic. Fernandez and Bussell [11] propose a lower and an upper bound on the number of processors required to execute a task precedence graph in a time not exceeding the length of the critical path. They also determine a lower bound on the execution time for a given number of processors. Kasahara and Narita [24] describe heuristic algorithms, combining critical path ideas with branch-and-bound, for scheduling to minimize the execution time. However, neither of the efforts includes communication overhead. Some scheduling heuristics considering communication overhead are also reported; e.g., *ELS heuristic* [22], *ETF heuristic* [22], and *MH heuristic* [10]. Al-Mouhamed [3] proposes an approximate lower bound on the completion time, and approximate lower bounds on the number of processors and the number of communication links required to process a task precedence graph within this completion time. An approximate lower bound on the completion time is also estimated for a given number of processors.

There have been a number of research efforts directed towards the problem of task allocation for a given system. Several variants of the problem have been considered and several different approaches have been investigated. Inspired by Stone's work on the two-processor problem [36], graph-theoretic techniques have been researched for serially partitioned tasks (i.e, even though there are m subtasks, only one is active at one time); e.g., [33, 4]. In [5], Bokhari extends the prior research for serially partitioned tasks [4] to parallel tasks by explicitly taking concurrency into account. Shen and Tsai [34] model the task assignment problem as a graph matching problem. In [23], Indurkhya et al. use an analytical approach for modeling and optimization of multiprocessing execution time for random-graph models of programs. They derive an optimal task-assignment policy by optimizing the execution time and confirm intuitive results for the 2-processor and n-processor cases. Nicol [30] and Haddad [15] describe similar research. Heuristic approaches have also been applied to the task allocation problem [9, 7, 19]. Houstis [19] describes heuristic algorithms for task allocation to homogeneous bus connected systems. An iterative algorithm is also given to determine the optimal number of processors.

Mathematical Programming has also been applied to the problem of task allocation for a given system. Chu et al. [6] described an integer 0-1 programming approach to the problem. They considered the problem of optimal allocation of a set of m subtasks to a set of p (fixed) processors already interconnected in some fashion. Ma et al. [27] also report an integer programming model for task allocation. However, the models described in [6, 27] do not consider the effects of precedence relations in the data flow among the subtasks. We are using a similar approach for synthesis and we take into account the precedence relations. Nonlinear programming has been used for task allocation under an assumption that the given task can be split into arbitrary size subtasks [2, 16]. A limitation of such research is that it is not directly applicable to practical situations where partitioning can usually be done only at specific points.

One research effort by Talukdar and Mehrotra [28] is oriented towards synthesis of multiprocessors. In this work, a simplified and similar version of our problem is described, and the goal is to find a minimum execution time system which meets the system cost constraint. The problem is modeled using mathematical programming, though the solution procedure is heuristic and iterative. The core of the solution procedure consists of an interactive program that estimates the minimum execution time of the task for a given system. In this work, no explicit consideration is given to the delays and costs associated with the communication links. Our research models the communication links explicitly.

Mathematical programming has also been applied to the data path synthesis problem. Hafer and Parker [18] 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. Hwang et al. [21] have described an integer linear programming model for the scheduling problem in data path synthesis under resource constraints and time constraints.

There have been research efforts directed towards synthesis of array-processor architectures [25, 12, 29, 26]. Such architectures are usually characterized by synchronized operation, and all the processors perform nearly identical and relatively simple computations. Synchronous operation makes array-processors best-suited for computations displaying reasonably regular structure and flow of data. However, more often a task contains a mix of quite different subtasks with quite different processing requirements and consequently heterogeneous systems are more suitable. Our research is geared towards the synthesis of such heterogeneous systems.

Superconcurrency [13] is a research effort directed towards heterogeneous processing. It is a form of distributed heterogeneous processing developed to support a variety of Navy High Performance Computing requirements. It is a general technique for matching and managing a heterogeneous suite of super-speed processors.

3 A Specific SOS Model

As mentioned in Section 1.2, the SOS approach is quite general and it can be applied to several varying design situations and design styles. The approach involves creation of a mathematical programming model and solution of the model. Depending on the specific design problem at hand, a specific model can be created. The challenge is to come up with the specific mathematical programming model applicable to the design problem at hand. The answer lies in the general characteristics of the application tasks as well as the systems being considered. Let us refer to the relevant characteristics of the application tasks as the *task model*, and those of systems as the *system model*. The task model describes the computational model underlying the application task. The system model describes the style of architecture used to implement the task. The task model and the system model together determine the mathematical programming model to be created. The application task itself determines the specific constraints to be used to solve a particular problem. Having discussed the SOS approach in general above, we now describe application of the approach to a specific task model and system model.

3.1 A Representative Task Model

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 (directed acyclic graph) as shown in Figure 1. The subtask nodes are labelled S_1, S_2, etc. (S_a in general). The input end of a data arc is labelled $i_{a,b}$ if it provides the b^{th} input to subtask S_a, and the output end is labelled $o_{a,c}$ if it transmits the c^{th} output from the subtask S_a. Although we represent the task by a data flow graph, we consider a subtle distinction between our model and the traditional model. 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 was 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 is the fraction of the subtask S_a that 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.

For each subtask S_a, a set P_a represents the set of processors capable of executing 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.

3.2 A Representative System Model

The multiprocessor system is specified in terms of the processors selected and the interconnection architecture between them. For the specific style under consideration, we assume 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}. Each processor is assumed to have local memory, and all the interprocessor communication takes place by message-passing over communication links. The subtasks get executed on the processors and the necessary data transfers take place over the communication links.

A processor could be executing at most one subtask at any given time. Also, one and only one processor performs a given subtask. So, once execution of a subtask begins on a

processor, the subtask occupies the processor for an uninterrupted duration of time before it completes. The length of the duration is equal to the execution time of the subtask, which depends on the processor type on which it is performed. A parameter, denoted as $D_{PS}(P_t, S_a)$, specifies the execution time for the subtask S_a if the processor type P_t is selected to perform it. Hence, if two subtasks are to be executed by the same processor, one must be scheduled to begin after the other is completed.

The data transfer corresponding to an arc from node S_{a1} to node S_{a2} may be a *remote transfer* (if S_{a1} and S_{a2} are mapped to different processors); or it may be a *local transfer* within the same processor (if S_{a1} and S_{a2} are mapped to the same processor). Delay associated with a data transfer depends on whether it is a remote transfer or a local transfer[2]. 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. In practice, the time spent in performing a remote data transfer depends on the amount of traffic in the interconnection network; if two data transfers are supposed to take place over the same communication link at the same time, then the second can only start after the first is completed (The second set of data will remain held in the local memory of the processor producing it). Essentially, the time spent in remote transfer consists of the *waiting time* and the *actual transfer time*. The parameter D_{CR} only captures the actual transfer time component. The waiting time component is captured in the mathematical programming model as a delay in scheduling the communication by enforcing exclusion in the usage of the communication links. Similar to subtask execution, once a data transfer operation begins, the communication link is released only after the operation is completed; i.e., the link is busy for an uninterrupted duration of $D_{CR}V_{a1,a2}$ time units if $V_{a1,a2}$ is the volume of data associated with the operation.

Our system model assumes overlap between computation and I/O operations. A subtask can produce output data at intermediate points of its execution. Transfer of such output data can start as soon as it is available (obviously, only if required communication links are available) without delaying the completion of the subtask. It is not necessary to wait for the completion of the subtask before starting the transfer of the output data, since it is assumed the processor executing this subtask does not get involved in the data transfer operation. Data transfer operations are taken care of by I/O modules. We assume for this specific model that each processor in the system will have the necessary I/O modules. Similarly, the processor receiving the data does not get involved in the data transfer operation, and could be performing computations while the data is being received by its I/O module (and so the inputs of the subtask could arrive after the processor has already started executing

[2]Local transfer delay could be negligible compared to the remote transfer delay.

the subtask).

A set P represents the set of all the processors (with varying functionality, cost and performance) available for selection as part of the synthesized system, where $P = \bigcup_a P_a$. Associated with each processor $p_d \in P$ is a parameter C_d which specifies the cost of the processor. C_L specifies the cost of creating a communication link between two processors.

3.3 The Mathematical Programming Model

3.3.1 The Constraints

In order to express the various constraints, the following variables need to be defined.

Timing Variables: There are three classes of timing variables.

- *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 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 transfer of the data required by input $i_{a,b}$ of subtask S_a ends.

Binary Variables: There are two types of binary variables.

- *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 as follows.

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:

$$\sum_{d|p_d \in P_a} \sigma_{d,a} = 1 \tag{3.3.1}$$

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:

$$\gamma_{a1,a2} = 1 - \sum_{d|p_d \in P_{a1} \cap P_{a2}} \sigma_{d,a1}\sigma_{d,a2} \tag{3.3.2}$$

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:

$$T_{IA}(i_{a,b}) = T_{CE}(i_{a,b}) \tag{3.3.3}$$

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:

$$T_{OA}(o_{a,c}) = T_{SS}(S_a) + f_A(o_{a,c})(T_{SE}(S_a) - T_{SS}(S_a)) \tag{3.3.4}$$

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:

$$T_{IA}(i_{a,b}) \leq T_{SS}(S_a) + f_A(i_{a,b})(T_{SE}(S_a) - T_{SS}(S_a)) \tag{3.3.5}$$

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 type being used for it. *A priori* we do not know which processor type 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 (where $Typ(p_d)$ represents the type of the processor p_d). The summation acts as a selection since only one $\sigma_{d,a} = 1$ for each a:

$$T_{SE}(S_a) = T_{SS}(S_a) + \sum_{d|p_d \in P_a} \sigma_{d,a} D_{PS}(Typ(p_d), S_a) \tag{3.3.6}$$

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. Except for external inputs, for each input data $i_{a2,b2}$ (to the subtask S_{a2}) being supplied by another subtask's output, if the output supplying the data is $o_{a1,c1}$, the following relation must be satisfied by $T_{CS}(i_{a2,b2})$, the start of the data transfer:

$$T_{CS}(i_{a2,b2}) \geq T_{OA}(o_{a1,c1}) \tag{3.3.7}$$

Data-transfer-end constraint: The time at which transfer of data ends, T_{CE}, 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:

$$T_{CE}(i_{a2,b2}) = T_{CS}(i_{a2,b2}) + \gamma_{a1,a2} D_{CR} V_{a1,a2} + (1 - \gamma_{a1,a2}) D_{CL} V_{a1,a2} \tag{3.3.8}$$

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 [18]). 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:

$$\sigma_{d,a1}\sigma_{d,a2}L([T_{SS}(S_{a1}), T_{SE}(S_{a1})], [T_{SS}(S_{a2}), T_{SE}(S_{a2})]) = 0 \tag{3.3.9}$$

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}). So, for each processor pair (p_{d1}, p_{d2}) and each pair of inputs $i_{a1,b1}$ and $i_{a2,b2}$, if the input $i_{a1,b1}$ is being supplied from S_{a3} to S_{a1} and the input $i_{a2,b2}$ from S_{a4} to S_{a2} then 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:

$$\sigma_{d2,a1}\sigma_{d2,a2}\sigma_{d1,a3}\sigma_{d1,a4}L([T_{CS}(i_{a1,b1}), T_{CE}(i_{a1,b1})], [T_{CS}(i_{a2,b2}), T_{CE}(i_{a2,b2})]) = 0 \tag{3.3.10}$$

The above constraint also captures the waiting times associated with the remote data transfers.

3.3.2 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 frequently measured by how fast the system can perform the given task, 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):

$$T_F \geq T_{SE}(S_a) \tag{3.3.11}$$

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 as follows.

Processor-selection variable, β_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

$$MINIMIZE \sum_{d|p_d \in P} \beta_d C_d + \sum_{d1,d2|p_{d1} \in P \wedge p_{d2} \in P} \chi_{d1,d2} C_L$$

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 3.2. The variables of type β_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 β_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 $p_d \in P_a$):

$$\beta_d \geq \sigma_{d,a} \tag{3.3.12}$$

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 $(\sigma_{d1,a1}\sigma_{d2,a2})$, 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}):

$$\chi_{d1,d2} \geq \sigma_{d1,a1}\sigma_{d2,a2} \tag{3.3.13}$$

The essence of the model has been presented. It is easy to see that *arbitrary constraints imposed by the designer (within the semantics of the model) can be expressed using the timing and binary variables defined in the model.*

3.4 Synthesis Using the Model

3.4.1 Linearization of the Model

Several constraints comprising the mathematical programming model presented in Section 3.3 are non-linear relations. In order to solve the model as an MILP, these relations must be linearized and the model converted into an MILP formulation.

Eq. (3.3.2) is non-linear. It can be linearized by defining a binary variable of the form $\delta_{d,a1,a2}$ for each product of the form $(\sigma_{d,a1}\sigma_{d,a2})$. Using the new variables defined, Eq. (3.3.2) can be replaced by the following set of linear relations:

$$\gamma_{a1,a2} = 1 - \sum_{d|p_d \in P_{a1} \cap P_{a2}} \delta_{d,a1,a2} \tag{3.4.14}$$

$$\delta_{d,a1,a2} \leq \sigma_{d,a1} \tag{3.4.15}$$

$$\delta_{d,a1,a2} \leq \sigma_{d,a2} \tag{3.4.16}$$

Now, let us consider linearization of Eq. (3.3.9). The constraint says that if two subtasks S_{a1} and S_{a2} are executed on the same processor p_d, then there must be no overlap in their execution time intervals. This implies that either the start time of S_{a1} is sometime after the completion time of S_{a2} or the start time of S_{a2} is sometime after the completion time of S_{a1}. Let us define a binary variable $\alpha_{a1,a2}$ whose value decides which of the two possibilities occurs. $\alpha_{a1,a2} = 1$ implies S_{a1} is executed first. Using this variable, Eq. (3.3.9) can be rewritten as the following pair of non-linear relations:

$$\begin{aligned} T_{SS}(S_{a2}) &\geq \alpha_{a1,a2}\sigma_{d,a1}\sigma_{d,a2}T_{SE}(S_{a1}) \\ T_{SS}(S_{a1}) &\geq (1 - \alpha_{a1,a2})\sigma_{d,a1}\sigma_{d,a2}T_{SE}(S_{a2}) \end{aligned}$$

To linearize the above pair, let us define a constant T_M whose value is larger than the possible values of all the timing variables in the model. Now, the following two linear relations express the desired constraint:

$$T_{SS}(S_{a2}) \geq T_{SE}(S_{a1}) - (3 - \alpha_{a1,a2} - \sigma_{d,a1} - \sigma_{d,a2})T_M \quad (3.4.17)$$

$$T_{SS}(S_{a1}) \geq T_{SE}(S_{a2}) - (2 + \alpha_{a1,a2} - \sigma_{d,a1} - \sigma_{d,a2})T_M \quad (3.4.18)$$

Similarly, to linearize Eq. (3.3.10), we need to define a binary variable $\phi_{a1,b1,a2,b2}$. $\phi_{a1,b1,a2,b2} = 1$ indicates the data transfer for $i_{a1,b1}$ takes place before the data transfer for $i_{a2,b2}$ if the same communication link is used for both the transfers. Using this variable, Eq. (3.3.10) can be rewritten as the following pair of linear relations:

$$T_{CS}(i_{a2,b2}) \geq T_{CE}(i_{a1,b1}) - (5 - \phi_{a1,b1,a2,b2} - \sigma_{d2,a1} - \sigma_{d2,a2} - \sigma_{d1,a3} - \sigma_{d1,a4})T_M \quad (3.4.19)$$

$$T_{CS}(i_{a1,b1}) \geq T_{CE}(i_{a2,b2}) - (4 + \phi_{a1,b1,a2,b2} - \sigma_{d2,a1} - \sigma_{d2,a2} - \sigma_{d1,a3} - \sigma_{d1,a4})T_M \quad (3.4.20)$$

Finally, non-linear Eq. (3.3.13) can be simply rewritten in the following linear form:

$$\chi_{d1,d2} \geq \sigma_{d1,a1} + \sigma_{d2,a2} - 1 \quad (3.4.21)$$

3.4.2 Solution of the Model

The linearized MILP model is solved using the *Bozo* program [17]. As a result of solving the model, we get the following information as outputs:

- a multiprocessor system; i.e., the chosen set of processors and the interconnection architecture,
- a schedule for the subtasks, and
- detailed timing information for computation and transfer of data.

4 Experiments and Results

The specific model described in Section 3 was used to experiment with 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 [28].

Table I

Processor Characteristics - Example 1

		Execution Time			
Proc.	Cost	S_1	S_2	S_3	S_4
p_1	4	1	1	-	3
p_2	5	3	1	2	1
p_3	2	-	3	1	-

4.1 Example 1: Four-Subtask 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, and p_3. The costs of these processors and the execution times of various subtasks on the processors are given in Table I. 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 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 21 timing and 72 binary variables, and 174 constraints. Bozo was used to generate 4 non-inferior[3] systems. These different systems 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 on the order of a few seconds. These runtimes are on a Solbourne Series5e/900 (similar to Sun SPARCsystem 4/490) with 128 MB of memory. Cost, performance and runtime for the four designs are given in Table II. 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 subtask S_1, processor p_{2a} performs subtasks S_2 and S_4 in that order, and processor p_{3a} performs subtask

[3]A system (characterized by its cost and performance) is considered non-inferior if cost (performance) can not be improved without degrading performance (cost).

Table II

Example 1 Systems

Design	Runtime (*sec*)	Cost	Performance
1	11	14	2.5
2	24	13	3
3	28	7	4
4	37	5	7

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 system 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.

4.2 Tradeoff Studies Using the Four-Subtask Graph

Some tradeoff studies were performed using the example in Section 4.1. We studied the role of inter-subtask communication in synthesis of the systems. The study was performed by varying the ratio between communication times and execution times.

4.2.1 Experiment 1: Increase the Communication Time

In this experiment, we increased the volume of data to be transferred for each arc in the four-subtask task graph. All the other parameters remained as given in Section 4.1.

When the volume of data is doubled for each of the arcs (i.e., the volume is two units instead of one), 3-processor designs become inferior. Only two designs remain non-inferior: the 2-processor design and the uniprocessor design.

The 2-processor design also becomes inferior when the volume of data is made six times the original volume (i.e., the volume is six units for each arc). Only the uniprocessor design remains non-inferior.

4.2.2 Experiment 2: Increase the Execution Time

In this experiment, we increased the size of each of the subtasks (and thus the execution times in Table I). All the other parameters remained as given in Section 4.1.

When the size of each of the subtasks is doubled (and hence the execution time is doubled for each of the processor types), the number of non-inferior designs becomes five (instead of four). The new non-inferior design is a 3-processor design. This design consists of 3 processors: p_{1a}, p_{1b} - two processors of type p_1, and p_{3a} - a processor of type p_3. Processor p_{1a} performs subtasks S_1 and S_2 in that order, processor p_{1b} performs subtask S_4, and processor p_{3a} performs subtask S_3. There are two communication links: $l_{1a,1b}$, and $l_{1a,3a}$. Data $i_{4,1}$ gets transmitted on link $l_{1a,1b}$, and data $i_{3,2}$ and data $i_{3,1}$ get transmitted on link $l_{1a,3a}$ in that order.

When the size of each of the subtasks is further increased and made three times the original size, the number of non-inferior designs becomes seven (as opposed to five for the double-size case above). The two new additions are a 4-processor design and a new 2-processor design. The 4-processor design consists of p_{1a}, p_{1b} - two processors 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 subtask S_1, processor p_{1b} performs subtask S_2, processor p_{2a} performs subtask S_4, and processor p_{3a} performs subtask S_3. There are three communication links: $l_{1a,2a}$, $l_{1a,3a}$, and $l_{1b,3a}$. 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 $l_{1b,3a}$. The new 2-processor design consists of p_{1a} - a processor of type p_1, and p_{2a} - a processor of type p_2. Processor p_{1a} performs subtask S_1 and S_2 in that order, and processor p_{2a} performs subtasks S_3 and S_4 in that order. There is a

Table III

Processor Characteristics - Example 2

		Execution Time								
Proc.	Cost	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
p_1	4	2	2	1	1	1	1	3	–	1
p_2	5	3	1	1	3	1	2	1	2	1
p_3	2	1	1	2	–	3	1	4	1	3

communication link: $l_{1a,2a}$. Data $i_{3,1}$, data $i_{3,2}$ and data $i_{4,1}$ get transmitted on link $l_{1a,2a}$ in that order.

The results of experiments 1 and 2 essentially indicate what is known intuitively, that as the ratios between the inter-subtask communication (in time units) and the sizes of subtasks(in time units) increase, designs with fewer processors are synthesized as they can achieve the desired performance with lower costs. However, as the sizes of subtasks increase (and thus inter-subtask communication becomes relatively less important), multiprocessing becomes useful and designs with more processors are synthesized as they can provide better performance.

4.3 Example 2: Nine-Subtask 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 III. The volume of data is one unit for each arc. We are given: $D_{CL} = 0$, $D_{CR} = 1$. For this graph, we synthesized systems for two different styles of interconnection.

4.3.1 Point-to-Point Interconnection Experiments

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 47 timing and 225 binary variables, and 1081 constraints. We generated 5 non-inferior

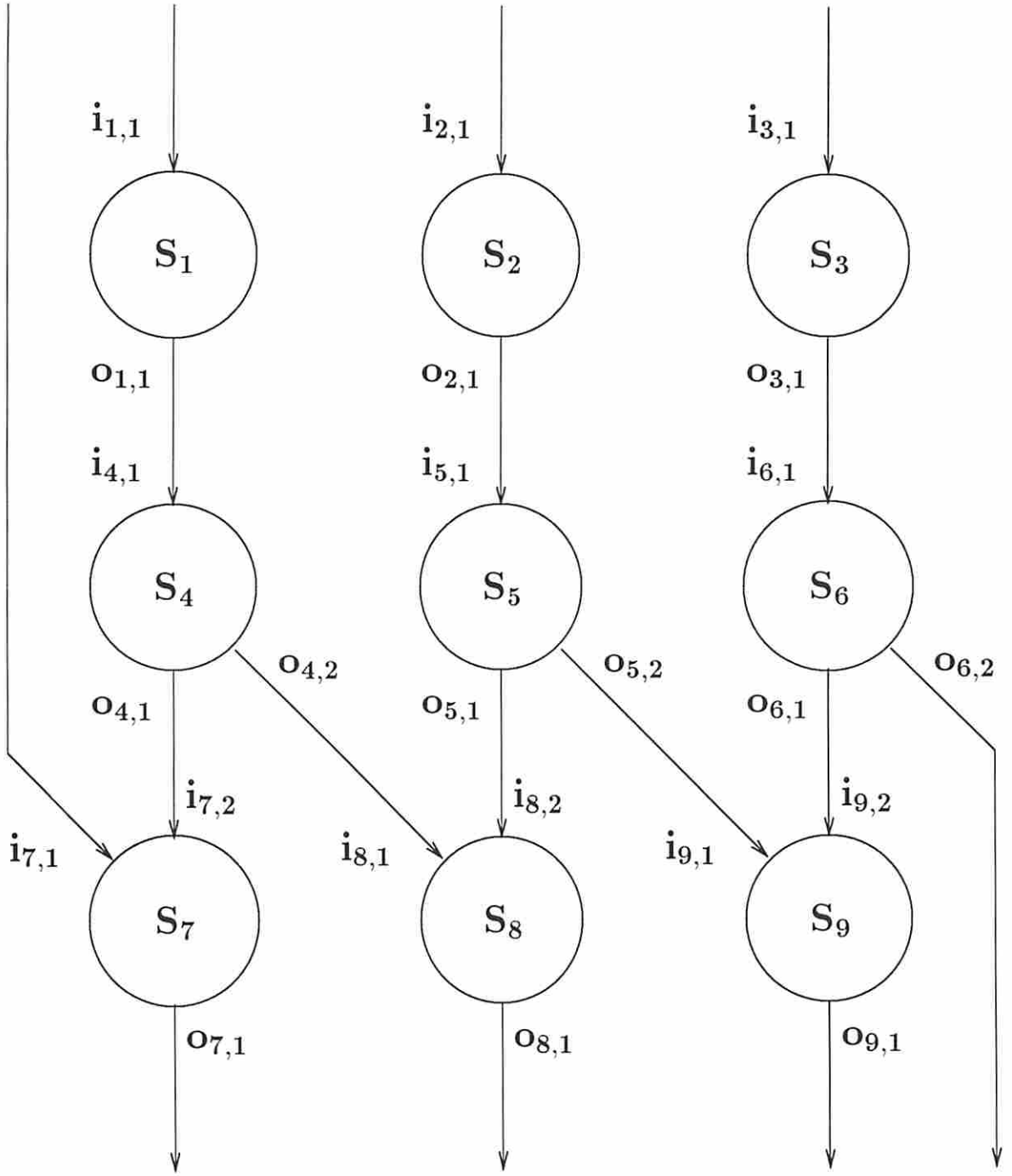


Figure 3: Example 2 Task Graph

Table IV

Example 2 Systems (Point-to-Point)

Design	Runtime (min)	Cost	Performance
1	62.2	15	5
2	445.17	12	6
3	538.67	8	7
4	75.18	7	8
5	6416.87	5	15

systems by changing the constraint value for the total system cost, and optimizing the system performance. Bozo's runtime for each of these designs is on the order of a few hours, except for design 5. Cost, performance and runtime for the five designs are given in Table IV. A brief discussion of the 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 subtasks S_3, S_6 and S_4 in that order, processor p_{2a} performs subtasks S_2, S_5, S_9 and S_7 in that order, and processor p_{3a} performs subtasks S_1 and S_8 in that order. There are four communication links: $l_{1a,2a}$, $l_{1a,3a}$, $l_{2a,3a}$, and $l_{3a,1a}$. Data $i_{9,2}$ and data $i_{7,2}$ get transmitted on link $l_{1a,2a}$ in that order, data $i_{8,1}$ gets transmitted on link $l_{1a,3a}$, data $i_{9,1}$ gets transmitted on link $l_{2a,3a}$, and data $i_{4,1}$ gets transmitted on link $l_{3a,1a}$.

Design 2: This design also consists of 3 processors: p_{1a}, p_{1b} - two processors of type p_1, and p_{3a} - a processor of type p_3. Processor p_{1a} performs subtasks S_1, S_4 and S_7 in that order, processor p_{1b} performs subtasks S_3, S_6 and S_9 in that order, and processor p_{3a} performs subtasks S_2, S_5 and S_8 in that order. There are two communication links: $l_{1a,3a}$, and $l_{3a,1b}$. Data $i_{8,1}$ gets transmitted on link $l_{1a,3a}$, and data $i_{9,1}$ gets transmitted on link $l_{3a,1b}$.

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_3, S_6, S_4, S_7 and S_9 in that order, and processor p_{3a} performs subtasks S_1, S_2, S_5 and S_8 in that order. There are two communication links: $l_{1a,3a}$, and $l_{3a,1a}$. Data $i_{8,1}$ gets transmitted on link $l_{1a,3a}$, and data $i_{4,1}$ and data $i_{9,1}$ get transmitted on link $l_{3a,1a}$ in that order.

Table V

Example 2 Systems (Bus-Style)

Design	Runtime (min)	Cost	Performance
1	107.3	10	6
2	89.53	6	7
3	61.52	5	15

Design 4: This design is similar to design 3, and also consists of 2 processors: p_{1a}, and p_{3a}. However, it has only one link: $l_{1a,3a}$. Presence of only one link forces a change in the mapping between the resources and the events. Processor p_{1a} performs subtasks S_3, S_6, S_1, S_4 and S_7 in that order, and processor p_{3a} performs subtasks S_2, S_5, S_9 and S_8 in that order. Data $i_{9,2}$ and data $i_{8,1}$ get transmitted on link $l_{1a,3a}$ in that order.

Design 5: 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_4, S_5, S_8, S_3, S_7, S_6 and S_9 in that order.

4.3.2 Bus-Style Interconnection Experiments

In this interconnection style, the system consists of a set of processors and a bus connecting all the processors to each other. The cost of the system is dominated by the costs of the processors selected. The SOS approach is capable of modeling such a system. The MILP bus-architecture model for the nine-subtask graph example consists of 47 timing and 153 binary variables, and 416 constraints. Three non-inferior systems were generated by changing the constraint value for the total system cost, and optimizing the system performance. Runtime for each of these designs is on the order of a few hours. Table V gives the statistics for the three designs. A brief discussion of the designs follows.

Design 1: This design consists of 3 processors: p_{1a}, p_{1b} - two processors of type p_1, and p_{3a} - a processor of type p_3. Processor p_{1a} performs subtasks S_1, S_4 and S_7 in that order, processor p_{1b} performs subtasks S_3, S_6 and S_9 in that order, and processor p_{3a} performs subtasks S_2, S_5 and S_8 in that order. Data $i_{8,1}$ and data $i_{9,1}$ get transmitted on the common bus in that order.

Design 2: 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_3, S_6, S_4, S_7 and S_9 in that order,

and processor p_{3a} performs subtasks S_1, S_2, S_5 and S_8 in that order. Data $i_{4,1}$, data $i_{8,1}$ and data $i_{9,1}$ get transmitted on the common bus in that order.

Design 3: 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_4, S_3, S_5, S_8, S_6, S_9 and S_7 in that order.

5 Application of the Approach to Other Design Scenarios

In Section 3, we described application of the SOS approach to a specific task model and system model. In Section 4, we reported the supporting experimental results for the specific model. It must be emphasized that the approach can be applied to several other design scenarios. As reported in Section 4.3.2, the approach has already been applied to a system model for bus-style interconnection, and the corresponding MILP model can be found in [32]. The approach can be applied to other interconnection styles. The MILP model for ring interconnection is being developed. Future work would involve development of the MILP models for different interconnection styles and their combinations.

As the reader might notice, in the specific model of Section 3, each processor is assumed to have local memory, but the cost of memory is not included in the system cost explicitly. The MILP model has been extended to perform local memory design. The extended model takes care of such cost explicitly. Solution of the model would output how much local memory is required at each processor in the synthesized system. Shared-memory systems can also be modeled using the SOS approach. The MILP model for shared-memory systems is being developed. Future work would involve dealing with more complex memory issues/structures.

The specific model of Section 3 assumes overlap between computation and I/O operations. An MILP model can also be developed for the situation when such an overlap is not possible. Again, the specific model does not explicitly consider the costs associated with the I/O modules. Future work would involve extension of the model to handle costs associated with I/O modules and memory buffers required at the I/O modules.

6 Conclusions

In this paper, we have presented a scheduling-driven approach for synthesizing optimal multiprocessor systems for given applications. The approach is applicable to several design situations, and we discussed how to apply it to a specific task model and system model. The crux of the approach lies in a mathematical programming model reflecting the design problem.

Several experiments have been conducted using the approach, and the results are reported. Most of the experiments were performed for the specific model discussed in the paper. The experiments indicate that the approach can indeed be used for synthesizing different systems for a given application, depending on the cost-performance requirements imposed by the designer. The approach has also been applied to a system model for bus-style interconnection, and some experiments with this model are reported.

Some tradeoff studies were also performed to study the role of inter-subtask communication. The reported results verify the intuitive expectation that heavy inter-subtask communication leads to designs with fewer processors and multiprocessing is more useful only when inter-subtask communication is reasonable.

References

1. Adam, T. L., Chandy, K. M., and Dickson, J. R. A comparison of list schedules for parallel processing systems. *Communications of the ACM* **17**, 12 (Dec. 1974), 685–690.
2. Agrawal, R. and Jagadish, H. V. Partitioning techniques for large-grained parallelism. *IEEE Transactions on Computers* **37**, 12 (Dec. 1988), 1627–1634.
3. Al-Mouhamed, M. A. Lower bound on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Transactions on Software Engineering* **16**, 12 (Dec. 1990), 1390–1401.
4. Bokhari, S. H. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Transactions on Software Engineering* **SE-7**, 6 (Nov. 1981), 583–589.
5. Bokhari, S. H. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers* **37**, 1 (Jan. 1988), 48–57.
6. Chu, W. W., Hollaway, L. J., Lan, M.-T., and Efe, K. Task allocation in distributed data processing. *Computer* **13**, 11 (Nov. 1980), 57–69.

7. Chu, W. W. and Lan, L. M.-T. Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers* **C-36**, 6 (June 1987), 667–679.

8. Coffman, E. G., Jr. and Denning, P. J. *Operating Systems Theory.* Prentice-Hall, Englewood Cliffs, N.J., 1973.

9. Efe, K. Heuristic models of task assignment scheduling in distributed systems. *Computer* **15**, 6 (June 1982), 50–56.

10. El-Rewini, H. and Lewis, T. G. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing* **9** (1990), 138–153.

11. Fernandez, E. B. and Bussell, B. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Transactions on Computers* **C-22**, 8 (Aug. 1973), 745–751.

12. Fortes, J. A. B. and Moldovan, D. I. Parallelism detection and transformation techniques useful for VLSI algorithms. *Journal of Parallel and Distributed Computing* **2** (May 1985), 277–301.

13. Freund, R. F. Superconcurrent Processing, a dynamic approach to heterogeneous parallelism. *Proceedings of the Parallel/Distributed Computing Networks Seminar.* San Diego Section, IEEE, Feb. 1990.

14. Garey, M. R., Graham, R. L., and Johnson, D. S. Performance guarantees for scheduling algorithms. *Operations Research* **26**, 1 (Jan-Feb 1978), 3–21.

15. Haddad, E. K. Analysis, modeling and optimization of multiprocessing execution time. Tech. Rep. TR 89-11, Department of Computer Science, Virginia Polytechnic Institute and State University, Falls Church, VA, 1989.

16. Haddad, E. K. Partitioned load allocation for minimum parallel processing time. *Proceedings 1989 International Conference on Parallel Processing.* IEEE Computer Society, Aug. 1989.

17. Hafer, L. J. and Hutchings, E. Bringing up Bozo. Tech. Rep. CMPT TR 90-2, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, Mar. 1990.

18. Hafer, L. J. and Parker, A. C. A formal method for the specification, analysis, and design of register-transfer level digital logic. *IEEE Transactions on Computer-Aided Design* **CAD-2**, 1 (Jan. 1983), 4–17.

19. Houstis, C. E. Module allocation of real-time applications to distributed systems. *IEEE Transactions on Software Engineering* **16**, 7 (July 1990), 699–709.

20. Hu, T. C. Parallel sequencing and assembly line problems. *Operations Research* **9** (Nov. 1961), 841–848.

21. Hwang, C.-T., Lee, J.-H., and Hsu, Y.-C. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design* **10**, 4 (Apr. 1991), 464–475.

22. Hwang, J.-J., Chow, Y.-C., Anger, F. D., and Lee, C.-Y. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.* **18**, 2 (Apr. 1989), 244–257.

23. Indurkhya, B., Stone, H. S., and Xi-Cheng, L. Optimal partitioning of randomly generated distributed programs. *IEEE Transactions on Software Engineering* **SE-12**, 3 (Mar. 1986), 483–495.

24. Kasahara, H. and Narita, S. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers* **C-33**, 11 (Nov. 1984), 1023–1029.

25. Kung, S.-Y. On supercomputing with systolic/wavefront array processors. *Proceedings of the IEEE* **72**, 7 (July 1984), 867–884.

26. Li, G.-J. and Wah, B. W. The design of optimal systolic arrays. *IEEE Transactions on Computers* **C-34**, 1 (Jan. 1985), 66–77.

27. Ma, P.-Y. R., Lee, E. Y. S., and Tsuchiya, M. A task allocation model for distributed computing systems. *IEEE Transactions on Computers* **C-31**, 1 (Jan. 1982), 41–47.

28. Mehrotra, R. and Talukdar, S. N. Scheduling of tasks for distributed processors. Tech. Rep. DRC-18-68-84, Department of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, PA, Dec. 1984.

29. Miranker, W. L. and Winkler, A. Spacetime representations of computational structures. *Computing* **32**, 2 (1984), 93–114.

30. Nicol, D. M. Optimal partitioning of random programs across two processors. *IEEE Transactions on Software Engineering* **15**, 2 (Feb. 1989).

31. Parker, A. C., Küçükçakar, K., Prakash, S., and Weng, J.-P. Unified System Construction (USC). In Camposano, R. and Wolf, W.(Eds.). *High-level VLSI Synthesis.* Kluwer Academic Publishers, Boston, 1991, chapter 14, pp. 331–354.

32. Prakash, S. and Parker, A. C. A mathematical programming model for synthesis of multiprocessor systems: Linearization, an example model, and some tradeoff studies. CEng Tech. Rep. 91-17, Department of EE-Systems, University of Southern California, Los Angeles, CA, July 1991.

33. Rao, G. S., Stone, H. S., and Hu, T. C. Assignment of tasks in a distributed processor system with limited memory. *IEEE Transactions on Computers* **C-28**, 4 (Apr. 1979), 291–299.

34. Shen, C.-C. and Tsai, W.-H. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Transactions on Computers* **C-34**, 3 (Mar. 1985), 197–203.

35. Siegel, H. J., Schwederski, T., Kuehn, J. T., and Davis IV, N. J. An overview of the PASM parallel processing system. In Gajski, D. D., Milutinovic, V. M., Siegel, H. J., and Furht, B. P.(Eds.). *Computer Architecture.* IEEE Computer Society Press, Washington, D.C., 1987, pp. 387–407.

36. Stone, H. S. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering* **SE-3**, 1 (Jan. 1977), 85–93.