A Mathematical Programming Model for Synthesis of Multiprocessor Systems: Linearization, An Example Model, and Some Tradeoff Studies *

Shiv Prakash and Alice C. Parker
Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-2562

July 5, 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.

1 Introduction

With VLSI systems becoming more and more commonplace, and with myriads of new application avenues opening for VLSI systems, there is a growing need for design of hardware systems for specific applications (i.e., the systems that perform well in application-specific domains). In other words, there is a need to design hardware systems that will perform given tasks efficiently. These 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 [66]. There is a growing need to develop tools for system-level design. The research described here addresses the 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.

The goal of the described research is to develop tools and techniques aimed at producing a custom multiprocessor architecture, as well as mapping the subtasks onto the architecture and providing a schedule for the task execution. *The 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 addressing 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 [46], we believe this is the first research attempt aimed at automatic synthesis of multiprocessor systems from task specifications.

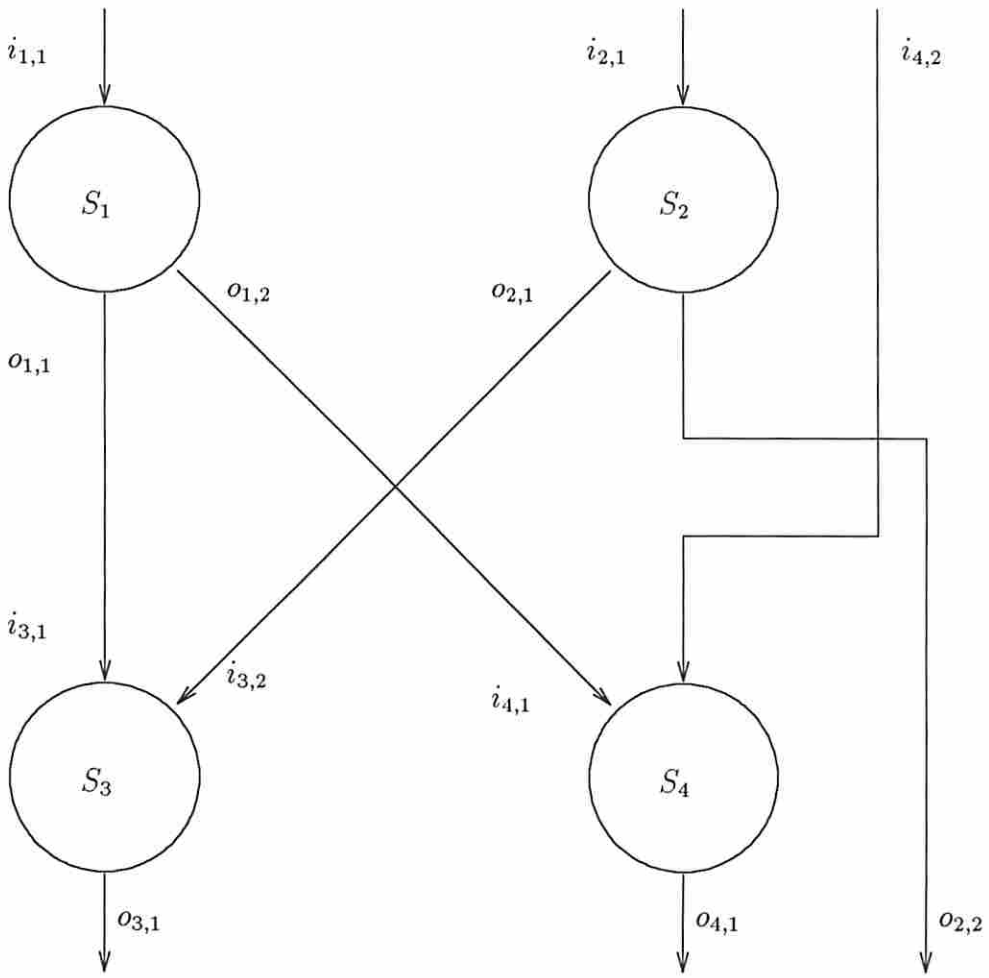
1.1 The Problem Statement

We are addressing the problem of multiprocessor architecture synthesis for a given application task. 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 exection 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.

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 may be a *remote transfer* (if the two subtasks are mapped to different processors in the synthesized system), where the data is transferred from a proccesor to another; or it maybe 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



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

Figure 1: Four-Node Task Graph

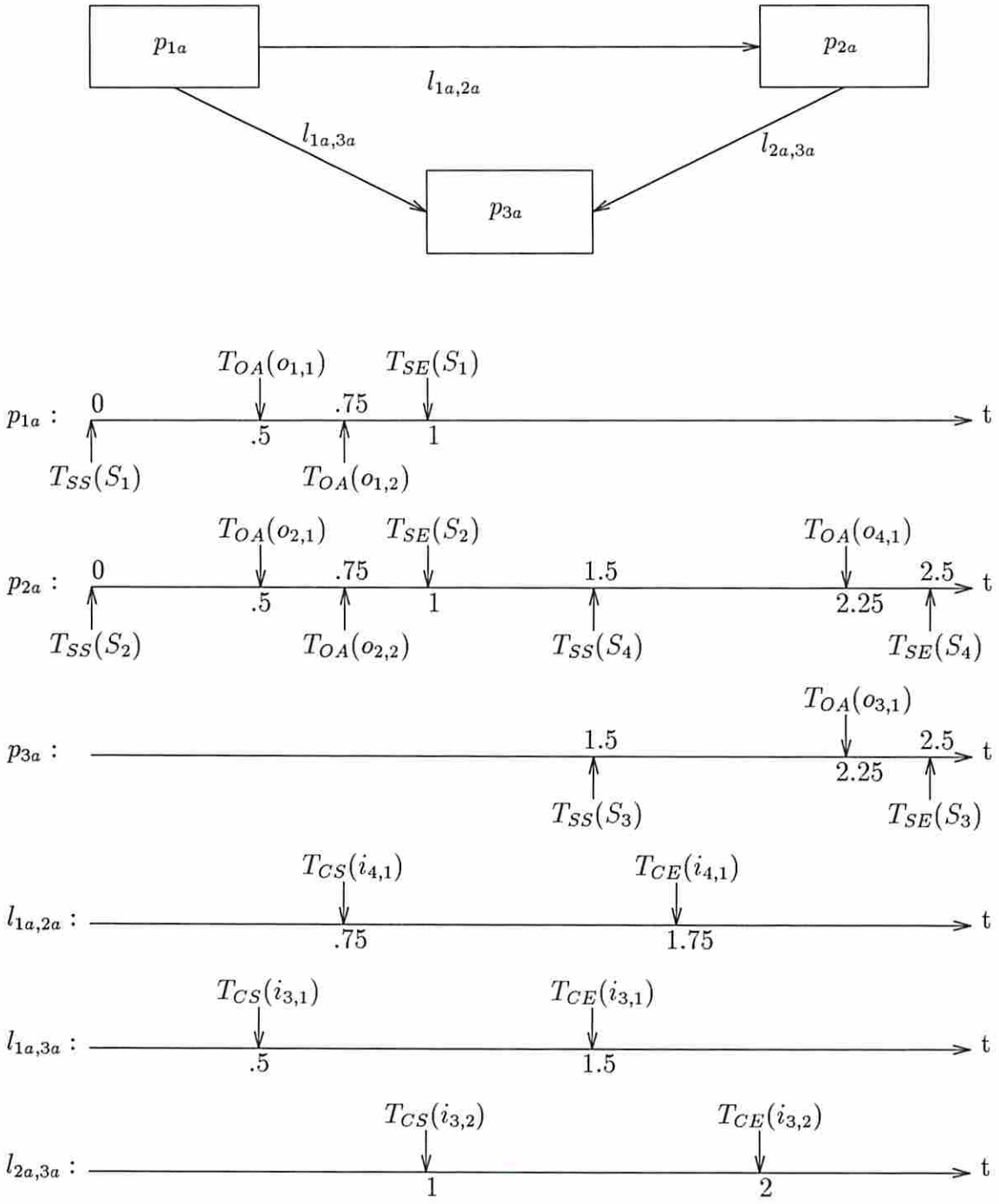


Figure 2: Synthesized Multiprocessor System I and Schedule for Four-Node Graph

delay.

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 $p_d \in P$ is a parameter C_d which specifies the cost of the processor.

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
  - 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

Our approach essentially consists of creation of a formal model of the multiprocessor synthesis problem using mathematical programming and the solution of this model.

The organization of the rest of the document is as follows. Section 2 surveys the related work. Primarily, it describes the multiprocessor task allocation research, datapath synthesis research (particularly the work where mathematical programming is used), some array-processor synthesis work, and some system-level CAD tools. Section 3 describes our model and approach to the synthesis problem in detail. Section 4 describes some examples and synthesis experiments using the approach; it concludes that the reduction of runtime is an important issue and outlines future research directions.

2 Previous Related Research

The goal of the described research is to develop tools and techniques for the design of application-specific systems (multiprocessor systems in this case), or in short, the goal is to perform *system-level synthesis*. Synthesis methodologies for application-specific systems are not well researched yet. The related past research of others covers a broad range of topics. The topics can be grouped into four categories:

- Multiprocessor research
- Datapath synthesis research
- Array-processor synthesis research
- System-level CAD tools

Our research builds on the past multiprocessor research and data path synthesis research.

2.1 Multiprocessor Research

Most multiprocessor design and theory is aimed at general-purpose computing. So, most of the previous related work on multiprocessors is directed at the problem of task allocation for a given architecture. Research has also been conducted in the areas of mapping specialized algorithms and scheduling precedence graphs onto multiprocessors. One research effort by Talukdar and Mehrotra [46, 47] is oriented towards synthesis of multiprocessors.

2.1.1 Task Allocation Work

Several variants of the task allocation problem have been considered and several different approaches have been investigated. We briefly review these efforts next.

2.1.1.1 Graph-Theoretic Approach

Serially Partitioned Tasks: 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. One such effort was Stone's work on the two-processor problem [71]. The work concentrates on the assignment of the subtasks to a system consisting of two processors (single-host, single-satellite) so as to avoid excessive interprocessor communication (IPC) while taking advantage of specific efficiencies of the processors. The goal is to minimize the collective costs due to IPC and to computation. The problem is solved efficiently using the network flow approach. The method involves the construction of a network flow graph in which nodes represent the subtasks to be assigned and edge capacities represent computation and communication costs in such a fashion that the minimum weight cut separating two distinguished nodes in the graph corresponds to the optimal assignment. An effort is also made to extend the method to three and n-processor cases, with only partial success.

In later research [68], Stone reconsidered the two-processor problem and extended the network flow techniques to examine the sequence of optimal assignments found as the load on one processor is held fixed and the load on the other is varied. The research indicated that for every subtask S (in the overall task) there exists a critical load factor f_S such that when the load on the processor with variable load is below f_S, S is assigned to that processor by an optimal assignment, and is otherwise assigned to the other processor. In other words, as the load on the processor with variable load increases, the optimal assignment is always such that subtasks move away from this processor to the other. Thus, successive optimal assignments for successively increasing loads are nested inside each other. In another research project [18], Gusfield developed a parametric computing method for combinatorial problems and applied it to the two-processor problem in the face of varying load levels on both the processors.

Several variants of Stone's two-processor problem have been researched [65, 3, 70]. In [65], Stone considers minimum cost assignment for the case where one processor has limited memory capacity. In [3], Bokhari considers the problem of finding an optimal dynamic assignment of a task. The cost of dynamically reassigning a subtask from from one processor to the other and the cost of subtask residence without execution are included in the model, and network flow algorithms are used. In [70], the basic network flow algorithm is generalized to three-processor systems. Static as well as dynamic assignments are considered. An attempt is also made to solve the problem for many-processor systems when the task structure is tree-like. The problem of assignment for tree-like structured tasks is also solved in [4], using an efficient dynamic programming approach. The *shortest tree algorithm* takes into account the interconnection structure of the system (i.e., the speeds of

links between pairs of processors), and works by constructing a weighted, layered assignment graph and finding a minimum *sum* weight path in it. Price and Pooch [63] extend the solution technique for tree-like structured tasks to allow an arbitrary subtask intercommunication pattern. The problem is modeled as a directed acyclic search graph and a shortest path algorithm is given that produces an assignment of subtasks to processors. However, the algorithm does not guarantee an optimal solution. A branch-and-bound algorithm is also described which may be applied to the search graph to produce assignments of generally inferior quality but with considerably less computational effort.

Parallel Tasks: In [6, 5], Bokhari extends the prior research for serially partitioned tasks [3, 70, 4] to parallel tasks by explicitly taking concurrency into account. A *sum-bottleneck path (SBP) algorithm* is developed that permits the efficient solution of many variants of the problem under some constraints on the structure of the partitions. Point-to-point interconnection is assumed. The system under consideration is of single-host, multiple-satellite type. The problem of partitioning multiple chain-structured parallel programs to minimize the time for execution is solved optimally by constructing an assignment graph and applying the SBP algorithm, under the constraint that each chain is partitioned into two contiguous subchains. The problem of partitioning multiple arbitrarily structured serial programs is transformed into the problem of partitioning multiple chains by use of Stone's network flow algorithm [71] for single-host, single-satellite assignments combined with his results on nested assignments [68]. The problem of partitioning single-tree structured parallel programs is solved optimally by the SBP algorithm, under the constraints that all satellites are identical and that individual maximal subtrees of the given tree are assigned to each satellite. Finally, the problem of partitioning chain-structured parallel programs across chain-connected systems is solved under the constraint that the chain is partitioned into contiguous subchains; and the solution technique is somewhat similar to the technique described in [4], except that a minimum bottleneck weight path instead of a sum weight path yields the optimal solution. This problem is also considered in [31], where the solution technique is evaluated and an alternative greedy approximation algorithm is described.

Shen and Tsai [67] model the task assignment problem as a graph matching problem. The cost function used is the maximum time for a task to complete subtask execution and communication in all the processors. The *minimax* optimization criterion is employed, which implies that the load on the bottleneck processor is minimized instead of the sum of all the processor loads and thus ensures load-balancing. The proposed approach allows consideration of various characteristics of the given system. Graphs are used to represent the subtask relationship of the given task and the processor structure of the distributed system. Subtask assignment to system processors is transformed into a type of graph matching. Although the problem is modeled as graph matching, the solution is not ob-

tained using graph-theoretic algorithms. Instead, the search of optimal task assignment (minimum-cost graph matching) is formulated as a state-space search problem which is solved by the well-known A^* algorithm in artificial intelligence [53].

The limitation of the graph-theoretic approach is that it is not easily extendible to solve the general problem of task allocation for an arbitrary number of processors. Another limitation is encountered in handling restrictions on resources (e.g., memory size restricted) and arbitrary constraints (e.g., constraint on task completion time).

2.1.1.2 Analytical Modeling Approach In [30, 69], Stone et al. use an analytical approach for modeling and optimization of multiprocessing execution time for random-graph models of programs. An optimal task-assignment policy is derived by optimizing the execution time. The execution time is modeled as the sum of the execution time of the busiest processor and the total communications overhead. The derived policy essentially says that the optimal task assignments are extremal in the sense that the cost of processing the subtasks is distributed among all processors as evenly as possible or not distributed at all, depending upon the ratio of runtimes to communication times as well as the ratios of the processing speeds of the processors. The policy holds in the case of homogeneous as well as heterogeneous (only in terms of speed, not functionality) processors. Nicol [52] describes similar research for partitioning of random programs in the two-processor case. This result, though striking, has its limitations because of the underlying assumptions made in its derivation. The model of the system assumes that there is an aggregate communication bandwidth among processors, and the total communications overhead is estimated by aggregating the traffic involved. Such a model is reasonable for systems sharing a common bus, but less acceptable for systems containing internal point-to-point connections, as the aggregation tends to obscure nonuniform traffic patterns in a point-to-point communication structure and does not account for saturation of individual point-to-point links. Another limitation arises due to the fact that a random graph is not an accurate model of programs and that the effect of precedence relations is ignored. Another severe limitation can be attributed to the fact that the communications overhead is simply added to the execution time of the busiest processor and thus the execution time formulation does not account for parallelism between communication and task execution. Finally, the optimization process used is based on certain approximations and is not exact. Haddad [19] uses a different approach for modeling of the execution time, which formulates a more accurate analytical representation. In this work, the internal inter-subtask communication times (time spent in communication between two subtasks assigned to the same processor) are also considered. However, precedence relations are ignored here also. The system consists of P heterogeneous processors and L communication channels. A result similar to Stone's policy of "even distribution or no distribution" is obtained by Haddad also under

some simplifying assumptions. One of the assumptions used is that the communication channels never get saturated. Although the results obtained by Stone and Haddad are based on certain limiting assumptions, they may still provide heuristics for our use in the application-specific domain.

2.1.1.3 Mathematical Programming Approach

Integer 0-1 Programming Approach: Chu et al. [10] considered the problem of optimal allocation (minimum overhead due to IPC) of a set of m subtasks to a set of p (fixed) processors already interconnected in some fashion, and suggested that an integer 0-1 programming approach is most likely to be useful in solving realistic task allocation problems. In this paper, the objective function for the integer 0-1 programming model is formulated as the sum of the processing costs of the subtasks on the processors assigned to them and the total IPC cost. Constraints are formulated to reflect memory size restrictions and task completion time requirements. It is a flexible technique because it allows constraints to be introduced into the model as appropriate to the application.

Ma et al. [44] also report an integer programming model for task allocation. Some extra constraints have been incorporated into the model to meet various application requirements. A branch-and-bound method is used to solve the model. For a given distributed system and a set of constraints, it generates the minimum-cost allocation.

However, the models described in [10, 44] 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 Approach for Arbitrarily Partitionable Tasks: Nonlinear programming has been used for task allocation under an assumption that the given task can be split into arbitrary size subtasks. Agrawal and Jagadish [1] present one such model that can be used for an optimal partitioning of the class of computations that are organizable as a one-level tree, and are homogeneous and separable. The system model consists of a master processor and several similar slave processors. The task is assumed to be divisible into an initial phase, a separable phase and a final phase. The initial and final phases are executed on the master alone. The separable phase can be split into several independent subtasks executing in parallel on slave processors. The separable phase subtasks communicate with the initial phase subtask and the final phase subtask. Given a number of processors n, the goal is to minimize the total execution time (including communications overhead) by partitioning the separable phase into n independent arbitrary size subtasks. Point-to-point communication over the network is assumed. A nonlinear

programming formulation is given, which in special cases becomes linear. An iterative technique is also outlined to determine the optimal number of slave processors. One of the major drawbacks with this work is the assumption that continuous partitioning of the separable phase is possible with no communication between the subtasks.

In [20, 21], Haddad also assumes that the overall given task can be split arbitrarily. The problem considered in this work is that of minimizing the execution completion time of a given task by partitioning into interacting subtasks and allocating to run on a heterogeneous system, and it is also formulated as a nonlinear programming problem. An exact solution to the problem is given using a new technique, and a theorem is presented stating the necessary and sufficient condition for minimum execution time. The modeling of the execution time is similar in precision to that in [19], in the sense that internal inter-subtask communication times are considered and that precedence relations are ignored.

A limitation of the research described in [1, 20, 21] is that it is not directly applicable to practical situations where partitioning can usually be done only at specific points.

2.1.1.4 Heuristic Approach Heuristic methods aim only to find a suboptimal assignment for a task and are usually based on some simplifying assumptions. They are useful when it is important to reduce the amount of computation. A heuristic approach is described in [12], which attempts to minimize IPC by using a clustering approach under a load-balancing constraint. The load-balancing constraint is achieved by balancing the time needed to execute subtasks assigned to processors within a given tolerance.

In [11], Chu et al. consider the precedence relationship (PR) among subtasks explicitly and study its effects on the performance. The research indicates the subtask-size ratio between two consecutive subtasks plays an important role in determining whether they should be colocated. A heuristic algorithm considering PR, execution time of subtasks, and IPC is given which attempts to minimize the bottleneck-processor utilization. The algorithm works by grouping the subtasks based on IPC, PR effects, and size of the group, and is shown to generate better task assignments than those not considering the PR effects. Only homogeneous systems with a given number of processors are considered. Constraints on task completion time are given a consideration. Houstis [26] also discusses similar issues and describes heuristic algorithms for task allocation to homogeneous bus connected systems. The objective is to minimize the total processing time of the task. The algorithms try to minimize IPC delays (computed by taking interconnection network characteristics into account) by a clustering approach while keeping PR in mind, under a load-balancing constraint which is achieved by constraining each processor's utilization not to exceed a prespecified upper bound. An iterative algorithm is also given to determine the optimal

number of processors. This is the first attempt at solving the allocation problem to determine the optimal number of processors. This research also considers data to memory assignment.

2.1.2 Mapping of Specialized Algorithms

There have been research efforts directed towards the mapping of specialized signal processing algorithms onto multiprocessors. For example, 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 [7]. Also, there have been many research efforts on static scheduling of DSP algorithms on already designed synchronous general purpose multiprocessors (e.g., [29]).

2.1.3 Multiprocessor Synthesis Work

As the reader would notice, all the research efforts described in Sections 2.1.1 and 2.1.2 essentially concentrate on the problem of efficient allocation of tasks to given systems with the goal of optimizing the performance. The motivation in these efforts is to efficiently utilize given systems. No effort is directed towards the synthesis of systems, and hence cost of the system as such is not a consideration. In synthesis, one would like to design cost-effective systems that would provide the desired performance for given tasks. One such effort has been described by Talukdar and Mehrotra [46, 47]. This work describes a procedure for high-level synthesis of special-purpose dedicated heterogeneous (only in terms of speed) multiprocessor systems. Precedence relations and cost of the system are given explicit consideration. 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.

2.1.4 Scheduling of Precedence Graphs

Research efforts have also been directed to study the problem of scheduling precedence graphs onto homogeneous systems. Fernandez and Bussell [15] propose a lower and an upper bound on the number of processors required to execute the 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 [36] describe heuristic algorithms, combining critical path ideas with branch-and-bound, for scheduling to minimize the execution time. However, both the efforts completely ignore the communication overhead. Al-Mouhamed [2] describes research which considers the communication overhead. He 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 the graph within this completion time. An approximate lower bound on the completion time is also estimated for a given number of processors. These bounds are estimated by defining the notions of "earliest possible starting time" and "largest possible delay without increasing the completion time" for each of the subtasks in the graph.

2.2 Datapath Synthesis Research

We are using a mathematical programming approach for the multiprocessor synthesis problem. Such an approach has been used for the data path synthesis problem. Hafer and Parker [24, 22] have used a mixed-integer linear programming (MILP) 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. Our approach is similar. The differences between their approach and our approach are discussed in Section 3.3. Some strategies for improving the computational performance of Hafer's MILP model are reported by Prakash [59]. Hwang et al. [27, 28] have described an integer linear programming model for the scheduling problem in data path synthesis under resource constraints and time constraints, and they present a heuristic technique called *Zone Scheduling* for solving large size problems.

Datapath synthesis research has been at the core of the ADAM system [32, 56]. In the ADAM system, given a data flow/control flow graph description and the desired constraints on the cost and performance, pipelined datapaths are automatically synthesized by Sehwa [54] and non-pipelined by MAHA [55]. Some lower bounds on the cost and performance of the datapaths are reported in [34, 33].

The CATHEDRAL-II system [64] synthesizes a multiprocessor architecture, however, the architecture is almost completely fixed. It uses a set of synchronous interprocessor communication protocols as opposed to the asynchronous protocols that we propose to use. Its design philosophy is that efficient design synthesis is possible when targeted towards one particular, well defined system architecture. So, it synthesizes customized multiprocessor architectures and each processor is optimized to perform one particular part of the algorithm. The data paths of the processors are tailored to the application by connecting a set of Execution Units (EXU's). The set of available EXU's is restricted to six. It is a rule-based system and the emphasis is on how to optimize each processor rather than on how to configure the overall system. Haroun and Elmasry [25] mention multiprocessor architecture synthesis for DSP applications, however, this paper primarily concentrates on the design of an individual processor within the system, not how to configure the overall system. Selection of CPU design styles was researched by Thomas[72] and implemented by Lawson[42].

2.3 Array-Processor Synthesis Research

There has been research effort directed towards synthesis of array-processor architectures. 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.

In general, for a given computational task, parallelism could be exploited at different levels. The first level of parallelism is offered by partitioning the task into smaller subtasks. The second level is found within each subtask. The regularity desirable for array-processing is usually found in exploiting the second level of parrallelism. At the first level (exploiting parallelism among major subtasks), regularity is less common. It is unusual for a task's major subtasks to be identical or similar. 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.

Kung and Leiserson [37, 39, 38] proposed a number of ad hoc special-purpose VLSI systolic array architectures for some important algorithms such as matrix-vector, matrix-matrix multiplications, LU decompositions, recurrence evaluations, and others. Kung et al. [41] describe a *Wavefront Array Processor* which is a programmable special-purpose multiprocessor array suited for recursive and local data-dependent algorithms. Such al-

gorithms exhibit a continuously advancing wave of data and computational activity, or a *computational wavefront.* This notion of computational wavefront is also discussed in [73]. In [40], Kung proposes a methodology for converting parallel recursive algorithms into synchronous systolic arrays or data-driven wavefront arrays, using the concept of computational wavefront. Johnsson and Cohen [35] start with an algorithm-representation and map it onto a VLSI array architecture by using expression manipulation and operator calculus. Moldovan and Fortes [50, 49, 51, 17, 16] describe a synthesis technique for mapping of cyclic loop algorithms into special-purpose systolic arrays. The technique works by modifying the algorithm using a transformation function which is selected to minimize processing time and interconnection complexity for VLSI arrays. The transformation function is selected to expose hidden parallelism in the algorithm, by using parallelism detection techniques based on algorithm data dependencies. Cappello and Steiglitz [8] transform the given algorithm into an abstract model and then apply geometric transforms to design systolic architectures. Miranker and Winkler [48] extend Fortes's approach to develop a more generalized theory and methodology for mapping a given algorithm into a systolic array. Li and Wah [43] describe a systematic methodology for the design of optimal pure planar systolic arrays for algorithms that are representable as linear recurrence processes. They formulate the design problem as a constrained optimization problem in terms of the systolic array parameters.

2.4 System-Level CAD Tools

There are some system-level CAD tools available. These tools assist the designer during the design process; they do not automate the process.

The configuration of existing components using predesigned interconnection strategies was the subject of the R1 expert system [45], a successful package used by DEC to configure the systems it markets. Sara is a well-known system-level tool package which supports the designer in making design decisions, but which makes no design decisions of its own[13, 14]. ADAS [9] is a commercial system-level package which supports the designer with representation and simulation tools. It is a methodology and supporting tools set for system design. It uses directed graph models for design construction and analysis. Essentially, the model used is a Petri net-like model. ADAS does not automate the design activity; it basically provides a CAD environment where the designer iterates through the design process. Thus, the designer constructs and simulates various designs using the ADAS tools until (s)he finds a satisfactory design. The most important tool provided by ADAS is the simulator which is essentially a Petri net simulator.

3 The Problem Approach

Our approach is inspired by the work of Chu [10], Talukdar [46], and Hafer [24]. We are using mathematical programming to produce a formal model for the problem. The model will be linearized and converted into a MILP (Mixed Integer-Linear Programming) formulation, and then a branch-and-bound program will be used to solve the MILP to synthesize an optimal architecture for the given application. Such a mathematical model allows us a deep understanding of the problem and allows us to verify our software more easily, even if expected 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. The approach is called SOS and is also described in [57, 58].

An example model is presented here, which is also described in [60, 61, 62]. The model 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}.

3.1 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.

3.1.1 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:

$$\sum_{d|p_d \in P_a} \sigma_{d,a} = 1 \tag{3.1.1}$$

- *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:

$$\gamma_{a1,a2} = 1 - \sum_{d|p_d \in P_{a1} \cap P_{a2}} \sigma_{d,a1}\sigma_{d,a2} \tag{3.1.2}$$

  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:

$$T_{IA}(i_{a,b}) = T_{CE}(i_{a,b}) \tag{3.1.3}$$

- *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 is 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:

$$T_{OA}(o_{a,c}) = T_{SS}(S_a) + f_A(o_{a,c})(T_{SE}(S_a) - T_{SS}(S_a)) \tag{3.1.4}$$

  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:

$$T_{IA}(i_{a,b}) \leq T_{SS}(S_a) + f_A(i_{a,b})(T_{SE}(S_a) - T_{SS}(S_a)) \tag{3.1.5}$$

- *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:

$$T_{SE}(S_a) = T_{SS}(S_a) + \sum_{d|p_d \in P_a} \sigma_{d,a} D_{pS}(p_d, S_a) \tag{3.1.6}$$

  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_{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 $T_{CS}(i_{a2,b2})$:

$$T_{CS}(i_{a2,b2}) \geq T_{OA}(o_{a1,c1}) \tag{3.1.7}$$

- *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:

$$T_{CE}(i_{a2,b2}) = T_{CS}(i_{a2,b2}) + \gamma_{a1,a2} D_{CR} V_{a1,a2} + (1 - \gamma_{a1,a2}) D_{CL} V_{a1,a2} \tag{3.1.8}$$

  In the above, 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.

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 [24]). 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:

$$\sigma_{d,a1}\sigma_{d,a2}L([T_{SS}(S_{a1}), T_{SE}(S_{a1})], [T_{SS}(S_{a2}), T_{SE}(S_{a2})]) = 0 \tag{3.1.9}$$

- *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:

$$\sigma_{d2,a1}\sigma_{d2,a2}\sigma_{d1,a3}\sigma_{d1,a4}L([T_{CS}(i_{a1,b1}), T_{CE}(i_{a1,b1})], [T_{CS}(i_{a2,b2}), T_{CE}(i_{a2,b2})]) = 0 \tag{3.1.10}$$

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.

3.1.2 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.

3.1.2.1 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):

$$T_F \geq T_{SE}(S_a) \tag{3.1.11}$$

3.1.2.2 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, 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:

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

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. 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 $p_d \in P_a$):

$$\beta_d \geq \sigma_{d,a} \tag{3.1.12}$$

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

$$\chi_{d1,d2} \geq \sigma_{d1,a1}\sigma_{d2,a2} \tag{3.1.13}$$

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.*

3.2 Synthesis Using the Model

3.2.1 Linearization of the Model

Several constraints comprising the formulation presented in Section 3.1 are non-linear relations. These relations are linearized and the model is converted into a MILP (Mixed Integer-Linear Programming) formulation.

Equation 3.1.2 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, Equation 3.1.2 can be replaced by the following set of linear relations:

$$\gamma_{a1,a2} = 1 - \sum_{d|p_d \in P_{a1} \cap P_{a2}} \delta_{d,a1,a2} \tag{3.2.14}$$

$$\delta_{d,a1,a2} \leq \sigma_{d,a1} \tag{3.2.15}$$

$$\delta_{d,a1,a2} \leq \sigma_{d,a2} \tag{3.2.16}$$

Now, let us consider linearization of Equation 3.1.9. 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, Equation 3.1.9 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:

$$T_{SS}(S_{a2}) \geq T_{SE}(S_{a1}) - (3 - \alpha_{a1,a2} - \sigma_{d,a1} - \sigma_{d,a2})T_M \quad (3.2.17)$$

$$T_{SS}(S_{a1}) \geq T_{SE}(S_{a2}) - (2 + \alpha_{a1,a2} - \sigma_{d,a1} - \sigma_{d,a2})T_M \quad (3.2.18)$$

Similarly, to linearize Equation 3.1.10, 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, Equation 3.1.10 can be rewritten as the following pair of linear relations:

$$\begin{aligned} T_{CS}(i_{a2,b2}) \geq\ & T_{CE}(i_{a1,b1}) - (5 - \phi_{a1,b1,a2,b2} \\ & -\sigma_{d2,a1} - \sigma_{d2,a2} - \sigma_{d1,a3} - \sigma_{d1,a4})T_M \quad (3.2.19) \\ T_{CS}(i_{a1,b1}) \geq\ & T_{CE}(i_{a2,b2}) - (4 + \phi_{a1,b1,a2,b2} \\ & -\sigma_{d2,a1} - \sigma_{d2,a2} - \sigma_{d1,a3} - \sigma_{d1,a4})T_M \quad (3.2.20) \end{aligned}$$

Finally, non-linear Equation 3.1.13 can be simply rewritten in the following linear form:

$$\chi_{d1,d2} \geq \sigma_{d1,a1} + \sigma_{d2,a2} - 1 \quad (3.2.21)$$

3.2.2 Size of the Linearized MILP Model

Let n be the number of nodes (subtasks) and m the number of edges (data transfers) in the task graph. Let $p = |P|$ be the total number of processors available for selection as part of the system. Then the following statements can be made about the worst case growth of the MILP model.

The number of real (timing) variables grows as $O(an + bm)$, where a and b are some constants. The number of binary variables grows as $O(cn^2 + dm^2 + ep^2)$, where c, d and e are some constants. The number of constraints grows as $O(fp^2m^2 + gpn^2)$, where f and g are some constants. Although these are the growth rates in the worst case, the actual numbers of variables and constraints are usually fewer (as we will see in the examples presented later).

3.2.3 Solution of the Model

The MILP model is solved using a branch-and-bound program, *Bozo*, developed by L. J. Hafer of Simon Fraser University [23]. Bozo implements a LP-based branch-and-bound algorithm and supports binary and real variables. It invokes a commercial linear programming package, XLP, developed by XMP Software, Inc. Some example models have been created and solved using Bozo to synthesize architectures. These results are reported in Section 4.

3.3 Comparison with Hafer's RT-Level Model

As we have mentioned earlier, Hafer and Parker [24, 22] have used a mathematical programming approach for register-transfer level synthesis. Hafer's RT-level model also involves expressing timing relationships that must be satisfied for a correct and complete design. At first sight, it may seem that our model is a replica of Hafer's model. However, it must be emphasized that such is not the case. There are some fundamental issues that have to be addressed differently at the two levels (RT-level vs. system-level). Some of the differences are listed here:

- Hafer's model does not take into account the delay associated with the interconnection of the hardware elements explicitly. Output of a hardware element (operator or register) is considered to be immediately available at the input of the hardware element using it. This approach was appropriate at the RT-level as the interconnection delay (wiring and multiplexer delays) is probably much smaller than the delays associated with the hardware elements. However, at the system level we can not ignore the interconnection (communication) delays. Output generated by a processor can not be assumed to be immediately available to another processor; it has to be communicated through the interconnection network and the delay is by no means negligible. So, communication delays have to be explicitly taken into account by the model. Our model makes an attempt in this direction.

- In Hafer's model, an operation can not start until all its inputs have become available. In our model, a subtask can begin its execution even with some (or no) inputs available. At the RT-level, the requirement that all inputs be available before an operation starts is probably acceptable; however, at the system level it is not acceptable as a subtask may not require a particular input until it reaches a certain point in its execution. Theoretically speaking, it is conceivable that even at RT-level there

		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	-

Table 1: Execution Time and Cost Table for Four-Node Graph

are operations which can start without requiring all inputs. However, the number of such operations and the performance loss seem to be low enough to ignore such operations in order to keep the model simple at the RT-level.

- In Hafer's model, all the outputs of an operation become available only after the operation is completed. In our model, some of the outputs of a subtask can become available even before the subtask is completed. Once again, discussion similar to the previous point applies here.

4 Experiments and Preliminary Results

The approach outlined 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 [46].

4.1 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.

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

Table 2: Architectures for Four-Node Graph

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 93 variables, 21 timing and 72 binary, and 174 constraints. The complete model is given in Appendix C. 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 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 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.

		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

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

- *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.

Some tradeoff studies were performed using this example, which are reported in Appendix A.

4.2 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.

4.2.1 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

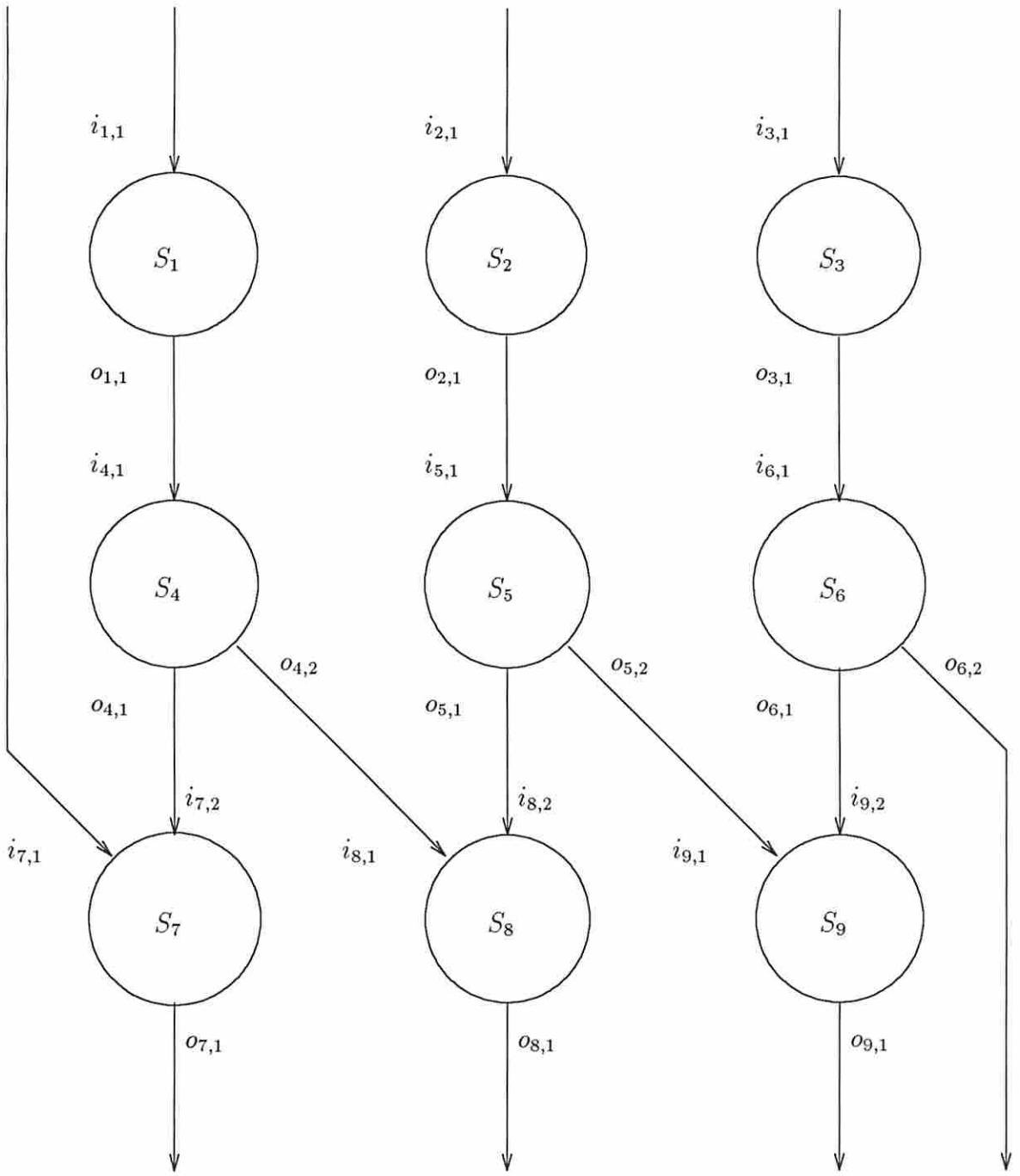


Figure 3: Nine-Node Task Graph

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

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 for the five designs are given in Table 4. Discussion of the designs is given in Appendix Section B.1.

4.2.2 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 model essentially remains similar to the model described in Section 3.1, except for a few differences. One difference is that there are no communication-link-creation variables in the bus-architecture model. Another difference is in the form of the communication-link-usage-exclusion constraint. Here, if any two data transfers are of remote type, then they both get transmitted over the same bus and exclusion in the usage of the bus must be ensured. So, Equation 3.1.10 gets replaced by the following:

$$\gamma_{a3,a1}\gamma_{a4,a2}L([T_{CS}(i_{a1,b1}), T_{CE}(i_{a1,b1})], [T_{CS}(i_{a2,b2}), T_{CE}(i_{a2,b2})]) = 0$$

For 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), the above relation ensures that the overlap in the usage of the bus by the two data transfers is prevented.

The MILP bus-architecture model for the nine-node graph example 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.

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

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

Table 5 gives the statistics for the three designs. Discussion of the designs is given in Appendix Section B.2.

4.3 Discussion of the Results and Future Work

Looking at the results described in Sections 4.1 and 4.2, it can be concluded that our approach does show promise for small examples. For the smaller example of four nodes, the runtime of the order of a few seconds should definitely be acceptable since an optimal design is ensured in return. Even for the nine-node example, the runtime of the order of a few hours is not extremely prohibitive for an optimal design. Depending upon the situation, the runtime of a few hours may indeed be acceptable if the optimality of the design is very crucial. Such may be the case because a human designer may actually require much longer design time if it is necessary to ensure the optimality of the design. The human designer may still not be sure about the optimality and the correctness of the design.

It is clear that the approach is usable for smaller examples depending upon the situation and the amount of runtime that the designer is willing to pay for. However, it is also quite clear that as the size of the designs grows, a point will be reached when the runtimes are certainly prohibitive. The only way to continue with this approach then would be to devise better solution strategies for the model. So, one direction of future research is aimed at improving the runtime.

The specific model discussed does not address several issues that may be of interest to the designer. For example, memory design in not included in the model. Similarly, the designer may be interested in other styles of interconnection. So, the other direction of future research is aimed at generating models which address these other aspects of design.

References

[1] R. Agrawal and H. Jagadish. Partitioning Techniques for Large-Grained Parallelism. *IEEE Transactions on Computers*, 37(12):1627–1634, December 1988.

[2] M. A. Al-Mouhamed. Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs. *IEEE Transactions on Software Engineering*, 16(12):1390–1401, December 1990.

[3] S. H. Bokhari. Dual Processor Scheduling with Dynamic Reassignment. *IEEE Transactions on Software Engineering*, SE-5(4):341–349, July 1979.

[4] S. H. Bokhari. A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System. *IEEE Transactions on Software Engineering*, SE-7(6):583–589, November 1981.

[5] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publishers, 1987.

[6] S. H. Bokhari. Partitioning Problems in Parallel, Pipelined, and Distributed Computing. *IEEE Transactions on Computers*, 37(1):48–57, January 1988.

[7] W. C. Brantley. *Automatically Decomposing Signal Processing Applications on Multiprocessors.* PhD thesis, Carnegie-Mellon University, 1979.

[8] P. Cappello and K. Steiglitz. Unifying VLSI Array Designs with Geometric Transformations. In *Proceedings 1983 International Conference on Parallel Processing*, pages 448–457. IEEE, 1983.

[9] Center for Digital Systems Research, Research Triangle Institute, Research Triangle Park, North Carolina 27709. *ADAS: An Architecture Design and Assessment System*, 1988. Training Manual.

[10] W. Chu, L. Hollaway, M. Lan, and K. Efe. Task Allocation in Distributed Data Processing. *Computer*, 13(11):57–69, November 1980.

[11] W. Chu and L. Lan. Task Allocation and Precedence Relations for Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(6):667–679, June 1987.

[12] K. Efe. Heuristic Models of Task Assignment Scheduling in Distributed Systems. *Computer*, 15(6):50–56, June 1982.

[13] G. Estrin. A methodology for design of digital systems - supported by SARA at the age of one. In *Proceedings of National Computer Conference*, volume 47, pages 313–324. NCC, 1978.

[14] G. Estrin, R. Fenchel, R. Razouk, and M. Vernon. SARA (System ARchitects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems. *IEEE Transactions on Software Engineering*, SE-12(2):293–311, February 1986.

[15] E. Fernandez and B. Bussell. Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules. *IEEE Transactions on Computers*, C-22(8):745–751, August 1973.

[16] J. Fortes and D. Moldovan. Parallelism Detection and Transformation Techniques Useful for VLSI Algorithms. *Journal of Parallel and Distributed Computing*, 2:277–301, May 1985.

[17] J. A. B. Fortes. *Algorithm Transformations for Parallel Processing and VLSI Architecture Design.* PhD thesis, Department of Electrical Engineering, University of Southern California, December 1983.

[18] D. Gusfield. Parametric Combinatorial Computing and a Problem of Program Module Distribution. *Journal of the Association for Computing Machinery*, 30(3):551–563, July 1983.

[19] E. K. Haddad. Analysis, Modeling and Optimization of Multiprocessing Execution Time. Technical Report TR 89-11, Department of Computer Science, Virginia Polytechnic Institute and State University, 1989.

[20] E. K. Haddad. Optimal Load Allocation for Parallel and Distributed Processing. Technical Report TR 89-12, Department of Computer Science, Virginia Polytechnic Institute and State University, April 1989.

[21] E. K. Haddad. Partitioned Load Allocation for Minimum Parallel Processing Time. In *Proceedings 1989 International Conference on Parallel Processing.* IEEE, August 1989.

[22] L. Hafer. *Automated Data-Memory Synthesis : A Formal Model for the Specification, Analysis and Design of Register-Transfer Level Digital Logic.* PhD thesis, Dept of Electrical Engineering, Carnegie Mellon University, Pittsburgh, Pa., May 1981.

[23] L. Hafer and E. Hutchings. Bringing up Bozo. Technical Report CMPT TR 90-2, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, March 1990.

[24] L. Hafer and A. Parker. A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic. *IEEE Transactions on Computer-Aided Design*, CAD-2(1), January 1983.

[25] B. S. Haroun and M. I. Elmasry. Architectural synthesis for DSP silicon compilers. *IEEE Trans. CAD.*, 8(4), April 1989.

[26] C. E. Houstis. Module Allocation of Real-Time Applications to Distributed Systems. *IEEE Transactions on Software Engineering*, 16(7):699–709, July 1990.

[27] C. Hwang, Y. Hsu, and Y. Lin. Optimum and Heuristic Data Path Scheduling Under Resource Constraints. In *Proceedings 27th Design Automation Conference*, pages 65–70. ACM/IEEE, June 1990.

[28] C. Hwang, J. Lee, and Y. Hsu. A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Transactions on Computer-Aided Design*, 10(4):464–475, April 1991.

[29] T. Barnwell III and D. Schwarz. Optimal implementation of flow graphs on synchronous multiprocessors. In *Int. Conf. on ASSP 84*, 1984.

[30] B. Indurkhya, H. S. Stone, and L. Xi-Cheng. Optimal Partitioning of Randomly Generated Distributed Programs. *IEEE Transactions on Software Engineering*, SE-12(3):483–495, March 1986.

[31] M. Iqbal, J. Saltz, and S. Bokhari. A Comparative Analysis of Static and Dynamic Load Balancing Strategies. In *Proceedings 1986 International Conference on Parallel Processing*, pages 1040–1047. IEEE, August 1986.

[32] R. Jain, K. Küçükçakar, M. Mlinar, and A. Parker. Experience with the ADAM Synthesis System. In *Proceedings 26th Design Automation Conference*, pages 56–61. ACM/IEEE, June 1989.

[33] R. Jain, M. Mlinar, and A. Parker. Area-Time Model for Synthesis of Non-Pipelined Designs. In *Proceedings International Conference on Computer-Aided Design*, pages 48–51. ACM/IEEE, November 1988.

[34] R. Jain, A. Parker, and N. Park. Predicting Area-Time Tradeoffs for Pipelined Design. In *Proceedings 24th Design Automation Conference*, pages 35–41. ACM/IEEE, July 1987.

[35] L. Johnsson and D. Cohen. A Mathematical Approach to Modelling the Flow of Data and Control in Computational Networks. In *Proceedings CMU Conf. VLSI Syst. Computations*, pages 213–225. Comput. Sci. Press, Rockville, Md., October 1981.

[36] H. Kasahara and S. Narita. Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing. *IEEE Transactions on Computers*, C-33(11):1023–1029, November 1984.

[37] H. T. Kung. Let's Design Algorithms for VLSI Systems. In *Proceedings Caltech Conf. VLSI*, pages 65–90, January 1979.

[38] H. T. Kung. Why Systolic Architectures. *Computer*, 15(1):37–46, January 1982.

[39] H. T. Kung and C. E. Leiserson. Systolic Arrays for VLSI. In *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980. Section 8.3.

[40] S. Y. Kung. On Supercomputing with Systolic/Wavefront Array Processors. *Proceedings of the IEEE*, 72(7):867–884, July 1984.

[41] S. Y. Kung, K. Arun, R. Gal-Ezer, and D. Rao. Wavefront Array Processor: Language, Architecture, and Applications. *IEEE Transactions on Computers*, C-31(11):1054–1066, November 1982.

[42] G. Lawson. Design style selector, an automated computer program implementation. Master's thesis, Dept. of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pa., August 1978.

[43] G. Li and B. Wah. The Design of Optimal Systolic Arrays. *IEEE Transactions on Computers*, C-34(1):66–77, January 1985.

[44] P. Ma, E. Lee, and M. Tsuchiya. A Task Allocation Model for Distributed Computing Systems. *IEEE Transactions on Computers*, C-31(1):41–47, January 1982.

[45] J. McDermott. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence*, 19(2), 1982.

[46] R. Mehrotra and S. Talukdar. Task Scheduling on Multiprocessors. Technical Report DRC-18-55-82, Department of Electrical Engineering, Carnegie-Mellon University, December 1982.

[47] R. Mehrotra and S. Talukdar. Scheduling of Tasks for Distributed Processors. Technical Report DRC-18-68-84, Department of Electrical Engineering, Carnegie-Mellon University, December 1984.

[48] W. Miranker and A. Winkler. Spacetime Representations of Computational Structures. *Computing*, 32(2):93–114, 1984.

[49] D. I. Moldovan. Computational Models for VLSI Systems. Technical Report DIM-82-3, Department of Electrical Engineering, University of Southern California, 1982.

[50] D. I. Moldovan. On the Analysis and Synthesis of VLSI Algorithms. *IEEE Transactions on Computers*, C-31(11):1121–1126, November 1982.

[51] D. I. Moldovan. On the Design of Algorithms for VLSI Systolic Arrays. *Proceedings of the IEEE*, 71(1):113–120, January 1983.

[52] D. M. Nicol. Optimal Partitioning of Random Programs Across Two Processors. *IEEE Transactions on Software Engineering*, 15(2), February 1989.

[53] N. J. Nilsson. *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[54] N. Park and A. Parker. Sehwa:A Program for Synthesis of Pipelines. In *Proceedings 23rd Design Automation Conference*, pages 454–460. ACM/IEEE, July 1986.

[55] A. Parker, J. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings 23rd Design Automation Conference*, pages 461–466. ACM/IEEE, July 1986.

[56] A. C. Parker, S. Hayati, R. Jain, K. Küçükçakar, M. Mlinar, S. Prakash, and J. Siedel. High-Level Synthesis in the ADAM System. In *2nd International Workshop of VLSI Design, Bangalore, India*, December 1988.

[57] A. C. Parker, K. Küçükçakar, S. Prakash, and J. Weng. Unified System Construction (USC). Technical Report CENG-91-01, Department of Electrical Engineering, University of Southern California, January 1991.

[58] A. C. Parker, K. Küçükçakar, S. Prakash, and J. Weng. Unified System Construction (USC). In *High-level VLSI Synthesis.* Kluwer Academic Publishers, 1991.

[59] S. Prakash. Guiding Design Decisions in RT-Level Logic Synthesis. Master's thesis, School of Computing Science, Simon Fraser University, Burnaby, B.C., April 1987.

[60] S. Prakash and A. C. Parker. Synthesis of Application-Specific Multiprocessor Architectures. Technical Report 90-25, Department of Electrical Engineering, University of Southern California, November 1990.

[61] S. Prakash and A. C. Parker. Synthesis of Application-Specific Multiprocessor Architectures. In *Proceedings 28th Design Automation Conference.* ACM/IEEE, June 1991.

[62] S. Prakash and A. C. Parker. Synthesis of Application-Specific Multiprocessor Architectures. In *Fifth International Workshop on High-Level Synthesis.* ACM, March 1991.

[63] C. Price and U. Pooch. Search Techniques for a Nonlinear Multiprocessor Scheduling Problem. *Naval Research Logistics Quarterly*, 29(2):213–233, June 1982.

[64] Jan Rabaey and Hugo De Man. Computer aided design of digital signal processing systems. In *Proc. ICCD*, 1987.

[65] G. S. Rao, H. S. Stone, and T. C. Hu. Assignment of Tasks in a Distributed Processor System with Limited Memory. *IEEE Transactions on Computers*, C-28(4):291–299, April 1979.

[66] CAD for system design: Is it practical? *SIGDA Newsletter*, 19(2):40–49, September 1989.

[67] C. Shen and W. Tsai. A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion. *IEEE Transactions on Computers*, C-34(3):197–203, March 1985.

[68] H. S. Stone. Critical Load Factors in Two-processor Distributed Systems. *IEEE Transactions on Software Engineering*, SE-4:254–258, May 1978.

[69] H. S. Stone. *High Performance Computer Architecture.* Addison-Wesley, 1987.

[70] H. S. Stone and S. H. Bokhari. Control of Distributed Processes. *Computer*, 11:97–106, July 1978.

[71] H.S. Stone. Multiprocessor scheduling with aid of network flow algorithms. *IEEE Trans. Software Eng.*, SE-3:85–93, January 1977.

[72] D. Thomas. *The Design and Analysis of an Automated Design Style Selector.* PhD thesis, Dept. of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pa., April 1977.

[73] U. Weiser and A. Davis. A Wavefront Notation Tool for VLSI Array Design. In *Proceedings CMU Conf. VLSI Syst. Computations*, pages 226–234. Comput. Sci. Press, Rockville, Md., October 1981.

[74] XMP Software, Inc., Tucson, AZ 85732. *XML2 Modeling Language*, 1988. Tutorial.

A Some Tradeoff Studies Using Four-Node Graph

In this appendix, we describe some experiments that were performed to study 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.

A.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-node 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: 2-processor design and uniprocessor design.

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

A.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 1). 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 subtasks S_4, and processor p_{3a} performs subtasks 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 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 experiment 1 and 2 essentially indicate 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 same performance with lower costs. However, as the sizes of subtasks increase (and thus inter-subtask communication becomes relatively negligible), multiprocessing becomes useful and designs with more processors are synthesized as they can provide better performance.

B Discussion of the Designs for Nine-Node Graph

In this appendix, we describe the synthesized designs for the nine-node task graph.

B.1 Point-to-Point Interconnection

As we mentioned in Section 4.2.1, five designs were synthesized for this style.

- *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.

- *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.

B.2 Bus-Style Interconnection

As we mentioned in Section 4.2.2, three designs were synthesized for this style.

- *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 $i_{8,1}$ 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.

C The MILP Model for Four-Node Graph

In this appendix, we present the complete MILP model for the four-node task graph. The model is written in the *XML2 Modeling Language*, the syntax of which is described in [74]. Bozo accepts input in this format. *PARAMETERS* in the model list the constant parameter values; *VARIABLES* list the real variables (or the timing variables in our case); *VARIABLES BINARY* list the 0-1 variables; *EQUATIONS* list the constraints and the objective function (the equation SYSTIM is the objective function in this case).

C.1 The Model

```
$ FOUR-NODE-TASK-GRAPH MODEL

$  PARAMETERS

! Constant parameter values used in the model

   @TAV_11 = 100
   @TAV_21 = 100
   @TAV_42 = 100

   @TP_11  = 1
   @TP_12  = 3
   @TP_21  = 1
   @TP_22  = 1
   @TP_23  = 3
   @TP_32  = 2
   @TP_33  = 1
   @TP_41  = 3
   @TP_42  = 1

   @DCR    = 1
   @V_31   = 1
   @V_32   = 1
   @V_41   = 1
   @D_31   = 1
   @D_32   = 1
```

```
@D_41   = 1

@TMAX   = 15
@TMAX2  = 30
@TMAX3  = 45
@TMAX4  = 60
@TMAX5  = 75

@COST_1 = 40
@COST_2 = 50
@COST_3 = 20

@C_LINK = 10

@CMAX   = 220

@FI_11  = 0.25
@FI_21  = 0.25
@FI_31  = 0.25
@FI_32  = 0.50
@FI_41  = 0.25
@FI_42  = 0.50

@FI_111 = 0.75
@FI_211 = 0.75
@FI_311 = 0.75
@FI_321 = 0.50
@FI_411 = 0.75
@FI_421 = 0.50

@FO_11  = 0.50
@FO_12  = 0.75
@FO_21  = 0.50
@FO_22  = 0.75
@FO_31  = 0.75
@FO_41  = 0.75

@FO_111 = 0.50
@FO_121 = 0.25
@FO_211 = 0.50
```

```
   @FO_221 = 0.25
   @FO_311 = 0.25
   @FO_411 = 0.25

$  VARIABLES

! Real (timing) variables in the model

   TSS_1, TSS_2, TSS_3, TSS_4
   TSE_1, TSE_2, TSE_3, TSE_4
   TOA_11, TOA_12, TOA_21, TOA_22, TOA_31, TOA_41
   TCS_31, TCS_32, TCS_41
   TCE_31, TCE_32, TCE_41
   TSYS

$  VARIABLES BINARY

! 0-1 variables in the model

   SIG_1A1, SIG_2A1
   SIG_1A2, SIG_1B2, SIG_2A2, SIG_2B2, SIG_3A2
   SIG_2A3, SIG_2B3, SIG_2C3, SIG_3A3, SIG_3B3
   SIG_1A4, SIG_1B4, SIG_1C4, SIG_2A4, SIG_2B4, SIG_2C4, SIG_2D4

   GAM_13, GAM_23, GAM_14

   BET_1A, BET_1B, BET_1C, BET_2A, BET_2B, BET_2C, BET_2D, BET_3A, BET_3B

   CHI_1A1B, CHI_1A1C, CHI_1A2A, CHI_1A2B, CHI_1A2C, CHI_1A2D, CHI_1A3A,
   CHI_1A3B
   CHI_1B2A, CHI_1B2B, CHI_1B2C, CHI_1B3A, CHI_1B3B
   CHI_2A1A, CHI_2A1B, CHI_2A1C, CHI_2A2B, CHI_2A2C, CHI_2A2D, CHI_2A3A,
   CHI_2A3B
   CHI_2B2A, CHI_2B2C, CHI_2B3A, CHI_2B3B
   CHI_3A2A, CHI_3A2B, CHI_3A2C, CHI_3A3B

   DEL_2A13, DEL_1A14, DEL_2A14, DEL_2A23, DEL_2B23, DEL_3A23

   ALF_12, ALF_24, ALF_34
```

```
   PHI_3132, PHI_3141, PHI_3241

$  EQUATIONS

! Constraints in the model

   MAPTSK1  ..  SIG_1A1 + SIG_2A1  =E=  1
   MAPTSK2  ..  SIG_1A2 + SIG_1B2 + SIG_2A2 + SIG_2B2 + SIG_3A2  =E=  1
   MAPTSK3  ..  SIG_2A3 + SIG_2B3 + SIG_2C3 + SIG_3A3 + SIG_3B3  =E=  1
   MAPTSK4  ..  SIG_1A4 + SIG_1B4 + SIG_1C4
              + SIG_2A4 + SIG_2B4 + SIG_2C4 + SIG_2D4  =E=  1

   DTTYP_13  ..  GAM_13 + DEL_2A13  =E=  1
   DDL2A131 ..  DEL_2A13 - SIG_2A1  =L=  0
   DDL2A133 ..  DEL_2A13 - SIG_2A3  =L=  0

   DTTYP_14  ..  GAM_14 + DEL_1A14 + DEL_2A14  =E=  1
   DDL1A141 ..  DEL_1A14 - SIG_1A1  =L=  0
   DDL1A144 ..  DEL_1A14 - SIG_1A4  =L=  0
   DDL2A141 ..  DEL_2A14 - SIG_2A1  =L=  0
   DDL2A144 ..  DEL_2A14 - SIG_2A4  =L=  0

   DDTYP_23  ..  GAM_23 + DEL_2A23 + DEL_2B23 + DEL_3A23  =E=  1
   DDL2A232 ..  DEL_2A23 - SIG_2A2  =L=  0
   DDL2A233 ..  DEL_2A23 - SIG_2A3  =L=  0
   DDL2B232 ..  DEL_2B23 - SIG_2B2  =L=  0
   DDL2B233 ..  DEL_2B23 - SIG_2B3  =L=  0
   DDL3A232 ..  DEL_3A23 - SIG_3A2  =L=  0
   DDL3A233 ..  DEL_3A23 - SIG_3A3  =L=  0

   BEGTSK1  ..  @FI_111 * TSS_1 + @FI_11 * TSE_1  =G=  @TAV_11
   BEGTSK2  ..  @FI_211 * TSS_2 + @FI_21 * TSE_2  =G=  @TAV_21
   BEGTSK31 ..  @FI_311 * TSS_3 + @FI_31 * TSE_3 - TCE_31  =G=  0
   BEGTSK32 ..  @FI_321 * TSS_3 + @FI_32 * TSE_3 - TCE_32  =G=  0
   BEGTSK41 ..  @FI_411 * TSS_4 + @FI_41 * TSE_4 - TCE_41  =G=  0
   BEGTSK42 ..  @FI_421 * TSS_4 + @FI_42 * TSE_4  =G=  @TAV_42

   ENDTSK1  ..  TSE_1 - TSS_1 - @TP_11 * SIG_1A1 - @TP_12 * SIG_2A1  =E=  0
   ENDTSK2  ..  TSE_2 - TSS_2 - @TP_21 * SIG_1A2 - @TP_21 * SIG_1B2
              - @TP_22 * SIG_2A2 - @TP_22 * SIG_2B2 - @TP_23 * SIG_3A2
```

```
             =E=  0
ENDTSK3  ..  TSE_3 - TSS_3 - @TP_32 * SIG_2A3 - @TP_32 * SIG_2B3
            - @TP_32 * SIG_2C3 - @TP_33 * SIG_3A3 - @TP_33 * SIG_3B3  =E=  0
ENDTSK4  ..  TSE_4 - TSS_4 - @TP_41 * SIG_1A4 - @TP_41 * SIG_1B4
            - @TP_41 * SIG_1C4 - @TP_42 * SIG_2A4 - @TP_42 * SIG_2B4
            - @TP_42 * SIG_2C4 - @TP_42 * SIG_2D4  =E=  0

OUTPUT11 ..  TOA_11 - @FO_111 * TSS_1 - @FO_11 * TSE_1  =E=  0
OUTPUT12 ..  TOA_12 - @FO_121 * TSS_1 - @FO_12 * TSE_1  =E=  0
OUTPUT21 ..  TOA_21 - @FO_211 * TSS_2 - @FO_21 * TSE_2  =E=  0
OUTPUT22 ..  TOA_22 - @FO_221 * TSS_2 - @FO_22 * TSE_2  =E=  0
OUTPUT31 ..  TOA_31 - @FO_311 * TSS_3 - @FO_31 * TSE_3  =E=  0
OUTPUT41 ..  TOA_41 - @FO_411 * TSS_4 - @FO_41 * TSE_4  =E=  0

SYST11    ..  TSYS - TSE_1 + TSS_1  =G=  0
SYST12    ..  TSYS - TSE_1 + TSS_2  =G=  0
SYST13    ..  TSYS - TSE_1 + TSS_3  =G=  0
SYST14    ..  TSYS - TSE_1 + TSS_4  =G=  0
SYST21    ..  TSYS - TSE_2 + TSS_1  =G=  0
SYST22    ..  TSYS - TSE_2 + TSS_2  =G=  0
SYST23    ..  TSYS - TSE_2 + TSS_3  =G=  0
SYST24    ..  TSYS - TSE_2 + TSS_4  =G=  0
SYST31    ..  TSYS - TSE_3 + TSS_1  =G=  0
SYST32    ..  TSYS - TSE_3 + TSS_2  =G=  0
SYST33    ..  TSYS - TSE_3 + TSS_3  =G=  0
SYST34    ..  TSYS - TSE_3 + TSS_4  =G=  0
SYST41    ..  TSYS - TSE_4 + TSS_1  =G=  0
SYST42    ..  TSYS - TSE_4 + TSS_2  =G=  0
SYST43    ..  TSYS - TSE_4 + TSS_3  =G=  0
SYST44    ..  TSYS - TSE_4 + TSS_4  =G=  0

BEGCOM31 ..  TCS_31 - TOA_11  =G=  0
BEGCOM32 ..  TCS_32 - TOA_21  =G=  0
BEGCOM41 ..  TCS_41 - TOA_12  =G=  0

ENDCOM31 ..  TCE_31 - TCS_31 - @D_31 * GAM_13  =E=  0
ENDCOM32 ..  TCE_32 - TCS_32 - @D_32 * GAM_23  =E=  0
ENDCOM41 ..  TCE_41 - TCS_41 - @D_41 * GAM_14  =E=  0

PLP1A121 ..  TSS_2 - TSE_1 - @TMAX * ALF_12 - @TMAX * SIG_1A1
```

```
            - @TMAX * SIG_1A2  =G=  - @TMAX3
PLP1A120 ..  TSS_1 - TSE_2 + @TMAX * ALF_12 - @TMAX * SIG_1A1
            - @TMAX * SIG_1A2  =G=  - @TMAX2
PLP2A121 ..  TSS_2 - TSE_1 - @TMAX * ALF_12 - @TMAX * SIG_2A1
            - @TMAX * SIG_2A2  =G=  - @TMAX3
PLP2A120 ..  TSS_1 - TSE_2 + @TMAX * ALF_12 - @TMAX * SIG_2A1
            - @TMAX * SIG_2A2  =G=  - @TMAX2
PLP2A13  ..  TSS_3 - TSE_1 - @TMAX * SIG_2A1 - @TMAX * SIG_2A3
            =G=  - @TMAX2
PLP1A14  ..  TSS_4 - TSE_1 - @TMAX * SIG_1A1 - @TMAX * SIG_1A4
            =G=  - @TMAX2
PLP2A14  ..  TSS_4 - TSE_1 - @TMAX * SIG_2A1 - @TMAX * SIG_2A4
            =G=  - @TMAX2
PLP2A23  ..  TSS_3 - TSE_2 - @TMAX * SIG_2A2 - @TMAX * SIG_2A3
            =G=  - @TMAX2
PLP2B23  ..  TSS_3 - TSE_2 - @TMAX * SIG_2B2 - @TMAX * SIG_2B3
            =G=  - @TMAX2
PLP3A23  ..  TSS_3 - TSE_2 - @TMAX * SIG_3A2 - @TMAX * SIG_3A3
            =G=  - @TMAX2
PLP1A241 ..  TSS_4 - TSE_2 - @TMAX * ALF_24 - @TMAX * SIG_1A2
            - @TMAX * SIG_1A4  =G=  - @TMAX3
PLP1A240 ..  TSS_2 - TSE_4 + @TMAX * ALF_24 - @TMAX * SIG_1A2
            - @TMAX * SIG_1A4  =G=  - @TMAX2
PLP1B241 ..  TSS_4 - TSE_2 - @TMAX * ALF_24 - @TMAX * SIG_1B2
            - @TMAX * SIG_1B4  =G=  - @TMAX3
PLP1B240 ..  TSS_2 - TSE_4 + @TMAX * ALF_24 - @TMAX * SIG_1B2
            - @TMAX * SIG_1B4  =G=  - @TMAX2
PLP2A241 ..  TSS_4 - TSE_2 - @TMAX * ALF_24 - @TMAX * SIG_2A2
            - @TMAX * SIG_2A4  =G=  - @TMAX3
PLP2A240 ..  TSS_2 - TSE_4 + @TMAX * ALF_24 - @TMAX * SIG_2A2
            - @TMAX * SIG_2A4  =G=  - @TMAX2
PLP2B241 ..  TSS_4 - TSE_2 - @TMAX * ALF_24 - @TMAX * SIG_2B2
            - @TMAX * SIG_2B4  =G=  - @TMAX3
PLP2B240 ..  TSS_2 - TSE_4 + @TMAX * ALF_24 - @TMAX * SIG_2B2
            - @TMAX * SIG_2B4  =G=  - @TMAX2
PLP2A341 ..  TSS_4 - TSE_3 - @TMAX * ALF_34 - @TMAX * SIG_2A3
            - @TMAX * SIG_2A4  =G=  - @TMAX3
PLP2A340 ..  TSS_3 - TSE_4 + @TMAX * ALF_34 - @TMAX * SIG_2A3
            - @TMAX * SIG_2A4  =G=  - @TMAX2
PLP2B341 ..  TSS_4 - TSE_3 - @TMAX * ALF_34 - @TMAX * SIG_2B3
```

```
            - @TMAX * SIG_2B4  =G=  - @TMAX3
PLP2B340 ..  TSS_3 - TSE_4 + @TMAX * ALF_34 - @TMAX * SIG_2B3
            - @TMAX * SIG_2B4  =G=  - @TMAX2
PLP2C341 ..  TSS_4 - TSE_3 - @TMAX * ALF_34 - @TMAX * SIG_2C3
            - @TMAX * SIG_2C4  =G=  - @TMAX3
PLP2C340 ..  TSS_3 - TSE_4 + @TMAX * ALF_34 - @TMAX * SIG_2C3
            - @TMAX * SIG_2C4  =G=  - @TMAX2

CLP31321 ..  TCS_32 - TCE_31 - @TMAX * PHI_3132 - @TMAX * SIG_1A1
            - @TMAX * SIG_1A2  =G=  - @TMAX3
CLP31322 ..  TCS_31 - TCE_32 + @TMAX * PHI_3132 - @TMAX * SIG_1A1
            - @TMAX * SIG_1A2  =G=  - @TMAX2
CLP31323 ..  TCS_32 - TCE_31 - @TMAX * PHI_3132 - @TMAX * SIG_2A1
            - @TMAX * SIG_2A2  =G=  - @TMAX3
CLP31324 ..  TCS_31 - TCE_32 + @TMAX * PHI_3132 - @TMAX * SIG_2A1
            - @TMAX * SIG_2A2  =G=  - @TMAX2
CLP31411 ..  TCS_41 - TCE_31 - @TMAX * PHI_3141 - @TMAX * SIG_2A3
            - @TMAX * SIG_2A4  =G=  - @TMAX3
CLP31412 ..  TCS_31 - TCE_41 + @TMAX * PHI_3141 - @TMAX * SIG_2A3
            - @TMAX * SIG_2A4  =G=  - @TMAX2
CLP31413 ..  TCS_41 - TCE_31 - @TMAX * PHI_3141 - @TMAX * SIG_2B3
            - @TMAX * SIG_2B4  =G=  - @TMAX3
CLP31414 ..  TCS_31 - TCE_41 + @TMAX * PHI_3141 - @TMAX * SIG_2B3
            - @TMAX * SIG_2B4  =G=  - @TMAX2
CLP31415 ..  TCS_41 - TCE_31 - @TMAX * PHI_3141 - @TMAX * SIG_2C3
            - @TMAX * SIG_2C4  =G=  - @TMAX3
CLP31416 ..  TCS_31 - TCE_41 + @TMAX * PHI_3141 - @TMAX * SIG_2C3
            - @TMAX * SIG_2C4  =G=  - @TMAX2
CLP32411 ..  TCS_41 - TCE_32 - @TMAX * PHI_3241 - @TMAX * SIG_1A1
            - @TMAX * SIG_2A4 - @TMAX * SIG_1A2 - @TMAX * SIG_2A3
              =G=  - @TMAX5
CLP32412 ..  TCS_32 - TCE_41 + @TMAX * PHI_3241 - @TMAX * SIG_1A1
            - @TMAX * SIG_2A4 - @TMAX * SIG_1A2 - @TMAX * SIG_2A3
              =G=  - @TMAX4
CLP32413 ..  TCS_41 - TCE_32 - @TMAX * PHI_3241 - @TMAX * SIG_1A1
            - @TMAX * SIG_2B4 - @TMAX * SIG_1A2 - @TMAX * SIG_2B3
              =G=  - @TMAX5
CLP32414 ..  TCS_32 - TCE_41 + @TMAX * PHI_3241 - @TMAX * SIG_1A1
            - @TMAX * SIG_2B4 - @TMAX * SIG_1A2 - @TMAX * SIG_2B3
              =G=  - @TMAX4
```

```
CLP32415 ..  TCS_41 - TCE_32 - @TMAX * PHI_3241 - @TMAX * SIG_1A1
           - @TMAX * SIG_2C4 - @TMAX * SIG_1A2 - @TMAX * SIG_2C3
             =G=  - @TMAX5
CLP32416 ..  TCS_32 - TCE_41 + @TMAX * PHI_3241 - @TMAX * SIG_1A1
           - @TMAX * SIG_2C4 - @TMAX * SIG_1A2 - @TMAX * SIG_2C3
             =G=  - @TMAX4
CLP32417 ..  TCS_41 - TCE_32 - @TMAX * PHI_3241 - @TMAX * SIG_2A1
           - @TMAX * SIG_2B4 - @TMAX * SIG_2A2 - @TMAX * SIG_2B3
             =G=  - @TMAX5
CLP32418 ..  TCS_32 - TCE_41 + @TMAX * PHI_3241 - @TMAX * SIG_2A1
           - @TMAX * SIG_2B4 - @TMAX * SIG_2A2 - @TMAX * SIG_2B3
             =G=  - @TMAX4
CLP32419 ..  TCS_41 - TCE_32 - @TMAX * PHI_3241 - @TMAX * SIG_2A1
           - @TMAX * SIG_2C4 - @TMAX * SIG_2A2 - @TMAX * SIG_2C3
             =G=  - @TMAX5
CLP3241A ..  TCS_32 - TCE_41 + @TMAX * PHI_3241 - @TMAX * SIG_2A1
           - @TMAX * SIG_2C4 - @TMAX * SIG_2A2 - @TMAX * SIG_2C3
             =G=  - @TMAX4

MAP_1A1   ..  BET_1A - SIG_1A1  =G=  0
MAP_1A2   ..  BET_1A - SIG_1A2  =G=  0
MAP_1A4   ..  BET_1A - SIG_1A4  =G=  0
MAP_1B2   ..  BET_1B - SIG_1B2  =G=  0
MAP_1B4   ..  BET_1B - SIG_1B4  =G=  0
MAP_1C4   ..  BET_1C - SIG_1C4  =E=  0
MAP_2A1   ..  BET_2A - SIG_2A1  =G=  0
MAP_2A2   ..  BET_2A - SIG_2A2  =G=  0
MAP_2A3   ..  BET_2A - SIG_2A3  =G=  0
MAP_2A4   ..  BET_2A - SIG_2A4  =G=  0
MAP_2B2   ..  BET_2B - SIG_2B2  =G=  0
MAP_2B3   ..  BET_2B - SIG_2B3  =G=  0
MAP_2B4   ..  BET_2B - SIG_2B4  =G=  0
MAP_2C3   ..  BET_2C - SIG_2C3  =G=  0
MAP_2C4   ..  BET_2C - SIG_2C4  =G=  0
MAP_2D4   ..  BET_2D - SIG_2D4  =E=  0
MAP_3A2   ..  BET_3A - SIG_3A2  =G=  0
MAP_3A3   ..  BET_3A - SIG_3A3  =G=  0
MAP_3B3   ..  BET_3B - SIG_3B3  =E=  0

LN1A1B14 ..  CHI_1A1B - SIG_1A1 - SIG_1B4  =G=  - 1
```

```
LN1A1C14 ..  CHI_1A1C - SIG_1A1 - SIG_1C4  =G=  - 1
LN1A2A13 ..  CHI_1A2A - SIG_1A1 - SIG_2A3  =G=  - 1
LN1A2A14 ..  CHI_1A2A - SIG_1A1 - SIG_2A4  =G=  - 1
LN1A2A23 ..  CHI_1A2A - SIG_1A2 - SIG_2A3  =G=  - 1
LN1A2B13 ..  CHI_1A2B - SIG_1A1 - SIG_2B3  =G=  - 1
LN1A2B14 ..  CHI_1A2B - SIG_1A1 - SIG_2B4  =G=  - 1
LN1A2B23 ..  CHI_1A2B - SIG_1A2 - SIG_2B3  =G=  - 1
LN1A2C13 ..  CHI_1A2C - SIG_1A1 - SIG_2C3  =G=  - 1
LN1A2C14 ..  CHI_1A2C - SIG_1A1 - SIG_2C4  =G=  - 1
LN1A2C23 ..  CHI_1A2C - SIG_1A2 - SIG_2C3  =G=  - 1
LN1A2D14 ..  CHI_1A2D - SIG_1A1 - SIG_2D4  =G=  - 1
LN1A3A13 ..  CHI_1A3A - SIG_1A1 - SIG_3A3  =G=  - 1
LN1A3A23 ..  CHI_1A3A - SIG_1A2 - SIG_3A3  =G=  - 1
LN1A3B13 ..  CHI_1A3B - SIG_1A1 - SIG_3B3  =G=  - 1
LN1A3B23 ..  CHI_1A3B - SIG_1A2 - SIG_3B3  =G=  - 1
LN1B2A23 ..  CHI_1B2A - SIG_1B2 - SIG_2A3  =G=  - 1
LN1B2B23 ..  CHI_1B2B - SIG_1B2 - SIG_2B3  =G=  - 1
LN1B2C23 ..  CHI_1B2C - SIG_1B2 - SIG_2C3  =G=  - 1
LN1B3A23 ..  CHI_1B3A - SIG_1B2 - SIG_3A3  =G=  - 1
LN1B3B23 ..  CHI_1B3B - SIG_1B2 - SIG_3B3  =G=  - 1
LN2A1A14 ..  CHI_2A1A - SIG_2A1 - SIG_1A4  =G=  - 1
LN2A1B14 ..  CHI_2A1B - SIG_2A1 - SIG_1B4  =G=  - 1
LN2A1C14 ..  CHI_2A1C - SIG_2A1 - SIG_1C4  =G=  - 1
LN2A2B13 ..  CHI_2A2B - SIG_2A1 - SIG_2B3  =G=  - 1
LN2A2B14 ..  CHI_2A2B - SIG_2A1 - SIG_2B4  =G=  - 1
LN2A2B23 ..  CHI_2A2B - SIG_2A2 - SIG_2B3  =G=  - 1
LN2A2C13 ..  CHI_2A2C - SIG_2A1 - SIG_2C3  =G=  - 1
LN2A2C14 ..  CHI_2A2C - SIG_2A1 - SIG_2C4  =G=  - 1
LN2A2C23 ..  CHI_2A2C - SIG_2A2 - SIG_2C3  =G=  - 1
LN2A2D14 ..  CHI_2A2D - SIG_2A1 - SIG_2D4  =G=  - 1
LN2A3A13 ..  CHI_2A3A - SIG_2A1 - SIG_3A3  =G=  - 1
LN2A3A23 ..  CHI_2A3A - SIG_2A2 - SIG_3A3  =G=  - 1
LN2A3B13 ..  CHI_2A3B - SIG_2A1 - SIG_3B3  =G=  - 1
LN2A3B23 ..  CHI_2A3B - SIG_2A2 - SIG_3B3  =G=  - 1
LN2B2A23 ..  CHI_2B2A - SIG_2B2 - SIG_2A3  =G=  - 1
LN2B2C23 ..  CHI_2B2C - SIG_2B2 - SIG_2C3  =G=  - 1
LN2B3A23 ..  CHI_2B3A - SIG_2B2 - SIG_3A3  =G=  - 1
LN2B3B23 ..  CHI_2B3B - SIG_2B2 - SIG_3B3  =G=  - 1
LN3A2A23 ..  CHI_3A2A - SIG_3A2 - SIG_2A3  =G=  - 1
LN3A2B23 ..  CHI_3A2B - SIG_3A2 - SIG_2B3  =G=  - 1
```

```
   LN3A2C23 ..  CHI_3A2C - SIG_3A2 - SIG_2C3  =G=  - 1
   LN3A3B23 ..  CHI_3A3B - SIG_3A2 - SIG_3B3  =G=  - 1

   MAP1A1B   ..  BET_1A - BET_1B  =G=  0
   MAP1A1C   ..  BET_1A - BET_1C  =G=  0
   MAP1B1C   ..  BET_1B - BET_1C  =G=  0
   MAP2A2B   ..  BET_2A - BET_2B  =G=  0
   MAP2A2C   ..  BET_2A - BET_2C  =G=  0
   MAP2A2D   ..  BET_2A - BET_2D  =G=  0
   MAP2B2C   ..  BET_2B - BET_2C  =G=  0
   MAP2B2D   ..  BET_2B - BET_2D  =G=  0
   MAP2C2D   ..  BET_2C - BET_2D  =G=  0
   MAP3A3B   ..  BET_3A - BET_3B  =G=  0

   SYSCOST   ..  @COST_1 * BET_1A + @COST_1 * BET_1B + @COST_1 * BET_1C
              + @COST_2 * BET_2A + @COST_2 * BET_2B + @COST_2 * BET_2C
              + @COST_2 * BET_2D + @COST_3 * BET_3A + @COST_3 * BET_3B
              + @C_LINK * CHI_1A1B + @C_LINK * CHI_1A1C + @C_LINK * CHI_1A2A
              + @C_LINK * CHI_1A2B + @C_LINK * CHI_1A2C + @C_LINK * CHI_1A2D
              + @C_LINK * CHI_1A3A + @C_LINK * CHI_1A3B
              + @C_LINK * CHI_1B2A + @C_LINK * CHI_1B2B + @C_LINK * CHI_1B2C
              + @C_LINK * CHI_1B3A + @C_LINK * CHI_1B3B
              + @C_LINK * CHI_2A1A + @C_LINK * CHI_2A1B + @C_LINK * CHI_2A1C
              + @C_LINK * CHI_2A2B + @C_LINK * CHI_2A2C + @C_LINK * CHI_2A2D
              + @C_LINK * CHI_2A3A + @C_LINK * CHI_2A3B
              + @C_LINK * CHI_2B2A + @C_LINK * CHI_2B2C + @C_LINK * CHI_2B3A
              + @C_LINK * CHI_2B3B
              + @C_LINK * CHI_3A2A + @C_LINK * CHI_3A2B + @C_LINK * CHI_3A2C
              + @C_LINK * CHI_3A3B  =N=

! Objective function for the model

   SYSTIM    ..  1000 TSYS
              + GAM_13 + GAM_23 + GAM_14
              + @COST_1 * BET_1A + @COST_1 * BET_1B + @COST_1 * BET_1C
              + @COST_2 * BET_2A + @COST_2 * BET_2B + @COST_2 * BET_2C
              + @COST_2 * BET_2D + @COST_3 * BET_3A + @COST_3 * BET_3B
              + @C_LINK * CHI_1A1B + @C_LINK * CHI_1A1C + @C_LINK * CHI_1A2A
              + @C_LINK * CHI_1A2B + @C_LINK * CHI_1A2C + @C_LINK * CHI_1A2D
              + @C_LINK * CHI_1A3A + @C_LINK * CHI_1A3B
```

```
            + @C_LINK * CHI_1B2A + @C_LINK * CHI_1B2B + @C_LINK * CHI_1B2C
            + @C_LINK * CHI_1B3A + @C_LINK * CHI_1B3B
            + @C_LINK * CHI_2A1A + @C_LINK * CHI_2A1B + @C_LINK * CHI_2A1C
            + @C_LINK * CHI_2A2B + @C_LINK * CHI_2A2C + @C_LINK * CHI_2A2D
            + @C_LINK * CHI_2A3A + @C_LINK * CHI_2A3B
            + @C_LINK * CHI_2B2A + @C_LINK * CHI_2B2C + @C_LINK * CHI_2B3A
            + @C_LINK * CHI_2B3B
            + @C_LINK * CHI_3A2A + @C_LINK * CHI_3A2B + @C_LINK * CHI_3A2C
            + @C_LINK * CHI_3A3B  =N=

$  BOUNDS

$  END
```