# Optimal Task Assignment in Homogeneous Networks

Cheol-Hoon Lee and Kang G. Shin, *Fellow*, *IEEE*

**Abstract**—This paper considers the problem of assigning the tasks of a distributed application to the processors of a distributed system such that the sum of execution and communication costs is minimized. Previous work has shown this problem to be tractable for a system of two processors or a linear array of $N$ processors, and for distributed programs of serial parallel structures. Here we focus on the assignment problem on a homogeneous network, which is composed of $N$ functionally-identical processors, each with its own memory. Some processors in the network may have unique resources, such as data files or certain peripheral devices. Certain tasks may have to use these unique resources; they are called *attached tasks*. The tasks of a distributed program should therefore be assigned so as to make use of specific resources located at certain processors in the network while minimizing the amount of interprocessor communication. The assignment problem in such a homogeneous network is known to be NP-hard even for $N = 3$, thus making it intractable for a network with a medium to large number of processors. We therefore focus on task assignment in general array networks, such as linear arrays, meshes, hypercubes, and trees. We first develop a modeling technique that transforms the assignment problem in an array or tree into a minimum-cut maximum-flow problem. The assignment problem is then solved for a general array or tree network in polynomial time.

**Index Terms**—Array and tree networks, homogeneous networks, cutsets, maximum flows, network flow, task assignment.

## 1 Introduction

In a general-purpose distributed system, the tasks of a distributed application must be assigned to the processors in such a way that the system resources will be utilized efficiently and certain cost will be minimized. Unfortunately, this assignment problem is NP-complete for general $N$-processor systems. Hence, the problem of finding a minimum-cost assignment is computationally intractable for all but small systems. We will therefore restrict the assignment problem to homogeneous arrays or trees, popular multicomputer interconnection topologies, for which one can derive tractable solutions. Our main result is the development of an algorithm that solves the assignment problem for an $n$-dimensional homogeneous array network of $N$ ($= n_1 \times n_2 \times \cdots \times n_n$) processors or an $N$-processor tree network in polynomial time.

Many researchers studied the problem of assigning tasks of an application to the processors of a distributed system. Stone [16] suggested an efficient optimal algorithm for the problem of assigning tasks to two processors (*two-processor problem*) by making use of the well-known network flow algorithm in two-terminal network graphs. He showed how the network flow model can be extended to systems made up of three or more processors. For the three-processor case, Stone and Bokhari [18] developed an algorithm that finds an optimal assignment. This algorithm works in most cases, but there are pathological cases for which it fails to find an optimal assignment. Stone [17] also developed an efficient algorithm for the two-processor problem in which the load on one of the two processors is varied. Bokhari [2] analyzed the problem of dynamic assignment for two-processor systems and transformed it into a network flow problem under the assumption that all the system characteristics are known for each phase of a distributed application. If the distributed application structure is constrained in a certain way, one can find the optimal assignment in a system of any number of processors in polynomial time. When the structure is constrained to be a tree, the shortest-tree algorithm developed by Bokhari [3] yields an optimal assignment. Towsley [19] generalized Bokhari's results to the case of series-parallel structures.

All of the above work is well documented in [4] and implicitly assumes the computing system to be fully-connected, i.e., there exists a communication link between any two processors. If a given distributed application is a chain-structured parallel or pipelined program, it can be optimally partitioned over a chain or ring of processors subject to the constraint that each processor is assigned a contiguous subchain of program modules [5], [15]. In this case, the objective of assignment is to minimize the load of a bottleneck processor rather than to minimize the total load of the processors. This approach can also be used to find an optimal global assignment of a set of independent serial distributed programs over a single-host, multiple-satellite system [5]. However, the general $N$-processor problem ($N > 3$) in a fully-connected system is NP-complete [3], [14], and hence, several heuristic methods [1], [13], [14] have been proposed to solve the problem.

Recently, we proposed in [12] an optimal algorithm to solve the problem of assigning interacting tasks to a linear array network of an arbitrary number of processors. A linear

- *C.-H. Lee is with the Parallel Processing Laboratory, Department of Computer Engineering, Chungnam National University, Daejon, 305-764, Korea.*
- *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, MI 48109-2122. E-mail: kgshin@eecs.umich.edu.*

*Manuscript received Oct. 13, 1994.*
*For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95246.*

array network is composed of linearly-arranged processors, i.e., when any two nonadjacent processors are to communicate with each other, the intermediate processors between them must participate in the communication. Thus, the communication cost per unit information transferred between any two processors increases linearly with the distance between them. The task assignment problem in a linear array network is first transformed into a two-terminal network flow problem, and then solved by applying the network flow algorithm on the related two-terminal network graph. This is a direct extension of Stone's network flow approach for the two-processor problem to the case of a linear array of $N$ processors.

The network considered here is homogeneous in the sense that it is composed of multiple functionally-identical processors, each with its own memory, joined by a certain interconnection topology. Some processors in the network may have unique resources such as data files or certain expensive peripheral devices, and thus, certain tasks may have to be assigned to certain processors in order to take advantage of their unique capabilities [7], [13]. The tasks of a distributed application should therefore be assigned so as to make use of specific resources of some processors in the network while minimizing the amount of interprocessor communication. (If there is no task requiring unique resources, the problem becomes trivial, since the best solution will be to assign all the tasks to any one processor.) While this problem for a general homogeneous $N$-processor network is NP-hard even for $N = 3$ [10], [14], there are certain interconnection topologies for which it becomes tractable.

In this paper, we develop a modeling technique that transforms the assignment problem into a minimum-cut maximum-flow problem on an appropriately-defined network flow graph representation. The network flow graph can represent task communication costs in a way that is dependent both on the interconnection topology and on how the unique resources are distributed among the processors in the network. For an $n$-dimensional array network with $N$ ($= n_1 \times n_2 \times \cdots \times n_n$) processors or an $N$-processor tree network, we can successfully apply the modeling technique. Thus, for both the general array network such as linear arrays, meshes, hypercubes, $n$-dimensional arrays, and trees, we can compute minimum-cost assignments in polynomial time by using well-known, efficient network flow graph algorithms.

The paper is organized as follows. In Section 2, we present the system model and the assumptions used. The problem of assigning tasks to the processors in a homogeneous network is also formally stated there. In Section 3, we present a graph-theoretic modeling technique that transforms the assignment problem for a general array network into a minimum-cut maximum-flow problem based on an appropriately-defined network flow graph representation. We then propose an algorithm using the modeling technique that solves the assignment problem for the general array network in polynomial time. Section 4 shows that the modeling technique can also be applied to trees. The paper concludes with Section 5.

## 2 Problem Statement

A homogeneous network is composed of $N$ functionally-identical processors $p_1, p_2, \cdots, p_N$, each with its own memory, joined by an interconnection network. The *distance*, $d_{k,l}$, between two processors $p_k$ and $p_l$ is defined as the minimum number of links connecting them, e.g., the distance between any two adjacent processors is equal to 1. When two nonadjacent processors $p_k$ and $p_l$ ($d_{k,l} > 1$) are to communicate with each other, the intermediate processors between them must participate in the communication.

Even in a homogeneous network, it is desirable to avoid duplicating resources such as data files or certain expensive peripheral devices [7], [13]. We therefore assume that some processors have certain unique capabilities in the network. It is also assumed that each unique capability exists in one and only one processor in the network. Some tasks may have to be assigned to certain fixed processors in order to exploit their unique capabilities. Considering this fact, we classify tasks into two categories:

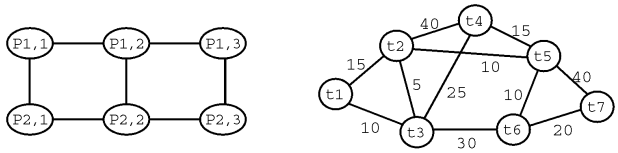
1) "attached tasks" that can only be assigned to certain processors;
2) "general tasks" that can be assigned to *any* processor in the network.

The notation $t_i \Rightarrow p_k$ is used to denote that task $t_i$ is "attached" to processor $p_k$. The processors may be multiprogrammed, may concurrently execute different programs, but may not concurrently execute the same program.

A distributed application $P$ to be executed on such a network of processors consists of $M$ interacting tasks $t_1, t_2, \ldots, t_M$. The execution cost of $t_i$ is assumed to be known *a priori* and is denoted as $E_i > 0$. The execution cost, $R_{i,k}$, of $t_i$ on processor $p_k$ is then equal to $E_i$ if $t_i$ is a general task. If $t_i$ cannot be executed by $p_k$, then its execution cost $R_{i,k}$ on $p_k$ is set to be infinite.

The interaction among the tasks in $P$ is represented by a *task interaction graph* (TIG), in which nodes correspond to the tasks in $P$ and there is an edge between two nodes if and only if the corresponding tasks interact. For each pair of tasks, $t_i$ and $t_j$, in the TIG, we define $W_{i,j} (= W_{j,i})$ as the communication volume, measured in number of packets, between $t_i$ and $t_j$ during their execution. Obviously, if there is no edge between $t_i$ and $t_j$ in TIG then $W_{i,j} = 0$. An example TIG of seven tasks, to be executed by a $2 \times 3$ array network (Fig. 1a), is shown in Fig. 1b with the execution costs shown in Fig. 1c, where $t_1 \Rightarrow p_{1,1}$, $t_3 \Rightarrow p_{2,2}$, and $t_7 \Rightarrow p_{1,3}$.

The communication cost in executing a set of tasks is defined as the sum of time units each communication link is used during the execution. In other words, the communication cost is a measure of the link resources used by an instance of execution expressed in time units. Suppose $c(\ell)$ is the number of time units needed to send a packet over a path of length / distance $\ell$, and the time a link is kept busy for purposes other than packet transmission—such as establishing a communication path—is assumed to be negligible. For packet-switched networks, it is obvious that we have $c(\ell) = \ell c(1)$. This relation may be less accurate in case of circuit switching. However, if the "call request" signal to hunt for a free path occupies each link only for a very short time, then this expression would be a good approximation for circuit-switched networks [20].




**(a)** A (2X3) two-dimensional array network. **(b)** A task interaction graph.

| | P1,1 | P1,2 | P1,3 | P2,1 | P2,2 | P2,3 |
|---|---|---|---|---|---|---|
| t1 | 25 | ∞ | ∞ | ∞ | ∞ | ∞ |
| t2 | 15 | 15 | 15 | 15 | 15 | 15 |
| t3 | ∞ | ∞ | ∞ | ∞ | 30 | ∞ |
| t4 | 20 | 20 | 20 | 20 | 20 | 20 |
| t5 | 25 | 25 | 25 | 25 | 25 | 25 |
| t6 | 40 | 40 | 40 | 40 | 40 | 40 |
| t7 | ∞ | ∞ | 30 | ∞ | ∞ | ∞ |

**(c)** The execution costs.

Fig. 1. A program graph on a 2 × 3 array network and the corresponding execution costs.

Without loss of generality, we define $c(1)$ as the *unit* of communication cost (i.e., the link usage by one packet traversing one link). Thus, if two interacting tasks $t_i$ and $t_j$ are assigned to two different processors $p_k$ and $p_l$, respectively, then the two tasks will incur the interprocessor communication cost, $W_{i,j} \cdot d_{k,l}$. We assume that the communication cost between two tasks assigned to the same processor is negligible, since all interprocess communication is done by reading from and/or writing to memory as opposed to message passing via (multiple) communication links. (The latter takes much longer than the former.)

We use an assignment function $X : t_i \rightarrow p_{X(i)}$ to represent an assignment of the $M$ tasks in TIG to the $N$ processors in the network. The cost of an assignment $X$ is the sum of the total execution and communication costs:

$$COST(X) = EXEC(X) + COMM(X)$$
$$= \sum_{i=1}^{M} R_{i,X(i)} + \sum_{i<j} W_{i,j} \cdot d_{X(i),X(j)}.$$

An assignment $X$ is said to be *feasible* if every task is executable under the assignment $X$, i.e., $t_i \Rightarrow p_{X(i)}$ for all attached tasks $t_i$s. For every feasible assignment $X$, the total execution cost incurred by the assignment is equal to $\sum_i E_i$ and is constant regardless of assignment. However, a different assignment will lead to a different communication cost. A feasible assignment for the example in Fig. 1 is shown in Fig. 2, where the total execution and communication costs of the assignment are 185 and 265, respectively. Note that the distance between two processors $p_{k_1,k_2}$ and $p_{l_1,l_2}$ in Fig. 2 is $|k_1 - l_1| + |k_2 - l_2|$. Therefore, the task assignment problem in homogeneous networks is the problem of finding a feasible assignment $X_o$ with the minimum communication cost, i.e., $COMM(X_o) = \min_X COMM(X)$.

The task assignment problem in homogeneous networks is intrinsically hard. Note that when the interconnection topology of an $N$ processor network is fully-connected and every processor in the network has exactly one attached task, the task assignment problem is equivalent to the following $N$-cut problem with specified vertices: given an undirected graph $G$, find a minimum weight $N$-cut which, when deleted, partitions the graph into exactly $N$ components each of which contains exactly one of the given $N$ vertices. The $N$-cut problem with specified vertices is NP-hard even for $N = 3$ [10], [14]. Thus, there is no known polynomial-time algorithm to find an optimal assignment in a general homogeneous network of $N \geq 3$ processors. However, the modeling technique to be presented in the next two sections has a polynomial worst-case bound on homogeneous array or tree networks.

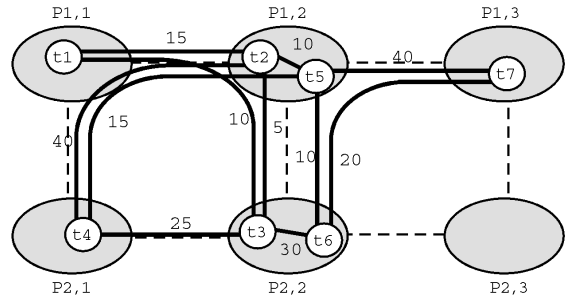



Fig. 2. A feasible assignment $X$ ($COMM(X) = 265$).

## 3 Assignment in Array Networks

In this section, we develop a modeling technique that transforms the assignment problem in a homogeneous array network of $N$ processors into a minimum-cut maximum-flow problem. The main idea of this technique is to construct a network-flow graph for the assignment problem in such a way that every feasible assignment is represented by a set of cutsets each on an appropriately-defined network-flow graph, and that the total weight of the cutsets is equal to the total communication cost of the assignment. Thus, an interpretation of the cutsets with the minimum total weight determines the assignment of tasks to processors with minimum total communication cost. In the next section, this modeling technique is applied to tree networks by slightly modifying the network-flow graph representation.

An $n$-dimensional homogeneous array is composed of $N$ $(= n_1 \times n_2 \times \cdots \times n_n)$ functionally-identical processors

$$\{p_{k_1,k_2,\cdots,k_n} \mid 1 \leq k_i \leq n_i, \text{ for all } i\}$$

with a communication link between each pair of processors $p_{k_1,k_2,\cdots,k_n}, p_{l_1,l_2,\cdots,l_n}$ if and only if $|k_j - l_j| = 1$ for some $j$th coordinate and $k_i = l_i$ for the other coordinates $1 \leq i(\neq j) \leq n$. The distance between any two processors $p_{k_1,k_2,\cdots,k_n}$ and $p_{l_1,l_2,\cdots,l_n}$ becomes $\sum_i |k_i - l_i|$.

### 3.1 The Cutset Formulation

Given the TIG $= (V, E)$ of a distributed application submitted to an $n_1 \times n_2 \times \cdots \times n_n$ array, we first generate a corresponding $N(= n_1 \times n_2 \times \cdots \times n_n)$-terminal network graph $G_N = (V_N, E_N)$.

The $N$-terminal network graph $G_N$ is obtained from the TIG by adding $N$ terminal nodes each of which corresponds to each processor, i.e., $V_N = V \cup \{p_{k_1,k_2,\cdots,k_n} \mid 1 \le k_i \le n_i$, for all $i\}$ and $E_N = E$. We denote by $\mathbf{P}_{i,j}$ the set of processor nodes whose $i$th coordinate is less than or equal to $j$, and by $\overline{\mathbf{P}}_{i,j}$ the set of the other processor nodes, i.e., $\mathbf{P}_{i,j} = \{p_{k_1,k_2,\cdots,k_n} \mid 1 \le k_i \le j\}$ and $\overline{\mathbf{P}}_{i,j} = \{p_{k_1,k_2,\cdots,k_n} \mid j < k_i \le n_i\}$. For a $(2 \times 3)$ two-dimensional array network, for example, $\mathbf{P}_{1,1} = \{p_{1,1}, p_{1,2}, p_{1,3}\}$, $\mathbf{P}_{2,1} = \{p_{1,1}, p_{2,1}\}$, $\mathbf{P}_{2,2} = \{p_{1,1}, p_{1,2}, p_{2,1}, p_{2,2}\}$. For each $i$th coordinate, we build $(n_i - 1)$ two-terminal network graphs $G_{i,j}$s, $1 \le j < n_i$, from the $N$-terminal network graph $G_N$ as follows.

1) Generate a node $S_{i,j}$ by combining all the processor nodes in $\mathbf{P}_{i,j}$ and all the task nodes which are attached to one of these processors, i.e.,

$$S_{i,j} = \mathbf{P}_{i,j} \cup \left\{t_a \mid t_a \Rightarrow p_{k_1,k_2,\cdots k_n} \text{ and } p_{k_1,k_2,\cdots,k_n} \in \mathbf{P}_{i,j}\right\}.$$

2) Generate a node $T_{i,j}$ by combining all processor nodes in $\overline{\mathbf{P}}_{i,j}$ and all the task nodes which are attached to one of these processors, i.e.,

$$T_{i,j} = \overline{\mathbf{P}}_{i,j} \cup \left\{t_b \mid t_b \Rightarrow p_{l_1,l_2,\cdots,l_n} \text{ and } p_{l_1,l_2,\cdots,l_n} \in \overline{\mathbf{P}}_{i,j}\right\}.$$

Each node $S_{i,j}$ is called a *source node* and $T_{i,j}$ a *sink node* of the corresponding two-terminal network graph $G_{i,j}$. For example, given a seven-task TIG to be executed on the $2 \times 3$ array network in Fig. 1, Fig. 3 shows the resulting $(2 \times 3)$-terminal network graph $G_{2\times3}$ and its corresponding three two-terminal network graphs with their cutsets defined as follows. In Fig. 3, tasks $t_1$, $t_3$, and $t_7$ are attached to specific processors, and the others are general tasks.

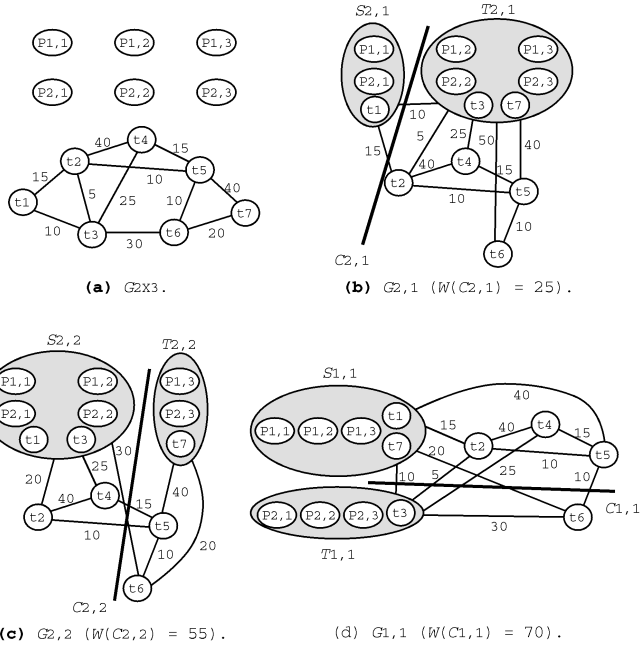



Fig. 3. The $(2 \times 3)$-terminal network graph $G_{2\times3}$ and three two-terminal network graphs. Tasks $t_1$, $t_3$, and $t_7$ are attached to processors $p_{1,1}$, $p_{2,2}$, and $p_{1,3}$, respectively.

DEFINITION 1. *A cutset $C_{i,j}$ of the two-terminal network graph $G_{i,j} = (V_{i,j}, E_{i,j})$ is a set of edges which when deleted, separate $\hat{S}_{i,j}$ from $\hat{T}_{i,j}$ such that $\hat{S}_{i,j} \cap \hat{T}_{i,j} = \varnothing$, $\hat{S}_{i,j} \cup \hat{T}_{i,j} = V_{i,j}$, $S_{i,j} \in \hat{S}_{i,j}$, and $T_{i,j} \in \hat{T}_{i,j}$. $\hat{S}_{i,j}$ is called the source set and $\hat{T}_{i,j}$ the sink set of the cutset. The weight of a cutset is the total weight of the edges in the cutset.*

Two cutsets $C_{i,j}$ and $C_{k,l}$ are said to cross each other if and only if each of the four sets $\hat{S}_{i,j} \cap \hat{S}_{k,l}$, $\hat{S}_{i,j} \cap \hat{T}_{k,l}$, $\hat{T}_{i,j} \cap \hat{S}_{k,l}$, and $\hat{T}_{i,j} \cap \hat{T}_{k,l}$ contains at least one node.

DEFINITION 2. *For each $i$th coordinate, let $C_i$ be a set of $(n_i - 1)$ cutsets $C_{i,j}$s each of which is on the corresponding two-terminal network graph $G_{i,j}$, i.e., $C_i = \{C_{i,j} \mid 1 \le j < n_i\}$. Then $C_i$ is said to be admissible if no two cutsets in $C_i$ cross each other. The weight of $C_i$, $W(C_i)$, is the total weight of the cutsets in $C_i$, i.e., $W(C_i) = \sum_j W(C_{i,j})$.*

DEFINITION 3. *Let $\mathbf{C}_A$ be the set of all cutsets $C_{i,j}$s, i.e., $\mathbf{C}_A = \bigcup_i C_i = \{C_{i,j} \mid 1 \le i \le n, 1 \le j < n_i\}$. Then $\mathbf{C}_A$ is said to be admissible if each $C_i$ is admissible. The weight of $\mathbf{C}_A$ is the total weight of the cutsets in $\mathbf{C}_A$, i.e., $W(\mathbf{C}_A) = \sum_i W(C_i) = \sum_i \sum_j W(C_{i,j})$.*

For example, a $(2 \times 3)$-terminal network graph $G_{2\times3}$ and a two-terminal network graph $G_{2,1}$ are presented in Figs. 4a and 4b, respectively, where the program graph TIG is just approximated by the outer circle. In Fig. 4b, $T_S$ ($T_T$) is the set of tasks which are attached to one of the processors in $\mathbf{P}_{2,1}(\overline{\mathbf{P}}_{2,1})$. Fig. 4c shows three cutsets $C_{1,1}$, $C_{2,1}$, and $C_{2,2}$, each of which is on the corresponding two-terminal network graph. Note that $C_1$, $C_2$ in Fig. 4c are all admissible.

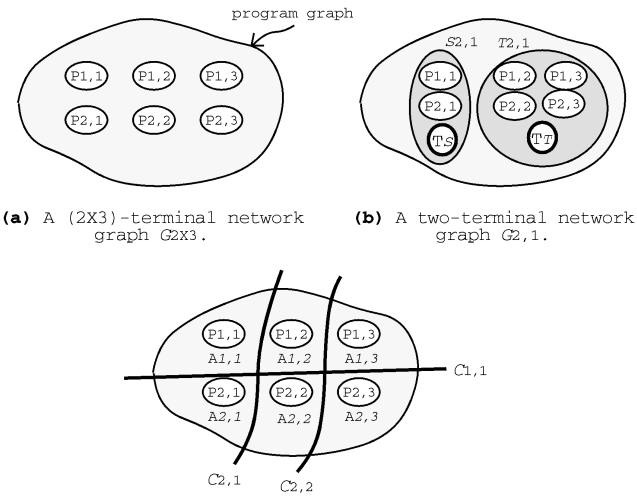



Fig. 4. $(2 \times 3)$-terminal and two-terminal network graphs with cutsets.

LEMMA 1. *Each admissible set $\mathbf{C}_A$ one-to-one corresponds to a feasible task assignment.*

PROOF. First, we prove that, for any admissible set $\mathbf{C}_A$, the

corresponding assignment $X$ is feasible, and then show that a feasible assignment produces an admissible set $\mathbf{C}_A$. For each $i$th coordinate, the $n_i - 1$ cutsets in $C_i$ partition the node set $V_N$ into $n_i$ disjoint subsets since $C_i$ is admissible. Therefore, all the cutsets in $\mathbf{C}_A$ partition $V_N$ into at most $N$ nonempty subsets. For any pair of processor nodes $p_{k_1,k_2,\cdots,k_n}$ and $p_{l_1,l_2,\cdots,l_n}$, without loss of generality, we can let $k_i < l_i$ for some $i$. Then these two processor nodes are separated by the cutset $C_{i,k_i}$. This means that every pair of nodes are separated by the cutsets in $\mathbf{C}_A$. Thus, the node set $V_N$ is partitioned into at least $N$ subsets by the cutsets in $\mathbf{C}_A$, because there are $N$ processor nodes in $V_N$. $V_N$ is therefore partitioned into exactly $N$ subsets by the cutsets in $\mathbf{C}_A$, each of which contains exactly one processor node. Let $A_{k_1,k_2,\cdots,k_n}$ be the subset that contains $p_{k_1,k_2,\cdots,k_n}$. Then $\mathbf{C}_A$ corresponds to the assignment $X$ where all tasks in $A_{k_1,k_2,\cdots,k_n}$ are assigned to $p_{k_1,k_2,\cdots,k_n}$. For each attached task $t_i \Rightarrow p_{k_1,k_2,\cdots,k_n}$, it is not separated from $p_{k_1,k_2,\cdots,k_n}$ by any cutset since $t_i$ and $p_{k_1,k_2,\cdots,k_n}$ are combined into a single node in every two-terminal network graph by the above procedure. Therefore, the corresponding assignment $X$ is feasible. Conversely, from a feasible assignment $X$, we can create the collection of cutsets. It is obvious that the set of created cutsets is admissible since no two cutsets can cross each other. So, each admissible set $\mathbf{C}_A$ one-to-one corresponds to a feasible task assignment. □

LEMMA 2. *Let an admissible set $\mathbf{C}_A$ correspond to a feasible assignment $X$. Then the weight of $\mathbf{C}_A$ is equal to the total communication cost of $X$.*

PROOF. We denote by $X(a)_i$ the value of the $i$th coordinate of the processor to which task $t_a$ is assigned under $X$. Then, the total communication cost of the assignment $X$ is such that

$$\begin{aligned} COMM(X) &= \sum_{a<b} W_{a,b} \cdot d_{X(a),X(b)} \\ &= \sum_{a<b} W_{a,b} \cdot \left( \sum_{i=1}^{n} \left| X(a)_i - X(b)_i \right| \right) \\ &= \sum_{i=1}^{n} \left( \sum_{a<b} W_{a,b} \cdot \left| X(a)_i - X(b)_i \right| \right) \\ &= \sum_{i=1}^{n} \left( \sum_{j=1}^{n_i-1} \sum_{\substack{1 \le X(a)_i \le j \\ j < X(b)_i \le n_i}} W_{a,b} \right) \\ &= \sum_{i=1}^{n} \left( \sum_{j=1}^{n_i-1} W\left(C_{i,j}\right) \right) \\ &= W\left(\mathbf{C}_A\right) \end{aligned}$$

thus proving the lemma. □

For example, the three cutsets in Fig. 4c correspond to the feasible assignment $X$ where every task in each subset $A_{i,j}$ is assigned to processor $p_{i,j}$. The total communication cost of the assignment $X$ is the sum of the weights of the cutsets, i.e., $COMM(X) = W(C_{1,1}) + W(C_{2,1}) + W(C_{2,2})$. Lemmas 1 and 2 say that the task assignment problem in an n-dimensional array network is equivalent to the problem of finding the minimum-weight admissible set $\mathbf{C}_{Ao}$ on the corresponding $N$-terminal network graph. In what follows we present the solution to this problem.

### 3.2 The Solution

From the correspondence of an admissible set $\mathbf{C}_A$ to a feasible assignment $X$, we can see that the value of the $i$th coordinate of the processor to which each task $t_a$ is assigned (i.e., $X(a)_i$) is uniquely determined by the cutsets in an admissible set $C_i$. Consider an $(n_1 \times n_2)$ two-dimensional array network. (We can easily generalize to three or more dimensions.) We can build $(n_1 - 1)$ two-terminal network graphs on columns and $(n_2 - 1)$ two-terminal network graphs on rows. The column problems determine which tasks are assigned to column 1 through $i$ and which are assigned to columns $i + 1$ through $n_1$ for each cutset $C_{1,i}$, $1 \le i < n_1$. Gomory and Hu [11] showed that minimum-weight cutsets do not have to cross each other. Hence, if all the column cutsets are minimum-weight cutsets on their corresponding two-terminal network graphs, we can determine uniquely to which column each task is assigned. By doing the same process for rows, we can determine uniquely to which row each task is assigned. We now show that if every cutset $C_{i,j}$ in a set $\mathbf{C}_A$ is a minimum-weight cutset on the corresponding two-terminal network graph $G_{i,j}$, then $\mathbf{C}_A$ is a minimum-weight admissible set $\mathbf{C}_{Ao}$ and it corresponds to an optimal assignment $X_o$. A minimum-weight admissible set $\mathbf{C}_{Ao}$ is obtained by the following procedure.

**procedure : ASSIGN_ARRAY**

**for** $i := 1$ **to** $n$ **do**

1) $S_{i,0} := \varnothing$;
2) **for** $j := 1$ **to** $n_i - 1$ **do**
   a) $S_{i,j} := S_{i,j-1} \cup \{p_{k_1,k_2,\cdots,k_n} \mid k_i = j\}$;
      $T_{i,j} := \{p_{k_1,k_2,\cdots,k_n} \mid j + 1 \le k_i \le n_i\}$;
   b) for every attached task $t_a$, say $t_a \Rightarrow p_{k_1,k_2,\cdots,k_n}$, if $k_i = j$ then $S_{i,j} := S_{i,j} \cup \{t_a\}$, else if $k_i > j$ then $T_{i,j} := T_{i,j} \cup \{t_a\}$;
   c) find a minimum-weight cutset $C_{i,j}$ of the two-terminal network graph $G_{i,j} = (V_{i,j}, E_{i,j})$ which separates $\hat{S}_{i,j}$ from $\hat{T}_{i,j}$ such that $\hat{S}_{i,j} \cap \hat{T}_{i,j} = \varnothing$, $\hat{S}_{i,j} \cup \hat{T}_{i,j} = V_{i,j}$, and such that $S_{i,j} \subseteq \hat{S}_{i,j}$ and $T_{i,j} \subseteq \hat{T}_{i,j}$;
   d) for every task $t_a$ in $\hat{S}_{i,j}$, if $t_a \notin \hat{S}_{i,j-1}$ then $S_{i,j} := S_{i,j} \cup \{t_a\}$ and $X(a)_i := j$;
   **endfor**
3) for every task $t_a$ in $\hat{T}_{i,n_i-1}$, $X(a)_i := n_i$;
   **endfor**

**end.**

For example, given a seven-task TIG to be executed on the $2 \times 3$ array network in Fig. 1, the resulting $(2 \times 3)$-terminal network graph $G_{2\times3}$ and its corresponding three two-terminal network graphs with their cutsets are presented in Fig. 3. Note that each cutset $C_{i,j}$ in Fig. 3 is a minimum-weight cutset in $G_{i,j}$. The following lemmas and theorem enable us to use the network-flow algorithm to find a minimum-cost assignment. The proof technique is similar to the proof technique used by Gomory and Hu [11].

LEMMA 3. *Let a set* $\mathbf{C}_A = \{C_{i,j} \mid 1 \le i \le n,\ 1 \le j \le n_i - 1\}$ *found in the procedure ASSIGN_ARRAY correspond to a task assignment X. Then the assignment X is feasible; that is,* $\mathbf{C}_A$ *is admissible.*

PROOF. For each $i$th coordinate, every task $t_a$ in $\hat{S}_{i,j}$ in Step 2.d for the $j$th iteration, $1 \le j < n_i - 1$, of the inner **for** loop is contained in $S_{i,j+1}$ by Step 2.a at the $(j+1)$th iteration. Thus, every task $t_a$ $(\in \hat{S}_{i,j})$ must be in $\hat{S}_{i,j+1}$ by the cutset $C_{i,j+1}$ found at Step 2.c. Consequently, a set $\mathbf{C}_A$ found in the procedure ASSIGN_ARRAY is admissible. □

LEMMA 4. *Let a set* $\mathbf{C}_A = \{C_{i,j} \mid 1 \le i \le n,\ 1 \le j \le n_i - 1\}$ *found in the procedure ASSIGN_ARRAY correspond to a feasible assignment X. Then, for any feasible assignment X', the following inequality holds for each* $i$th *coordinate:*

$$\sum_{\substack{1\le X(a)_i\le j \\ j<X(b)_i\le n_i}} W_{a,b} \le \sum_{\substack{1\le X'(c)_i\le j \\ j<X'(d)_i\le n_i}} W_{c,d}, \quad \text{for all } j.$$

PROOF Each cutset $C_{i,j}$ in $\mathbf{C}_A$ is a minimum-weight cutset of $G_{i,j}$ and separates $\hat{S}_{i,j}$ from $\hat{T}_{i,j}$. Every task in $\hat{S}_{i,j}$ is assigned to one of the processors $p_k$'s, $1 \le k_i \le j$, and every task in $\hat{T}_{i,j}$ is assigned to one of the processors $p_l$'s, $j+1 \le l_i \le n_i$ by the procedure ASSIGN_ARRAY. Then the weight of the cutset $C_{i,j}$, $W(C_{i,j})$, is:

$$W(C_{i,j}) = \sum_{\substack{1\le X(a)_i\le j \\ j<X(b)_i\le n_i}} W_{a,b}.$$

We prove the inequality by induction on $j$.

1) The result is true if $j = 1$, since $C_{i,1}$ is a minimum-weight cutset.
2) Suppose it holds for $j = k - 1$. Without loss of generality, assume that the task nodes are partitioned into two subsets $A$ and $B$ by the minimum cutset $C_{i,k-1}$ as shown in Fig. 5a. Let $c(A, B)$ denote the sum of the weights of all edges between two sets $A$ and $B$. Then, $W(C_{i,k-1}) = c(A, B)$. By the procedure ASSIGN_ARRAY, the next cutset $C_{i,k}$ cannot partition the task nodes in $A$ any more since every task node in $A$ is already included into $S_{i,k}$ at the $k$th iteration of the inner for loop of the procedure. Let $C_{i,k}$ partition the task nodes into two subsets $A \cup B_2$ and $B_2'$ (see Fig. 5b), i.e.,

$$\begin{aligned} W(C_{i,k}) &= c(A, B_2') + c(B_2, B_2') \\ &= c(A_1, B_2') + c(A_1', B_2') + c(B_2, B_2'), \end{aligned}$$

where $A_1 \cup A_1' = A$ and $B_2 \cup B_2' = B$. (Each of the subsets may be empty, but this will not alter the proof.) We prove this by contradiction. Suppose the inequality does not hold for $j = k$. Then there exists another feasible assignment $X'$ which partitions the task nodes into two subsets $A_1 \cup B_1$ and $A_1' \cup B_1'$, where $B_1 \cup B_1' = B$ (see Fig. 5c), such that

$$\begin{aligned} &c(A_1, A_1') + c(A_1, B_1') + c(B_1, A_1') + c(B_1, B_1') \\ &\quad < c(A_1, B_2') + c(A_1', B_2') + c(B_2, B_2'). \end{aligned} \tag{3.1}$$

Every task in $A_1'$ is executable on at least one of $p_l$'s for $k+1 \le l_i \le n_i$, since $X'$ is feasible. Thus, the cutset $C'_{i,k-1}$ shown in Fig. 5d is a cutset of a feasible assignment which assigns every task in $A_1$ to one of $p_l$'s for $1 \le l_i \le k-1$ and every task in $A_1' \cup B_1 \cup B_1'$ to one of $p_l$'s for $k \le l_i \le n_i$. Then $W(C_{i,k-1}) \le W(C'_{i,k-1})$ by the assumption for $j = k-1$, i.e.,

$$\begin{aligned} &c(A_1, B_1) + c(A_1', B_1) + c(A_1, B_1') + c(A_1', B_1') \\ &\quad \le c(A_1, A_1') + c(A_1, B_1) + c(A_1, B_1'). \end{aligned} \tag{3.2}$$

Every task in $B_1$ is executable on at least one of $p_l$'s for $k \le l_i \le n_i$, since $X$ is feasible. Also, every task in $B_1$ is executable on at least one of $p_l$'s for $1 \le l_i \le k$, since $X'$ is feasible. Thus, every task in $B_1$ is executable on $p_l$, $l_i = k$. The cutset $C'_{i,k}$ in Fig. 5d is a possible cutset in $G_{i,k}$ which assigns every task in $B_1$ to $p_k$ (we assumed that every task in $A_1 \cup A_1'$ has already been assigned to one of the processors $p_l$'s, for $1 \le l_i \le k-1$, by the cutsets $C_{i,1}, C_{i,2}, \cdots, C_{i,k-1}$). Then $W(C_{i,k}) \le W(C'_{i,k})$ since $C_{i,k}$ is a minimum-weight cutset in $G_{i,k}$, i.e.,

$$\begin{aligned} &c(A_1, B_2') + c(A_1', B_2') + c(B_2, B_2') \\ &\quad \le c(A_1, B_1') + c(A_1', B_1') + c(B_1, B_1'). \end{aligned} \tag{3.3}$$

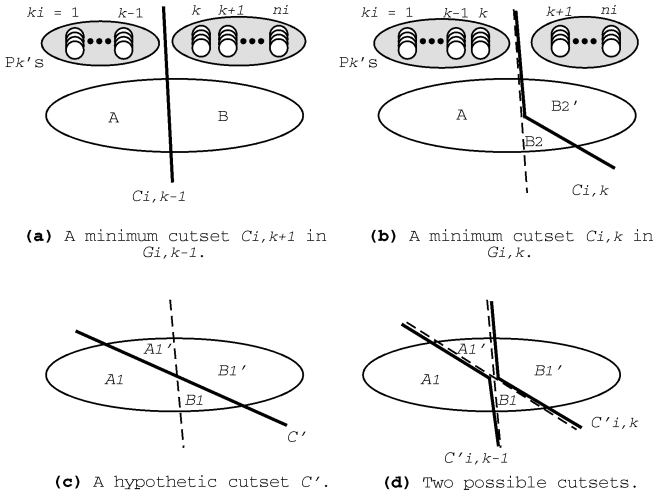
By combining the above three inequalities (3.1), (3.2), and (3.3), we obtain the following inequality:

$$c(A_1', B_1) < 0.$$

This contradicts the fact that the weight between any two subsets cannot be negative. Thus, the inequality holds for $j = k$. □

THEOREM 1. *Let a set* $\mathbf{C}_A = \{C_{i,j} \mid 1 \le i \le n,\ 1 \le j \le n_i - 1\}$ *found in the procedure ASSIGN_ARRAY correspond to a feasible assignment X. Then the assignment X is an optimal assignment with the total communication cost of* $\sum_i \sum_j W(C_{i,j})$.

PROOF By contradiction, assume that $X$ is not an optimal assignment. Let another feasible assignment $X'$ be an optimal assignment, i.e., $COMM(X') < COMM(X)$. Then there exists at least one $j$, $1 \le j \le n_i - 1$, for at least one coordinate $i$, $1 \le i \le n$, such that




**(a)** A minimum cutset $C_{i,k+1}$ in $G_{i,k-1}$. **(b)** A minimum cutset $C_{i,k}$ in $G_{i,k}$. **(c)** A hypothetic cutset $C'$. **(d)** Two possible cutsets.

Fig. 5. Illustrative figures for Lemma 4.

$$\sum_{\substack{1\le X'(a)_i\le j\\ j<X'(b)_i\le n_i}} W_{a,b} < \sum_{\substack{1\le X(c)_i\le j\\ j<X(d)_i\le n_i}} W_{c,d}.$$

This is contradictory to Lemma 4, and hence, $X$ is an optimal assignment. The total communication cost of $X$ is $\sum_i\sum_j W(C_{i,j})$ by Lemma 2. Thus, the theorem follows. □

In the above example shown in Fig. 3, the cutsets $C_{1,1}$, $C_{2,1}$, and $C_{2,2}$ corresponds to the optimal assignment, where $t_1$ is assigned to $p_{1,1}$, $t_2$ and $t_4$ to $p_{1,2}$, $t_5$ and $t_7$ to $p_{1,3}$, $t_3$ to $p_{2,2}$, and $t_6$ to $p_{2,3}$, as shown in Fig. 6. The total weight of the cutsets, $\sum_i\sum_j W(C_{i,j})$, is 150 which is equal to the total communication cost of the assignment. Since a set $\mathbf{C}_A$ of minimum-weight cutsets $C_{i,j}$s found in the procedure ASSIGN_ARRAY corresponds to a feasible assignment with the minimum total communication cost by Theorem 1, the solutions to the task assignment problem for homogeneous array networks can be found in polynomial time. There exist many algorithms [6], [8], [9] that find efficiently the minimum cutset of a two-terminal network graph, started by Ford and Fulkerson [8]. The best known algorithm is proposed by Goldberg and Tarjan [9] with the time complexity of $O(EV \log(V^2/E))$, where $E$ and $V$ are the number of edges and the number of nodes of the network graph, respectively. Each two-terminal network graph developed in the previous subsection has at most $(M + 2)$ nodes and $M(M - 1)/2$ edges, where $M$ is the number of tasks. Thus, we can find each of the $\sum_i(n_i - 1)$ minimum-weight cutsets in time no worse than $O(M^3)$. Therefore, the task assignment problem for a homogeneous $n$-dimensional array network can be solved in time no worse than $O(\sum_i(n_i - 1)M^3)$ by applying the network-flow algorithm to each of the $\sum_i(n_i - 1)$ two-terminal network graphs constructed in the procedure ASSIGN_ARRAY. The hypercube is a special case of the array network, i.e., $n_i = 2$ for all $i$. Thus, we can solve the task assignment problem in an $n$-dimensional hypercube with $N = 2^n$ processors in time no worse that $O(nM^3)$.

## 4 Assignment in Tree Networks

A homogeneous tree network is composed of $N$ functionally-identical processors $p_1, p_2, \cdots, p_N$ connected via a tree structure with $N - 1$ communication links. There is only one path between any two processors $p_k$ and $p_l$ in a tree, since there is no cycle in a tree. The distance between two processors is then the number of links in the path connecting them.

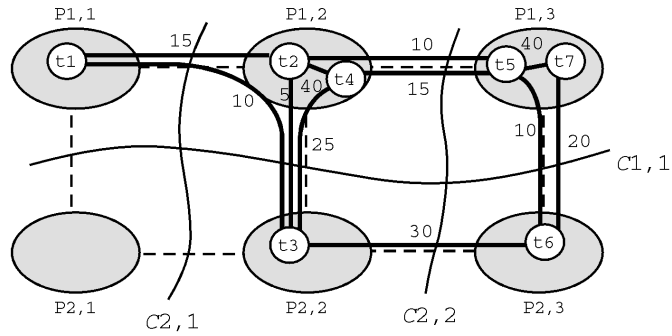



Fig. 6. The optimal assignment $X_o$ ($COMM(X_o) = 150$).

### 4.1 The Cutset Formulation

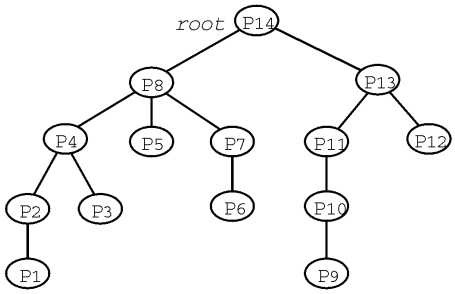
For tractability, we arrange the processors in depth-first search order. Given an undirected processor graph $G_T = (V_T, E_T)$ with $N$ nodes representing processors and $N - 1$ edges representing communication links between the processors in the network, we first choose an arbitrary processor node as the *root*. Then, we assign a new ID number (starting from 1) to each processor node in sequence as they are visited by the postorder traversal algorithm. For example, a tree network of 14 processors is shown in Fig. 7a with new IDs. From now on, we will use the new ID for each processor. The *ancestors* of a node $p_k$ are all the nodes along the path from $p_k$ to the root node. The *parent* of a node is the first node visited along the path to the root node. If node $p_k$ is an ancestor (parent) of node $p_l$, then we call $p_l$ a descendent (child) of $p_k$. Note that every node is assigned a higher ID than its descendents. Also, for each pair of processors $p_k$ and $p_{k+1}$, $p_{k+1}$ is a parent of $p_k$ or $p_{k+1}$ is a leaf node. Let $\ell_k$ be the communication link between $p_k$ and its parent node. Then each $\ell_k$ separates $p_k$ with all its descendents from the other processor nodes. We denote by $\mathbf{P}_k$ the set of a processor node $p_k$ and all of its descendents, and by $\overline{\mathbf{P}}_k$ the set of the other nodes, i.e., $\mathbf{P}_k \cup \overline{\mathbf{P}}_k = V_T$. In Fig. 7a, the communication link between $p_4$ and $p_8$ is $l_4$, and $\mathbf{P}_4 = \{p_1, p_2, p_3, p_4\}$ and $\overline{\mathbf{P}}_4 = \{p_5, p_6, \cdots, p_{14}\}$.

Given the TIG = $(V, E)$ of a distributed application submitted to an $N$-processor tree, we first make a corresponding $N$-terminal network graph $G_N = (V_N, E_N)$, by adding all the processor nodes to the TIG as was done for the array network. Then, we construct $(N - 1)$ two-terminal network graphs $G_i$s, $1 \le i < N$, from the $N$-terminal network graph $G_N$ as follows.

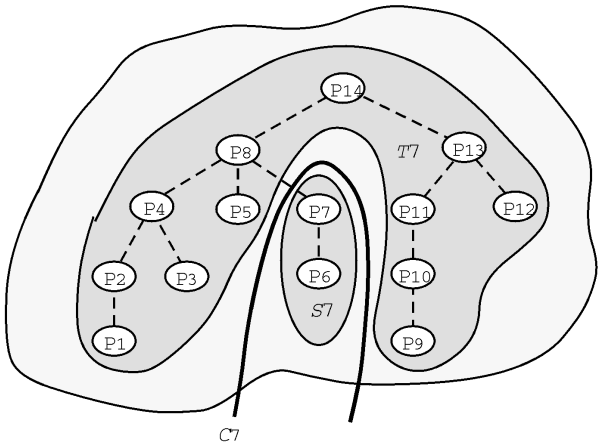
1) Generate a source node $S_i$ by combining all the processor nodes in $\mathbf{P}_i$ and all the task nodes which are attached to one of these processors.

2) Generate a sink node $T_i$ by combining all the processor nodes in $\overline{\mathbf{P}}_i$ and all the task nodes which are attached to one of these processors.

Note that the root node $p_N$ is always combined with the sink node $T_i$, since $p_N \in \overline{\mathbf{P}}_i$ for all $1 \le i < N$. For example, a two-terminal network graph $G_7$ for the example in Fig. 7a is shown in Fig. 7b with its cutset $C_7$.




**(a)** A tree network of 14 processors.




**(b)** A two-terminal network graph $G_7$ and its cutset $C_7$.

Fig. 7. An example.

DEFINITION 4. *Let $\mathbf{C}_T$ be a set of $(N-1)$ cutsets $C_i$s each of which is on the corresponding two-terminal network graph $G_i$. Then $\mathbf{C}_T$ is said to be admissible if no two cutsets in $\mathbf{C}_T$ cross each other.*

LEMMA 5. *Each admissible set $\mathbf{C}_T$ one-to-one corresponds to a feasible task assignment.*

PROOF. Each admissible set $\mathbf{C}_T$ of a graph $G_N$ partitions the nodes of $G_N$ into $N$ subsets $A_k$s each of which has exactly one processor node $p_k$. Then we can associate $\mathbf{C}_T$ with the assignment that every task in $A_k$ is assigned to $p_k$, and vice versa. □

LEMMA 6. *Let an admissible set $\mathbf{C}_T$ correspond to a feasible assignment $X$. Then the weight of $\mathbf{C}_T$ is equal to the total communication cost of $X$.*

PROOF. The communication cost imposed on each link $\ell_i$ under the assignment $X$, $comm(X, l_i)$, is equal to the weight of the corresponding cutset $C_i$. Then the total communication cost is the sum of the communication costs imposed on all the links, i.e.,

$$COMM(X) = \sum_{i=1}^{N-1} comm(X, l_i) = \sum_{i=1}^{N-1} W(C_i) = W(\mathbf{C}_T).$$

This proves the lemma. □

### 4.2 The Solution

In what follows, we show that if every cutset $C_i$ in an admissible set $\mathbf{C}_T$ is a minimum-weight cutset on the corresponding two-terminal network graph $G_i$, then $\mathbf{C}_T$ is a minimum-weight admissible set $\mathbf{C}_{To}$ and it corresponds to an optimal assignment $X_o$. The correctness follows for the same reason as for array networks. Each cutset in the tree determines if a task is assigned to a particular subtree or not. Since minimum-weight cutsets do not have to cross each other, we can determine uniquely to which processor each task is assigned. A minimum-weight admissible set $\mathbf{C}_{To}$ is obtained by the following procedure.

**procedure : ASSIGN_TREE**

1) declare that every task is not assigned;
2) **for** $i := 1$ **to** $N-1$ **do**
   a) $S_i := \mathbf{P}_i$; $T_i := \overline{\mathbf{P}}_i$;
   b) **for** every task $t_a$, **if** it is attached or assigned to one of the processors in $\mathbf{P}_i$ **then** $S_i := S_i \cup \{t_a\}$, **else if** it is attached or assigned to one of the processors in $\overline{\mathbf{P}}_i$ **then** $T_i := T_i \cup \{t_a\}$;
   c) find a minimum-weight cutset $C_i$ of the two-terminal network graph $G_i = (V_i, E_i)$ which separates $\hat{S}_i$ from $\hat{T}_i$ such that $\hat{S}_i \cap \hat{T}_i = \varnothing$, $\hat{S}_i \cup \hat{T}_i = V_i$, and such that $S_i \subseteq \hat{S}_i$ and $T_i \subseteq \hat{T}_i$;
   d) for every unassigned task $t_a$ in $\hat{S}_i$, declare that it is assigned to $p_i$;
   e) **endfor**
3) declare that all the unassigned tasks are assigned to $p_N$;

**end.**

LEMMA 7. *Let a set $\mathbf{C}_T = \{C_i \mid 1 \le i < N\}$ found in the procedure ASSIGN_TREE correspond to a task assignment $X$. Then the assignment $X$ is feasible; that is, $\mathbf{C}_T$ is admissible.*

PROOF: For each $i$th iteration, every unassigned task in $\hat{S}_i$ is assigned to $p_i$ at Step 2.d. Any cutset $C_j$, $i < j$, cannot partition $\hat{S}_i$ any more, since every assigned task is contained in $S_j$ or $T_j$ at Step 2.b for the $j$th iteration. Therefore, any two cutsets found by the procedure ASSIGN_TREE do not cross each other. □

LEMMA 8. *Let a set $\mathbf{C}_T = \{C_i \mid 1 \le i < N\}$ found in the procedure ASSIGN_TREE correspond to a feasible assignment $X$. Then, for any feasible assignment $X'$, the following inequality holds for each communication link $\ell_i$:*

$$comm(X, l_i) \le comm(X', l_i), \text{ for all } i.$$

PROOF. We prove the inequality by induction on $i$. The proof is similar to the proof technique used in Lemma 4.

1) The result holds if $i = 1$, since $C_i$ is a minimum-weight cutset.
2) Suppose it holds for $1 \le i \le k-1$. Without loss of generality, we can assume that the task nodes are partitioned into three subsets $X$, $Y$, and $Z$, such that each task in $X$ or $Y$ is assigned to one of the processors in $P_X$ or $P_Y$, respectively, by the $(k-1)$ minimum cutsets $C_j$s, $1 \le j \le k-1$, as symbolically shown in Fig. 8a, where

$$\mathbf{P}_X = \mathbf{P}_k - \{p_k\},$$
$$\mathbf{P}_Z = \{p_l \mid l > k\}, \text{ and}$$
$$\mathbf{P}_Y = V_T - \mathbf{P}_k - \mathbf{P}_Z.$$

Then, by assumption 2, the cutset $C_X$ ($C_Y$) in Fig. 8a is a minimum cutset on the corresponding two-terminal network graph $G_X$ ($G_Y$) where $\mathbf{P}_X \subseteq S_X$ ($\mathbf{P}_Y \subseteq S_Y$) and all the other processor nodes are in $T_X$ ($T_Y$). Note that if $p_k$ is a leaf node then both of the sets $\mathbf{P}_X$ and $X$ become empty, but this will not alter the proof. Let $C_k$ partition the task nodes into two subsets $X \cup Z_1$ and $Z_2 \cup Y$ (see Fig. 8b), i.e.,

$$W(C_k) = comm(X, l_k)$$
$$= c(X, Z_2) + c(X, Y) + c(Z_1, Z_2) + c(Z_1, Y).$$

where $Z_1 \cup Z_2 = Z$. Suppose the inequality does not hold for $i = k$. Then there exists another feasible assignment $X'$ which partitions the task nodes into two subsets $X_1 \cup Z_{11} \cup Z_{21} \cup Y_1$ and $X_2 \cup Z_{12} \cup Z_{22} \cup Y_2$ (see Fig. 8c), such that

$$comm(X', l_k) = W(C') < comm(X, l_k) = W(C_k), \quad (4.1)$$

where $X_1 \cup X_2 = X$, $Z_{11} \cup Z_{12} = Z_1$, $Z_{21} \cup Z_{22} = Z_2$, and $Y_1 \cup Y_2 = Y$. Then, every task in $X_2$, $Z_{12}$, $Z_{21}$, and $Y_1$ must not be an attached task, since both $X$ and $X'$ are feasible assignments. The three cutsets $C'_X$, $C'_Y$, and $C'_k$ shown in Fig. 8d are all possible cutsets on the corresponding two-terminal network graphs $G_X$, $G_Y$, and $G_k$, respectively. Since each of the cutsets $C_X$, $C_Y$, and $C_k$ is a minimum cutset, the following three inequalities hold:

$$W(C_X) \le W(C'_X), \quad (4.2)$$

$$W(C_Y) \le W(C'_Y), \quad (4.3)$$

$$W(C_k) \le W(C'_k). \quad (4.4)$$

By combining the four inequalities (4.1), (4.2), (4.3), and (4.4), we obtain:

$$c(X_2, Z_{11} \cup Z_{21} \cup Y_1) + c(Z_{12} \cup Z_{22}, Y_1) < 0.$$

This is a contradiction since all weights are nonnegative. Thus, the inequality holds for $i = k$ and the inequality holds for all $i$ by the principle of induction. □

THEOREM 2. *Let a set $\mathbf{C}_T = \{C_i \mid 1 \le i < N\}$ found in the procedure ASSIGN_TREE correspond to a feasible assignment $X$. Then, the assignment $X$ is an optimal assignment with the total communication cost of $\sum_i W(C_i)$.*

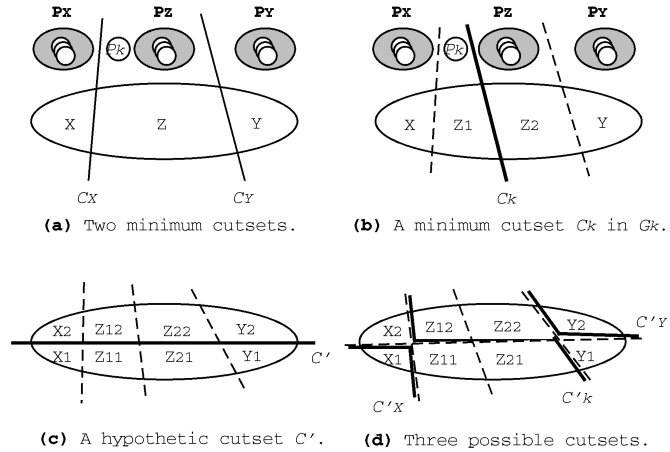



Fig. 8. Illustrative figures for Lemma 8.

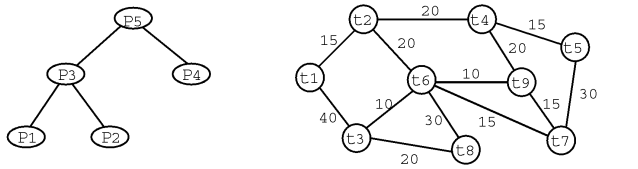
PROOF. By contradiction, assume that $X$ is not an optimal assignment. Let another feasible assignment $X'$ be an optimal assignment, i.e., $COMM(X') < COMM(X)$. Then there exists at least one $i$, $1 \le i < N$, such that $comm(X', l_i) < comm(X, l_i)$. This is contradictory to the result of Lemma 8, and thus, $X$ is an optimal assignment. The total communication cost of $X$ is $\sum_i W(C_i)$ by Lemma 6. □

Theorem 2 says that the optimal task assignment in a homogeneous tree network of $N$ processors can be found by applying the network-flow algorithm $N-1$ times each on the corresponding two-terminal network graph. Each two-terminal network graph (as developed in the previous subsection) has at most $M+2$ nodes and $M(M-1)/2$ edges. Therefore, the task assignment problem for an $N$-processor tree network can be solved in time no worse than $O(NM^3)$ by applying the network-flow algorithm to each of the $N-1$ two-terminal network graphs. For example, consider a five-processor tree network and a nine-task TIG as shown in Figs. 9a and 9b, respectively, where $t_1$, $t_6$, $t_8$, and $t_9$ are attached to $p_1$, $p_3$, $p_2$, and $p_4$, respectively. The optimal assignment for this example is presented in Fig. 10, where $t_1$ and $t_3$ are assigned to $p_1$, $t_8$ to $p_2$, $t_2$ and $t_6$ to $p_3$, $t_9$ to $p_4$, and $t_4$, $t_5$ and $t_7$ to $p_5$. The total communication cost of the assignment is 185 which is equal to the sum of the weights of the cutsets, i.e., $COMM(X_o) = W(C_1) + W(C_2) + W(C_3) + W(C_4) = 45 + 50 + 45 + 45 = 185$.

We have assumed that the distance between each processor $p_i$ and its parent $p_{parent(i)}$ in a tree network is equal to 1, i.e., $d_{i,parent(i)} = 1$. But this restriction can be relaxed to any $d_{i,parent(i)} > 0$, since Theorem 2 and all lemmas in this section hold even for the relaxed case. In such case, the set $\mathbf{C}_T$ found in the procedure ASSIGN_TREE corresponds to an optimal assignment with the minimum total communication cost of $\sum_i W(C_i) \cdot d_{i,parent(i)}$.

## 5 CONCLUSION

The problem of assigning the tasks of a distributed application to the processors of a distributed system is in general NP-complete. The problem is shown to be tractable only for a system of two processors [16] or a linear array of $N$ proc-




**(a)** A tree network with 5 processors. **(b)** An example program graph with 9 tasks where t1, t6, t8, and t9 are attached to p1, p3, p2, and p4, respectively.

Fig. 9. An example program graph on a tree network.

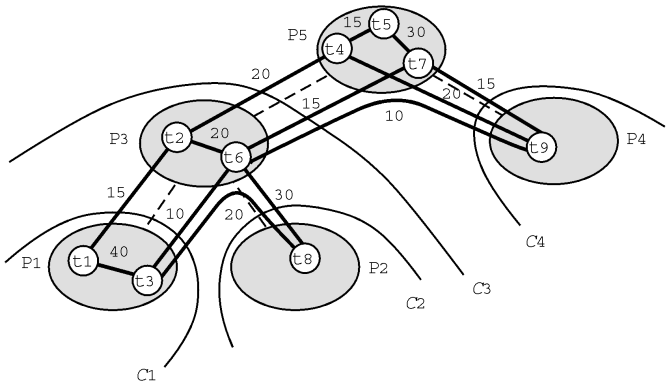



Fig. 10. The optimal assignment $X_{fo}$ ($COMM(X_{fo}) = 185$).

essors [12], and for distributed programs in which intertask relationships are constrained in certain ways [3], [19]. In this paper, we investigated the assignment problem in homogeneous networks in the presence of attached tasks. We showed that the assignment problem in an $N$-processor homogeneous network may be tractable for certain interconnection topologies and may not be tractable for others.

Our investigation of the problem has led to the development of a modeling technique that is sensitive to the interconnection topology, and transforms the assignment problem to a minimum-cut maximum-flow problem. We applied the modeling technique successfully to solve the problem of assigning $M$ tasks in an $n$-dimensional array and an $N$-processor tree in time no worse than $O(\sum_i(n_i - 1)M^3)$ and $O(NM^3)$, respectively.

Since the assignment problem for both the array and tree networks has an efficient solution, the problem for certain other topologies is likely to be tractable. It may be possible to apply our graph-theoretic modeling approach to obtain efficient solutions for other cases of the assignment problem. These are a matter of our future inquiry.

## ACKNOWLEDGMENTS

The work reported in this paper was supported in part by the U.S. National Science Foundation under Grant MIP-9203895 and by the U.S. Office of Naval Research under Grant N00014-94-1-0229. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the funding agencies.

## REFERENCES

[1] R.K. Arora and S.P. Rana, "Heuristic Methods for Process Assignment in Distributed Computing Systems," *Information Processing Letters*, vol. 11, pp. 199-203, Dec. 1980.

[2] S.H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," *IEEE Trans. Software Eng.*, vol. 5, pp. 341-349, July 1979.

[3] S.H. Bokhari, "A Shortest Tree Algorithm For Optimal Assignments Across Space and Time in a Distributed Processor Systems," *IEEE Trans. Software Eng.*, vol. 7, pp. 583-589, Nov. 1981.

[4] S.H. Bokhari, *Assignment Problems in Parallel and Distributed Computing*. Boston: Kluwer Academic, 1987.

[5] S.H. Bokhari, "Partitioning Problems in Parallel, Pipelined and Distributed Computing," *IEEE Trans. Computers*, vol. 37, no. 1, pp. 48-57, Jan. 1988.

[6] J. Edmonds and R.M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Algorithms," *J. ACM*, vol. 19, pp. 248-264, Apr. 1972.

[7] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *Computer*, pp. 50-56, June 1982.

[8] L.R. Ford Jr. and D.R. Fulkerson, *Flows in Networks.* Princeton, N.J.: Princeton Univ. Press, 1962.

[9] A.V. Goldberg and R.E. Tarjan, "A New Approach to the Maximum Flow Problem," *Proc. 18th Ann. Symp. Theory of Computing*, pp. 136-146, 1987.

[10] O. Goldschmidt and D.S. Hochbaum, "Polynomial Algorithm for the k-Cut Problem," *Proc. 29th Ann. Symp. Foundations of Computer Science*, pp. 444-451, 1988.

[11] R.E. Gomory and T.C. Hu, "Multiterminal Network Flows," *J. SIAM*, vol. 9, pp. 551-570, Dec. 1961.

[12] C.-H. Lee, D. Lee, and M. Kim, "Optimal Task Assignment in Linear Array Networks," *IEEE Trans. Computers*, vol. 41, no. 7, pp. 877-880, July 1992.

[13] C.-H. Lee, M. Kim, and C.-I. Park, "An Efficient k-Way Graph Partitioning Algorithm for Task Allocation in Parallel Computing Systems," *Proc. First Int'l Conf. Systems Integration*, pp. 748-751, 1990.

[14] V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *IEEE Trans. Computers*, vol. 37, no. 11, pp. 1,384-1,397, Nov. 1988.

[15] D.M. Nicol and D.R. O'Hallaron, "Improved Algorithms for Mapping Pipelined and Parallel Computations," *IEEE Trans. Computers*, vol. 40, no. 3, pp. 295-306, Mar. 1991.

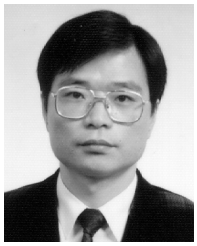
[16] H.S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. Software Eng.*, vol. 3, pp. 85-93, Jan. 1977.

[17] H.S. Stone, "Critical Load Factors in Distributed Computer Systems," *IEEE Trans. Software Eng.*, vol. 4, pp. 254-258, May 1978.

[18] H.S. Stone and S.H. Bokhari, "Control of Distributed Processes," *Computer*, vol. 11, pp. 97-106, July 1978.

[19] D.F. Towsley, "Allocating Programs Containing Branches and Loops within a Multiple Processor System," *IEEE Trans. Software Eng.*, vol. 12, pp. 1,018-1,024, Oct. 1986.

[20] B.-R. Tsai and K.G. Shin, "Communication-Oriented Assignment of Task Modules in Hypercube Multicomputers," *Proc. 12th Int'l Conf. Distributed Computing Systems*, pp. 38-45, 1992.



**Cheol-Hoon Lee** received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1983, and the MS and PhD degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Seoul, Korea, in 1988 and 1992, respectively.

From 1983 to 1994, he worked for Samsung Electronics Company in Seoul, Korea, as a researcher. From 1994 to 1995, he was with the University of Michigan, Ann Arbor, as a research scientist at the Real-Time Computing Laboratory. Since 1995, he has been a full-time instructor in the Department of Computer Engineering, Chungnam National University, Daejon, Korea. His research interests include parallel processing, operating system, and fault-tolerant computing.

**Kang G. Shin** received the BS degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. Dr. Shin is a professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor.

He has authored or coauthored more than 360 technical papers (about 150 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He has written (jointly with C.M. Krishna) a textbook, *Real-Time Systems*, which is scheduled to be published by McGraw-Hill in 1996. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

Dr. Shin has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, and manufacturing applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. (The latter is being pursued as a key thrust area of the newly-established NSF Engineering Research Center on Reconfigurable Machining Systems.)

He is an IEEE fellow and chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a distinguished visitor of the Computer Society of the IEEE, an editor of *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*.