Assigning Optimal Configurations to Task Precedence Graphs

Hyeong-Ah Choi and Bhagirath Narahari

Department of Electrical Engineering & Computer Science
The George Washington University
Washington, DC
U.S.A. 20052

Abstract. Reconfigurable parallel computers provide a number of choices of algorithm or architecture configurations to execute a task. This paper introduces and discusses the problem of allocating configurations to nodes of a task precedence graph, where each node represents a subtask. Each subtask can be executed on one of a number of distinct configurations and one of these choices must be assigned to it. We are given the execution time on a configuration, and the reconfiguration time between any allocatable pair of configurations of related subtasks. The objective is to assign a configuration for each subtask such that the overall completion time of the entire task is minimized. This paper provides a graph theoretic formulation for this configuration assignment problem, and shows that it is NP-hard even if the maximum degree of the precedence graph is at most 3 and the number of choices for each subtask is at most 2. The problem is shown to be solvable when the maximum degree of the precedence graph is 2, thus closing the gap between the P and NP cases in terms of the degree of the graph. We then present efficient polynomial time algorithms, to find the optimal assignment, for two special cases of precedence graphs—(1) trees, and (2) series-parallel graphs.

1. Introduction

The advances in technology have led to the design of reconfigurable parallel computer architectures which include the class of partitionable parallel architectures [8], [10]. In a reconfigurable parallel computer, the configuration of the set of processors assigned to a task, which is to be executed using a parallel algorithm, can be varied to better match the requirements of the task. The configuration can vary in the type of architecture or in the type of algorithm [9] and each configuration can result in a different execution time for the task. An example of the first instance is when the number of processors or their interconnection topology is a variable. An example of the second instance is when we can change the data structures and data allocation schemes of the algorithm. Changing data structures or the data allocation actually results in a different algorithm, and thus results in the selection of a parallel algorithm for the problem. Since the configuration can vary in the architecture or the algorithm, we do not distinguish between the two. When the architecture has to be reconfigured, from its present configuration to the configuration required by the task, it incurs a cost (time) which is called the reconfiguration cost.

A task, to be executed on a reconfigurable parallel architecture, can consist of a number of constituent subtasks which must be executed to compute the final

JCMCC 13 (1993), pp. 77-96

result. Furthermore, the data dependencies among the subtasks enforce a precedence relation among the subtasks. Examples of such tasks are frequently found in computer vision [3], [9]. A set of precedence constrained tasks can be represented by a directed acyclic graph called a task precedence graph. An edge from the node corresponding to subtask T_i to the node corresponding to T_j denotes that T_j cannot start execution until T_i has completed and requirés the data/ouput from T_i as part of its input. Under this situtation, the machine must reconfigure from that (the configuration) assigned to T_i to the configuration assigned to T_j, and this reconfiguration cost/time adds to the total completion time of the entire task. The above framework, of reconfigurable architectures and task precedence graphs, gives rise to the design problem of assigning a configuration to each task [2], [9] to minimize the completion time of the task. The presence of reconfiguration costs implies that the problem of assigning (i.e., selecting) configurations is a non-trivial one, since without this cost we can simply assign the best configuration for each task.

This paper formulates and discusses the problem of allocating configurations to nodes of a task precedence graph of n nodes, where each node represents a subtask. Each subtask T_i, for $1 \leq i \leq n$, can be executed on one of q_i distinct configurations (q_i is the maximum number of allocatable configurations for this subtask) and must be assigned to one of these choices. As part of the input to the problem, we are given the execution time on a configuration and the reconfiguration time between any allocatable pair of configurations of related subtasks. The objective is to assign (i.e., select or allocate) a configuration for each subtask such that the overall completion time of the entire task is minimized. There are a number of methods of identifying bounds on the execution and reconfiguration times, for example see [2]. In this paper we use the terms allocation, assignment and selection interchangeably to mean the same.

We provide a graph theoretic formulation of the optimal configuration assignment problem. In our formulation, finding a feasible optimal assignment corresponds to finding a subgraph of a certain property with minimum weight. We show that the problem is NP-hard even if the number of choices for each subtask is at most 2 while the degree of each node of the precedence graph is limited to 3, i.e., each subtask is dependent on at most 3 other subtasks and $\max_i\{q_i\} \leq 2$. We develop efficient polynomial time algorithms for the special cases where the precedence graph is a tree and a series-parallel graph. Finally, we show that the problem is solvable in polynomial time when the degree of each node is less than 3 (for arbitrary q_i), thus closing the gap between the P and NP cases in terms of the degree of the precedence graph.

The paper is organized as follows. The following section formulates the problem and expresses it as a subgraph assignment problem. Section 3 discusses the complexity of the problem, and polynomial time algorithms for special cases are presented in Section 4. Section 5 provides concluding remarks and future endeavors.

The problem studied in this paper can also be applied to the design of parallel algorithms for a system of tasks. As observed in [9], there are many issues that need to be accounted for during the design of parallel algorithms including the number of configurations that the algorithm can execute on. Therefore our assignment problem can also be viewed as an abstraction of the problem of selecting efficient parallel algorithms for a precedence constrained task.

2. Problem Formulation

As an input to the configuration allocation problem, henceforth denoted CA, we are given a task precedence graph, choices of configurations for each subtask, execution time on each choice (i.e., on each configuration), and the reconfiguration time (referred to as RC time) between choices of dependent subtasks. Let $G_0 = (V, E)$ be the task precedence graph, where the set of vertices $V(G_0) = \{v_1, v_2, \ldots, v_n\}$ denotes the n subtasks and $E(G_0)$ is the set of directed edges denoting the precedence relations. If $(v_i, v_j) \in E(G_0)$ then subtask v_j cannot start until v_i has completed and in addition, the edge indicates that data must be transfered from v_i to v_j.

The set of choices of configurations, for each subtask, and the execution and reconfiguration times associated with the choices can be represented by a weighted communication graph G_1. Each subtask T_i, denoted by node v_i in the precedence graph, has q_i choices of configurations that can be allocated to it. The vertex set of G_1 is defined as

$$V(G_1) = \bigcup_{1 \le i \le n} V_i, \text{ where } V_i = \{v_i^1, v_i^2, \ldots, v_i^{q_i}\}.$$

The vertex subset V_i corresponds to subtask T_i where v_i^p represents the p-th choice (of configuration) for T_i. In other words, each node v_i in the precedence graph is replaced by q_i nodes, where each of the q_i nodes represents an assignment of a configuration for the subtask T_i.

The edge set of G_1 is defined such that for any two vertex sets $V_i \subset V(G_1)$ and $V_j \subset V(G_1)$, for $1 \le i, j \le n$,

> either $E(V_i, V_j) = \emptyset$ if $(v_i, v_j) \notin E(G_0)$ or $E(V_i, V_j) = \{(v_i^p, v_j^q) \mid 1 \le p \le q_i$ and $1 \le q \le q_j\}$ if $(v_i, v_j) \in E(G_0)$, where $E(V_i, V_j)$ represents the set of edges in G_1 whose tails are in V_i and whose heads are in V_j.

To each vertex $v_i^p \in V(G_1)$, there is assigned a positive number $w(v_i^p)$ representing the execution time on the p-th choice for subtask T_i. Similarly, to each edge $e = (v_i^p, v_j^q) \in E(G_1)$, there is assigned a positive number $w(e)$ representing the reconfiguration time, the q-th choice for T_j, i.e., the time to reconfigure from configuration p of subtask T_i to configuration q of subtask T_j. Figure 1 shows a precedence graph and a corresponding communication graph when q_i is 2 for all i.

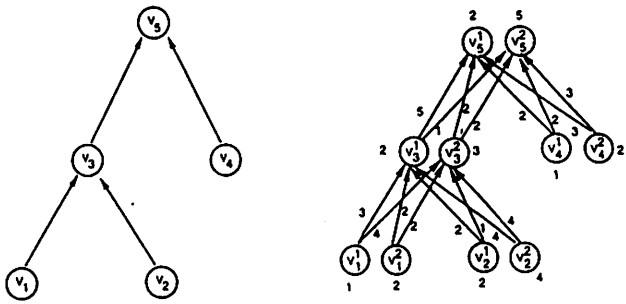


Figure 1
Precedence Graph G_0 and Communication Graph G

Let $\Delta(G_0)$ denote the maximum degree of G_0; $\Delta(G_0)$ gives the maximum number of related subtasks for any subtask in G_0 (i.e., it is a predecessor or successor of at most $\Delta(G_0)$ subtasks). Given a communication graph G_1, we define α to be $\max_{1 \leq i \leq n}\{q_i\}$. In practice, α is a small number bound by a constant. A subgraph $H(V, E)$ of $G(V, E)$ is called an *induced subgraph* of $G(V, E)$, if $V(H) \subseteq V(G)$ and $E(H) = \{(u, v) \in E(G) \mid u, v \in V(H)\}$ [6].

A feasible assignment, of configurations to subtasks, is one in which one configuration (among the q_i choices) is selected for each subtask T_i, $1 \leq i \leq n$, i.e., from each node subset V_i, of $V(G_1)$, select one node v_i^p. The set of selected nodes and the edges connecting these nodes would form an induced subgraph of G_1 that is isomorphic to the precedence graph G_0. We call such a graph an *allocation graph*; note that an assignment is given by $V(H)$ the nodes in H. Therefore an assignment can be simply stated as: from the communication graph G_1 select an alocation graph H. Figure 2 gives an allocation graph for the communication graph in the example in Figure 1.

The vertices of the allocation graph H determine the set of configurations that must be selected, i.e., assigned, to give us minimum completion time on the parallel architecture. In the remainder of this paper we refer to the allocation graph as an assignment although an assignment is in fact the vertices of the allocation graph. In our problem we are only focusing on determining what choices to select, i.e., determine H, so as to minimize objective function, and thus the problem can be viewed as an assignment problem as opposed to a scheduling problem.

For any allocation graph H, the vertex and edge weights dictate the resulting execution and reconfiguration times. We now define the completion time resulting from an allocation graph H. For any allocation graph $H \subseteq G_1$, let P_H be a directed path in H. The weight of the path P_H denoted by $W(P_H)$ is defined as:

$$W(P_H) = \sum_{v \in V(P_H)} w(v) + \sum_{e \in E(P_H)} w(e).$$

The completion time of the task is the weight of the "heaviest" path, i.e., the path with the largest weight, denoted by $W_{\max}(H)$, is

$$W_{\max}(H) = \max\{W(P_H) \mid P_H \text{ is a directed path in } H\}.$$

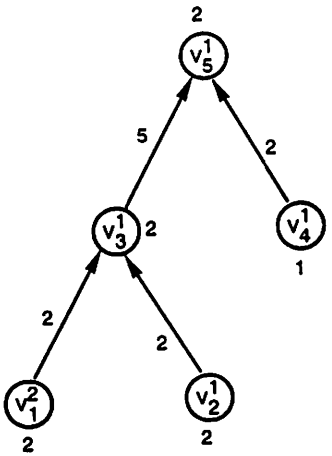


Figure 2
An Allocation Graph H

The configuration assignment problem with the objective of minimizing overall completion time of the task, denoted CA-MAX problem, as: "Find an allocation graph H of G_1 such that $W_{\max}(H)$ is as small as possible."

The variable $W_{\max}(H)$ denotes the completion time of the task resulting from an allocation corresponding to the vertices in H. If we define the objective to be the sum of the vertex and edge weights in H, then our problem with this objective (note that this objective does not give the completion time and hence is not applicable to our problem) can be reduced to the well studied task assignment problem in distributed systems [1], [4].

3. Problem Complexity

We now prove that the problem is NP-hard even for a very restricted case where each subtask has at most two choices of configurations and at most three related subtasks, and the precedence graph is bipartite.

The proof of NP-hardness is based on the following decision problem whose NP-completeness result is well-known [7].

3SAT Problem

INSTANCE: Given a set U of variables and a collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.

QUESTION: Is there a truth assignment for U such that each clause in C has at least one true literal?

Theorem 1. *CA-MAX problem is NP-hard even if* $\alpha \leq 2, \Delta(G_0) \leq 3$ *and* G_0 *is bipartite.*

Proof: To prove the NP-hardness, we will show the NP-completeness of the decision version of the CA problem defined as follows:

INSTANCE: A precedence graph G_0 the communication graph G_1 and deadline time D.

QUESTION: Is there an allocation graph H such that $W_{\max}(H) \leq D$?

It is not difficult to see that the decision version of the problem belongs to NP since for any assignment we can determine the path with heaviest weight. We next show a polynomial time transformation from the 3SAT problem.

Let $U = \{u_1, u_2, \ldots, u_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ be an instance of the 3SAT problem. For each i, $1 \leq i \leq n$, let $A(i) = \{i_1, i_2, \ldots, i_{s_i}\}$ be the set of indices such that $i_t \in A(i)$ if and only if either literal u_i or $\bar{u}_i$ appears in clause c_{i_t}, $1 \leq i_1 < i_2 < \cdots < i_{s_i} \leq m$. Using this instance, we construct a precedence graph G_0 as follows. The vertex set of G_0 is

$$V(G_0) = \{c_j, d_j \mid 1 \leq j \leq m\} \bigcup_{1 \leq i \leq n} \{x^i_{i_t} \mid i_t \in A(i)\} \bigcup_{1 \leq i \leq n} B_i.$$

To define B_i, we construct a rooted binary tree T_i corresponding to variable u_i, for $1 \leq i \leq n$, in the following way. The vertex set of T_i is $V(T_i) = \{x^i_{i_1}, x^i_{i_2}, \ldots, x^i_{i_{s_i}}\} \cup B_i$, where $\{x^i_{i_1}, x^i_{i_2}, \ldots, x^i_{i_{s_i}}\}$ represents the set of leaves all located at the same level of the tree, and B_i denotes all nonleaf nodes. The edge set of T_i is denoted by A_i.

Note that that each clause c_j has three literals and thus there are three vertices $x^f_j, x^g_j, x^h_j \in V(G_0)$ for some f, g, h such that $f < g < h$. Now, to define the edge set of G_0 construct a path connecting from c_j to d_j through x^f_j, x^g_j, x^h_j and let E_j denote the edges in this path, i.e., $E_j = \{(c_j, x^f_j), (x^f_j, x^g_j), (x^g_j, x^h_j), (x^h_j, d_j)\}$. The edge set of G_0 is then defined such that

$$E(G_0) = \bigcup_{1 \leq j \leq m} E_j \bigcup_{1 \leq i \leq n} A_i.$$

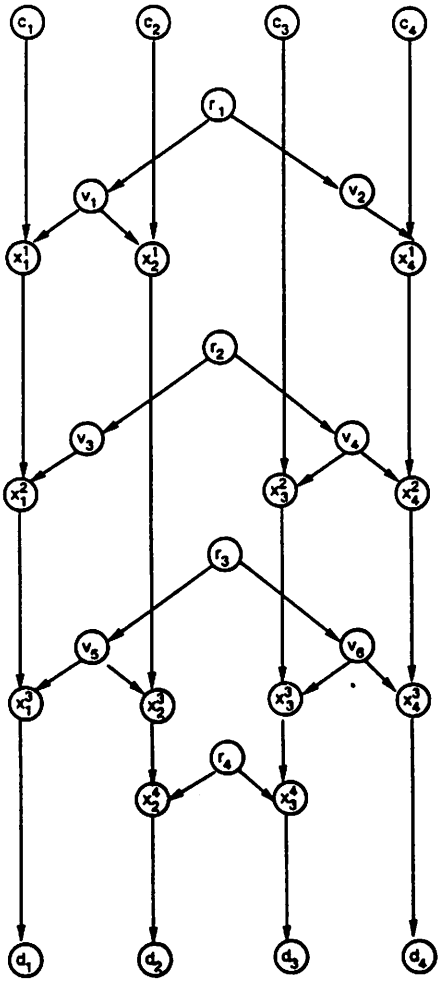


Figure 3
Precedence Graph G_0 constructed from
$C = \{(u_1, u_2, u_3), (\bar{u}_1, u_3, \bar{u}_4), (\bar{u}_2, \bar{u}_3, u_4), (u_1, u_2, \bar{u}_3)\}$

It is observed that the number of edges adjacent to each vertex in G_0 is at most three and G_0 is bipartite. Hence $\Delta(G_0) \leq 3$. Figure 3 shows an example of G_0 by following the above procedure, when $U = \{u_1, u_2, u_3, u_4\}$ and $C = \{\{u_1, u_2, u_3\}, \{\bar{u}_1, u_3, \bar{u}_4\}, \{\bar{u}_2, \bar{u}_3, u_4\}, \{u_1, u_2, \bar{u}_3\}\}$.

We next describe a procedure to construct a communication graph G_1 of G_0 such that each subtask has two choices of configurations. The vertex set of G_1 is defined such that each vertex v in $V(G_0) - \{c_j, d_j \mid 1 \leq j \leq m\}$ has

two corresponding vertices (represented as v and $\bar{v}$) in G_1 and each vertex in $\{c_j, d_j \mid 1 \leq i \leq m\}$ has one corresponding vertex (represented as c_j or d_j) in G_1. The edge set of G_1 is defined such that for each $(y, z) \in E(G_0)$, there are edges connecting from the vertices corresponding to $y \in V(G_0)$ to the vertices corresponding to $z \in V(G_0)$. The number of such edges is either 2 or 4 depending on whether, or not, y or z is in $\{c_j, d_j \mid 1 \leq j \leq m\}$. Formally, graph G_1 is defined as follows. The vertex set of G_1 is

$$V(G_1) = \{c_j, d_j \mid 1 \leq j \leq m\} \bigcup_{1 \leq i \leq n} \{x_{i_t}^i, \bar{x}_{i_t}^i \mid i_t \in A(i)\} \bigcup \{v, \bar{v} \mid v \in B_1 \cup \cdots \cup B_n\}.$$

The edge set of G_1 is

$$E(G_1) = \bigcup_{1 \leq j \leq m} D_j \bigcup_{1 \leq i \leq n} F_i,$$

where $D_j = \{(c_j, x_j^f), (c_j, \bar{x}_j^f), (x_j^f, x_j^g), (x_j^f, \bar{x}_j^g), (\bar{x}_j^f, x_j^g), (\bar{x}_j^f, \bar{x}_j^g), (x_j^g, x_j^h), (x_j^g, \bar{x}_j^h), (\bar{x}_j^g, x_j^h), (\bar{x}_j^g, \bar{x}_j^h), (x_j^h, d_j), (\bar{x}_j^h, d_j)\}$; and for each edge $(y, z) \in A_i (\subseteq E(G_0))$, there is a corresponding set F_i of four edges $(y, z), (y, \bar{z}), (\bar{y}, z), (\bar{y}, \bar{z})$, i.e., $F_i = \{(y, z), (y, \bar{z}), (\bar{y}, z), (\bar{y}, \bar{z}) \mid (y, z) \in A_i\}$.

To assign the weight of each vertex of G_1, set $w(v) = 1$, for all $v \in V(G_1)$. To assign the weight of each edge, let M_1 be any number larger than $2\lceil \log_2 m \rceil + 7$ and M_2 be any number larger than $2\lceil \log_2 m \rceil + 7 + 2M_1$. For each edge $e = (z, w) \in D_j$ where $w \in \{x_j^f, \bar{x}_j^f, x_j^g, \bar{x}_j^g, x_j^h, \bar{x}_j^h\}$, set $w(e) = 1$, if the literal corresponding to w (i.e., $u_f, \bar{u}_f, u_g, \bar{u}_g, u_h$ or $\bar{u}_h$) appears in clause c_j; and set $w(e) = M_1$, otherwise. For each edge directed to d_j, for $1 \leq j \leq m$, set $w(e) = 1$. Finally, for each edge $e \in F_i$ corresponding to $(y, z) \in A_i$, set $w(e) = 1$, if e is either (y, z) or $(\bar{y}, \bar{z})$; and set $w(e) = M_2$ if e is either in $(y, \bar{z})$ or $(\bar{y}, z)$. Figure 4 shows a communication graph G_1 corresponding to the precedence graph G_0 shown in Figure 3 by following the above procedure.

We next show that there exists a truth assignment for U such that each clause in C has at least one true literal if and only if there exists an allocation graph H of G_1 such that $W_{\max}(H)$ is less than $2\lceil \log_2 m \rceil + 7 + 2M_1$.

Suppose there exists a truth assignment for U. Using this truth assignment, an allocation graph $H \subseteq G_1$ is constructed in the following manner. The vertex set of H is

$$\begin{aligned} V(H) = &\{c_j, d_j \mid 1 \leq j \leq m\} \cup \{x_j^i \mid u_i \text{ is true}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m\} \\ &\cup \{\bar{x}_j^i \mid u_i \text{ is true}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m\} \\ &\cup \{v \in B_i \mid u_i \text{ is true}, 1 \leq i \leq n\} \cup \{\bar{v} \in B_i \mid u_i \text{ is true}, 1 \leq i \leq n\}. \end{aligned}$$

The edge set of H is

$$E(H) = \{(z, w) \in E(G_1) \mid z, w \in V(H)\}.$$

Figure 5 depicts an allocation graph H when $u_1 = T$, $u_2 = T$, $u_3 = F$, and $u_4 = F$.

To compute $W_{\max}(H)$ it is observed that (i) for any directed path P_1 in H from c_j to d_j, for $1 \leq j \leq m$, $\sum_{e \in E(P_1)} w(e) \leq 2M_1 + 7$ and (ii) for any directed path P_2 in H from the vertex corresponding to the root of T_i to d_j, for any i, j, $\sum_{e \in E(P_2)} w(e) \leq 2\lceil \log_2 m \rceil + 2M_1 + 5$. This follows from the observation that each binary tree T_i of G_0 has at most m leaves and thus the depth of T_i is at most $\lceil \log_2 m \rceil$. Therefore, we have $W_{\max}(H) \leq 2\lceil \log_2 m \rceil + 2M_1 + 7$.

Conversely, assume that there exists an allocation graph $H \subseteq G_1$ such that $W_{\max}(H) \leq 2\lceil \log_2 m \rceil + 2M_1 + 7$. Since $M_2 > \lceil \log_2 m \rceil + 2M_1 + 7$, for any edge $e \in E(H)$, $w(e) < M_2$. Thus, if $e \in F_i$ and $e \in E(H)$, e must be either (y, z) or $(\bar{y}, \bar{z})$. Further, for any directed path in H from c_j to d_j, there are at most two edges with weight M_1. The above discussion establishes the existence of a truth assignment for U. In particular, assigning true value to literal u_i (or $\bar{u}_i$) if $(y, z) \in F_i$ (or $(\bar{y}, \bar{z}) \in F_i$) is in $E(H_i)$. This completes the proof of the theorem. ∎

4. Polynomial Time Special Cases

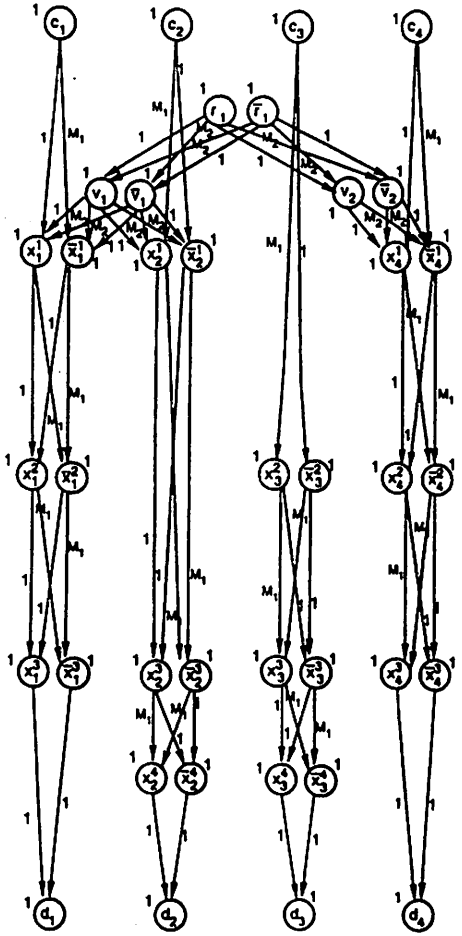
Although the CA-MAX problem is NP-hard, we show that some useful special cases can be solved in polynomial time. We first consider the case when the precedence graph is a tree. We then consider the case when the precedence graph is a series-parallel graph. Finally, we discuss the case when the maximum degree of the precedence graph is 2; this closes the gap between P and NP in terms of the degree of the graph.

4.1 Algorithm for Tree

Programs whose precedence graphs are tree-like form an important class; for example, divide and conquer algorithms exhibit a tree-like precedence graph as do semigroup computations. In addition, programs written as a hierearchy of subroutines also have a tree-like structure [1].

Theorem 2. *CA-MAX can be solved in* $O(|E(G_1)|) = O(\alpha^2 \mid V(G_0)|)$ *time if* G_0 *is a tree.*

Proof: Suppose the precedence graph G_0 is a tree, i.e., a precedence tree. Either out-trees (edges directed to child nodes) or in-trees (edges directed to parent node) are permissible. Without loss of generality, because path lengths are not affected by the arc direction, our discussion will consider in-trees. Therefore the root is the terminal node.



The vertices $r_2, \bar{r}_2, v_3, \bar{v}_3, v_4, \bar{v}_4, r_3, \bar{r}_3, v_5, \bar{v}_5, v_6, \bar{v}_6, r_4, \bar{r}_4$ and their adjacent edges are missing.

Figure 4

Communication graph G_1 (partially drawn) corresponding to G_0 in Figure 3

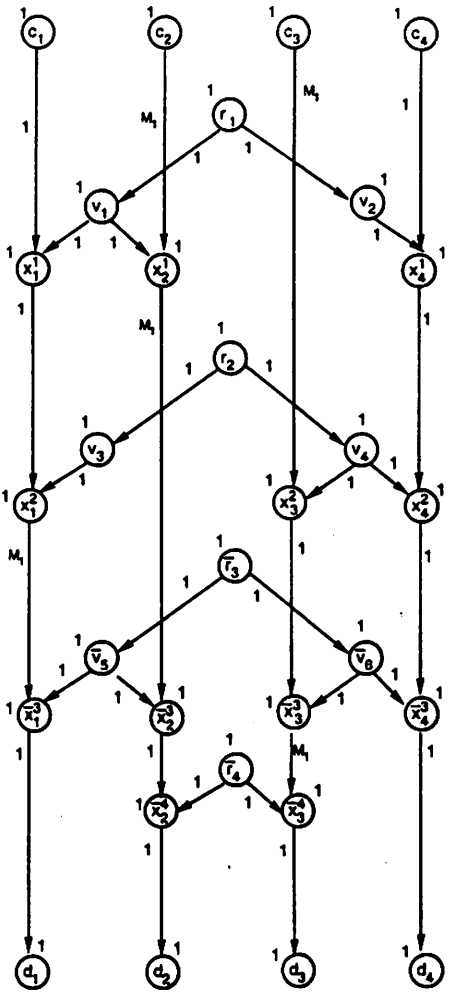


Figure 5

Allocation Graph H with $W_{\max}(H) = 2M_1 + 7 < \lceil \log_2 m \rceil + 2M_1 + 7$

The completion time of a node v_i in G_0 is dictated by the completion time of its children. The node v_i can be executed on any one of q_i configurations, and let $f(v_i^j)$ for each vertex v_i^j in G_1, denote the earliest time that subtask v_i can complete using its j-th choice of configuration. We assume that a list $L[f(v_i^j)]$ stores the assignments, of configurations to its predecessors, that result in the time $f(v_i^j)$. Note that the term earliest completion time is in fact the shortest/minimum time for v_i^j.

From the definition of $f(v_i^j)$, it follows that for any leaf vertex $v_i \in G_0$,

$f(v_i^j) = w(v_i^j)$ for all $1 \leq j \leq q_i$. Now consider any non-leaf vertex $v_i \in V(G_0)$, and let $B = \{v_{i_1}, v_{i_2}, \ldots, v_{i_b}\}$ denote the children of v_i (i.e., B is the set of predecessors of v_i in G_0). Note that vertex $v_i \in V(G_0)$ corresponds to subset $V_i \subseteq V(G_1)$ and vertex $v_{i_l} \in V(G_0)$ to subset $V_{i_l} \subseteq V(G_1)$ for all l, $1 \leq l \leq b$.

The earliest completion time $f(v_i^j)$ for each v_i^j, the j-th choice for v_i, is determined by the earliest completion times of its children nodes and the reconfiguration time, from the choices assigned to the children nodes to v_i^j. Consider an assignment of configurations $j_1, j_2, \ldots, j_b$, to the child nodes $v_{i_1}, v_{i_2}, \ldots, v_{i_b}$ respectively. For this assignment the earliest time for nodes in B is determined by the variable $f(v_{i_l}^{j_l})$ for $v_{i_l} \in B$ (with corresponding assignments stored in the lists $L[f(v_{i_l}^{j_l})]$). Thus, given this assignment of configurations to nodes in B the completion time of v_i^j (the j-th choice for the parent node) is the maximum among all path lengths from v_i^j to leaves of the tree. But these paths to the leaves must go through the nodes in $\{v_{i_l}^{j_l} \mid 1 \leq l \leq b\}$ (using nodes in the lists $L[f(v_{i_l}^{j_l})]$), and therefore the completion time of v_i^j for the assignment $(j_1, j_2, \ldots, j_b)$ can be expressed as:

$$w(v_i^j) + \max_{v_{i_l} \in B}\{f(v_{i_l}^{j_l}) + w(v_{i_l}^{j_l}, v_i^j)\}.$$

However, for each child node $v_{i_l} \in B$, there are q_{i_l}, choices of configurations that can be assigned. The earliest completion time $f(v_i^j)$ for v_i^j is that assignment which results in the minimum value. Firstly note that the set of paths from the child nodes to leaves are disjoint, thus we can select the best configuration for each node $v_{i_l} \in B$ independent of the configuration selected for other nodes in B. To determine the best choice for $v_{i_l} \in B$ we search all q_{i_l} possible choices, and select the assignment with the minimum value for completion time for v_i^j, and this is done for each $v_{i_l} B$. The value (and the assignment) of $f(v_i^j)$ is the maximum among these times. The earliest time for each child node v_{i_l} is determined by $f(v_{i_l}^{j_l})$. Therefore from the principle of optimality we have the following relation for $f(v_i^j)$, for $1 \leq j \leq q_i$.

$$f(v_i^j) = w(v_i^j) + \max_{v_{i_l} \in B}\left\{\min_{1 \leq j' \leq q_{i_l}}\{f(v_{i_l}^{j'}) + w(v_{i_l}^{j'}, v_i^j\}\right\}.$$

The above dynamic programming formulation can be directly implemented into an iterative procedure. This can be done by first computing $f(v_i^j)$ for all $v_i^j \in V(G_1)$, where the predecessors of $v_i \in V(G_0)$ are all leaves. We then iteratively compute the values of all the remaining vertices until the values of the vertices in $V(G_1)$ corresponding to the root (or terminal) vertex in $V(G_0)$ are computed. In each step of the above computation, if there exists a tie, we can arbitrarily select one among them. After we have completed the computation of

$f(v_i^j)$ for all $v_i^j \in V(G_1)$, the minimum completion time for the given input is equal to $\min\{f(v_r^j) \mid 1 \le j \le q_r\}$, where $v_r \in V(G_0)$ is the root of G_0. By storing the assignments (i.e., the values of $j_1, j_2, \ldots, j_b$ in the dynamic programming equation) along with the value of $f(v_i^j)$, at each step of the algorithm, we can recover the optimal assignment. The subgraph induced on the selected nodes that give the minimum value is an allocation graph. Figure 6 illustrates an application of the above described procedure to the example in Figure 1.

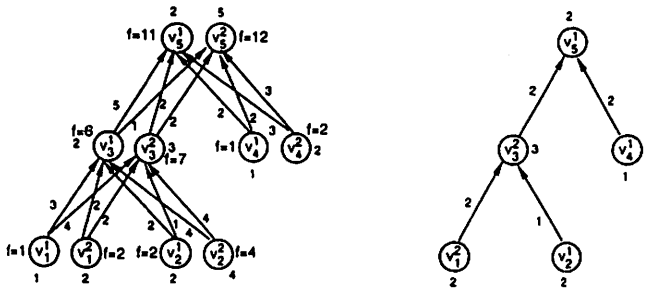


Figure 6
Computing $f(v_i^j)$ and the Optimum Allocation Graph H

The time complexity of our algorithm is seen to be $O(|E(G_1)|) = O(\alpha^2 n)$ since each edge $(v_i^p, v_j^q) \in E(G_1)$ is considered exactly once in the computation of $f(v_j^q)$ for all $v_j^q \in G_1$. ∎

4.2 Series-Parallel Graphs

We now consider a subclass of directed acyclic graphs, *directed series-parallel* graphs. Two edges of a directed graph are *series* if the head of one edge is the tail of the other one and are *parallel* if both edges have the same head and the same tail. A *directed series-parallel* graph is a directed acyclic multigraph which can be defined recursively as follows [5].

A directed single edge is a directed series-parallel graph. If G is a directed series-parallel graph, then a graph obtained from G by replacing an edge by series or parallel edges (i.e., a series or parallel composition) is a series-parallel graph. Figure 7 shows the construction of a series-parallel graph by a sequence of series and parallel compositions. Throughout the remaining section, directed series-parallel graphs are simply referred to as series-parallel graphs.

Series-parallel graphs form another important class of program dependency graphs; they can represent programs in which subtasks lie in loops or conditional branches (i.e., they model loops and conditional branches). Series-parallel graphs also include the class of *fork-join* graphs in which a process (i.e., subtask) creates a number of dependent processes which have common terminatation points,

i.e., they have a common join point in the graph. We now describe the following algorithm to solve CA-MAX problem for series-parallel graphs, and the ensuing theorem concludes the correctness of the algorithm.

Note that G_0 represents the dependency relations between the nodes and thus, we can assume without loss of any generality of the problem that for any two nodes u and v in G_0 there exists at most one edge from u to v. We also observe that since G_0 is a series-parallel graph, G_0 is a planar graph. It then follows that $|E(G_0)| = O(|V(G_0)|)$. We next describe our algorithm that finds an optimum solution by replacing the input by another one with smaller size but such that the minimum for the new input is the same as that of the original one.

Initially, set F_0 and F_1 to be G_0 and G_1, respectively. Construct a new precedence graph F_0 and its communication graph F_1' from F_0 and F_1 such that the number of edges in F_0 is one less than that in F_0 by executing either Step 1 or Step 2.

Step 1: If there exist two edges $e_1 = (v_i, v_l)$, $e_2 = (v_l, v_j) \in E(F_0)$, i.e., e_1 and e_2 are series edges, then delete e_1 and e_2 from F_0 and add a new edge $e = (v_i, v_j)$ to F_0. (Note that the vertex subsets in F_1 corresponding to v_i, v_l, v_j in F_0 are denoted by V_i, V_l, V_j, respectively.) Let the resulting graph be F_0. Construct a communication graph F_1' from F_1 by deleting the vertices in V_l and together with their adjacent edges (i.e., whose tails or heads are in V_l) and by adding the edges (denoted by $E(V_i, V_j)$) from each vertex in V_i to every vertex in V_j. Assign the weight of each edge $(v_i^a, v_j^b) \in E(V_i, V_j)$ such that

$$w(v_i^a, v_j^b) = \min_{1 \le c \le q_l} \{w(v_i^a, v_l^c) + w(v_l^c, v_j^b)\}.$$

Step2: If there exist two edges $e_1 = (v_i, v_j)$, $e_2 = (v_i, v_j) \in F_0$, i.e., e_1 and e_2 are parallel edges, then delete e_1 and e_2 from F_0 and add $e = (v_i, v_j)$ to F_0. Let F_0 be the resulting graph. Denote the set of edges in F_1 corresponding to e_1 (and e_2) by $E^1(V_i, Vj)$ (and $E^2(V_i, V_j)$). Construct a new communication graph F_1' from F_1 such that the edges in $E^1(V_i, V_j) \cup E^2(V_i, V_j)$ are deleted and the edges connecting from each vertex in V_i to every vertex in V_j are added. (This new edge set is denoted by $E(V_i, V_j)$.) Assign the weight of each edge $(v_i^a, v_j^b) \in E(V_i, V_j)$ such that

$$w(v_i^a, v_j^b) = \max\{w^1(v_i^a, v_j^b), w^2(v_i^a, v_j^b)\},$$

where $w^1(v_i^a, v_j^b)$ and $w^2(v_i^a, v_j^b)$, respectively, represent the weights of edges in $E^1(V_i, V_j)$ and $E^2(V_i, V_j)$ connecting from v_i^a to v_j^b.

If F_0 has more than one edge, we set F_0 to be F_0' and F_1 to be F_1', and repeat the above procedure. If F_0' consists of a single edge (v_i, v_j), then the optimum

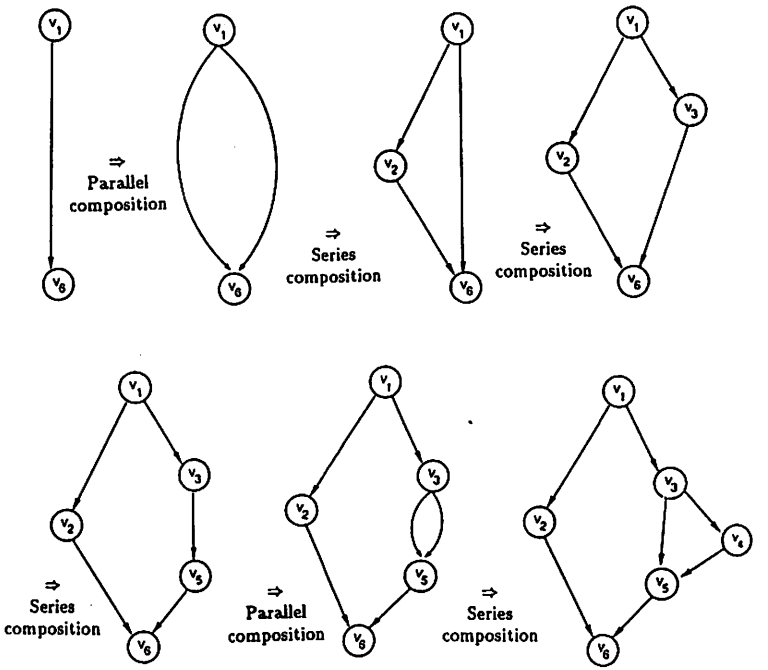


Figure 7
Series-Parallel Graphs

solution is

$$\min\{w(v_i^a) + w(v_i^a, v_j^b) + w(v_j^b) \mid 1 \leq a \leq q_i, 1 \leq b \leq q_j\}.$$

Figure 9 illustrates the application of the above procedure, and the resulting optimum allocation graph H, to the example (of precedence graph and corresponding communication graph) shown in Figure 8.

Theorem 3. *CA-MAX can be solved in $O(\alpha^3 n)$ time, if G_0 is a series-parallel graph.*

Proof: The proof is completed by showing the correctness of the two step procedure described above. To verify the correctness of the algorithm, we first note that after completing Step 1 or Step 2 at each iteration, the minimum solution of F_0

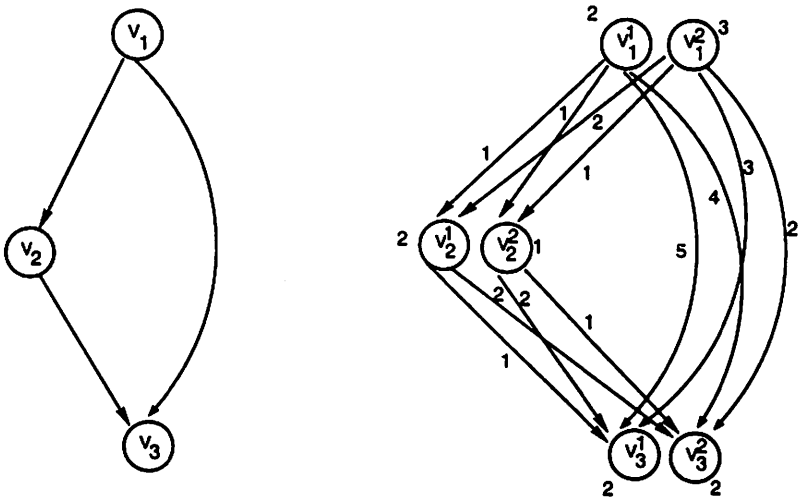


Figure 8
Examples of G_0 and G_1

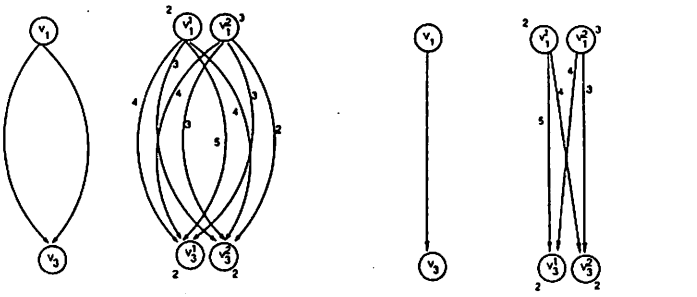
and F_1 is the same as that of F_0 and F_1. It is also observed that if F_0 has more than one edge, we can always find two edges that are either series edges or parallel edges. This follows from the fact that F_0 at each iteration is a series-parallel graph.

To analyze the time complexity of the algorithm, note that $|E(F_0')| = |E(F_0)|-1$ after executing Step 1 or Step 2 at each iteration. At each iteration, we need $O(\alpha^3)$ or $O(\alpha^2)$ time for doing Step 1 or Step 2 respectively. The number of iterations is bound by the number of edges in $E(G_0)$. Since $|E(G_0)| = O(|V(G_0)|)$, it follows that the time complexity of the algorithm is $O(\alpha^3 n)$, where $n = |V(G_0)|$. ∎

4.3 Graph of degree two

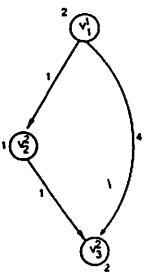
We finally show that an optimum allocation graph of G_1 can be found in polynomial time if each subtask has less than three dependent subtasks. This result, together with the NP-hardness result in Theorem 1, completely closes the gap between the polynomially solvable cases and the NP-hardness cases in terms of the maximum degree of the precedence graph. We first describe the algorithm and the theorem follows as a consequence.

Let G_0 and G_1 be a precedence graph and a communication graph. If the number of edges in G_0 incident at each vertex is no larger than two, the underlying graph of G_0 (i.e., the graph without considering the directions of the edges) is either a simple path or a cycle. As the former case can be solved using the algorithm



(a)
After applying Step 1 to F_0 and F_1 in Figure 8

(b)
After applying Step 2 to F_0 and F_1 in Part (a)



(c)
Optimum allocation graph H

Figure 9

for the latter case, we only consider the case when the underlying graph of G_0 is a cycle. The following steps will produce an optimum allocation graph H of G_1.

Step 1: Construct directed graphs F_0 and F_1 from G_0 and G_1 as follows. If there exist two edges $e_1 = (v_i, v_l), e_2 = (v_l, v_j) \in E(F_0)$, then delete e_1 and e_2 from F_0 and add a new edge $e = (v_i, v_j)$ to F_0; delete all the vertices in V_l together with their adjacent edges from F_1, and add edges to F_1 connecting from every vertex in V_i to every vertex in V_j, and assign the weight of each new edge $(v_i^a, v_j^b) \in E(F_1)$ such that $w(v_i^a, v_j^b) = \min_{1 \le c \le q_l}\{w(v_i^a, v_l^c) + w(v_l^c) + w(v_l^c, v_j^b)\}$. Repeat the above process unlil every vertex in F_0 has either two incoming edges or two outgoing edges.

Step 2: Find a cycle C_1 (by ignoring the directions of edges) from F_1 by selecting

one vertex from V_i, for each $V_i \subseteq V(F_1)$ such that $\max\{w(e) \mid e \in E(C_1)\}$ is the minimum.

Step 3: Construct an optimum allocation graph H by replacing each edge in C_1 by the corresponding path in G_1.

Figure 11 illustrates the application of the above three step procedure to the example shown in Figure 10. Figure 11(a) and (b) show F_0 and F_1 respectively. Figure 11(c) shows C_1 and (d) shows the optimum allocation graph H.

Theorem 4. *CA-MAX can be solved in $O(\alpha^3 n)$ time if each subtask has at most two dependent subtasks.*

Proof: To analyze the time complexity of the above steps, we first note that Step 1 can be done in $O(\alpha^3 n)$ time, since there are at most $O(n)$ iterations and each iteration takes $O(\alpha^3)$ time. To find a cycle C_1 in Step 2, select an arbitrary vertex subset V_i of $V(F_1)$. For each $v_i^p \in V_i$, find a cycle going through v_i^p while leaving the minimum value of $\max\{w(e) \mid e \in E(C_1)\}$. Such a cycle can be found in $O(\alpha^2 n)$ time by defining a function f similarly as described in Theorem 2. Since there are $O(\alpha)$ verices in V_i, Step 2 can be done in $O(\alpha^3 n)$ times. Finally, Step 3 can be done in $O(|E(G_1)|) = (\alpha^2 n)$ time, which gives the desired time complexity. To conclude the correctness, note that the optimality of the cycle selected in Step 2 can be concluded from the arguments used in Theorem 2. ∎

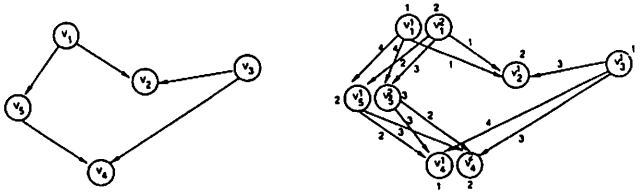
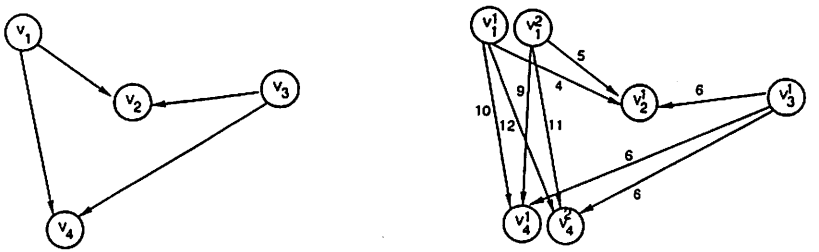


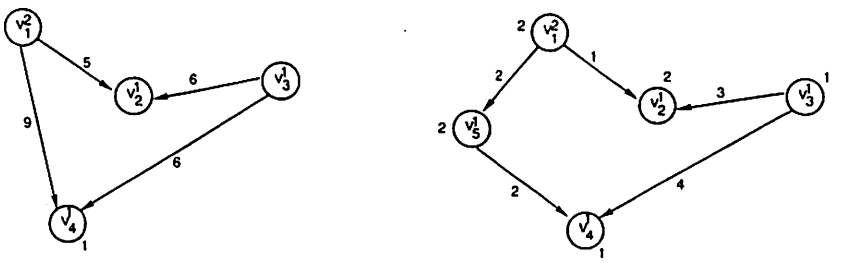
Figure 10
Degree 2 Precedence Graph G_0 and Communication Graph G_1

5. Conclusions

This paper introduced and formulated a graph assignment problem which has applications in reconfigurable parallel architectures. Specifically, we considered the assignment of machine configurations to nodes of a task precedence graph when we are given the time/cost of a node on each configuration. The problem was seen to be NP-hard even when the degree of the precedence graph was 3 and each subtask has at most 2 choices of configurations. This led to the investigation of special instances of precedence graphs such as trees and series-parallel graphs. These two classes of graphs model many common programs and thereby



(a)
After applying Step 1 to G_0 and G_1 in Figure 10



(b)
C_1 after Step 2

(c)
Optimum allocation graph H

Figure 11
Application of the Algorithm

constitute an important class of precedence graphs. We developed an $O(\alpha^2 n)$ algorithm for tree precedence graphs and an $O(\alpha^3 n)$ algorithm for series-parallel graphs. By showing that the CA-MAX problem is solvable for any precedence graph with degree at most 2, we closed the gap between the P and NP cases in terms of the degree of the graph. Since α is bound by a small constant in practice, our algorithms effectively run in linear time. Future endeavors include the investigation of approximation algorithms and the pursuance of additional useful special instances.

References

1. S.H. Bokhari, "Assignment Problems in Parallel and Distributed Computing", Kluwer Academic Publishers, 1987.
2. H.-A. Choi, B. Narahari, S. Rotenstreich, and A. Youssef, *Scheduling on Parallel Processing Systems Using Parallel Primitives*, in "Parallel Architec-

tures", ed. N. Rishe, S. Navathe, and D. Tal, IEEE Computer Society Press, Boston, MA, 1990.

3. A.N. Choudhary and J.H. Patel, "Parallel Architectures and Parallel Algorithms for Computer Vision", Kluwer Academic Publishers, Boston, MA, 1990.
4. W.W. Chu, L. J. Holloway, M. Lan, and K. Efe, *Task Allocation in Distributed Data Processing*, Compuler (1980), 57–69.
5. R. J. Duffin, *Topology of Series Parallel Networks*, Journal of Math. Applic. **10** (1965), 303–318.
6. S. Even, "Graph Algorithm", Comp. Science Press, 1979.
7. M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP Completeness", W.H. Freeman, San Francisco, 1979.
8. J.P. Hayes, T.N. Mudge, Q.F. Stout, and S. Colley, *Architecture of a Hypercube Supercomputer*, Proceedings of the 1986 International Conference on Parallel Processing.
9. L.H. Jamieson, *Characterizing Parallel Algorithms*, in "The Characteristics of Parallel Algorithms", ed. L.H. Jamieson, D.B. Gannon, and R.J. Douglas, MIT Press, 1987.
10. H.J. Siegel, L.J. Siegel, F.C. Kemmerer, P.T. Mueller, Jr.; H.E. Smalley, and S.D. Smith, *PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition*, IEEE Transactions on Computers **C-30** (1981).
11. D. Towsley, *Allocating programs containing branches and loops within a multiple processor system*, IEEE Tran. on Soft. Eng. **SE-12** (1986), 1018–1024.