A Generalized Scheme for Mapping Parallel Algorithms

Vipin Chaudhary, *Member, IEEE*, and J. K. Aggarwal, *Fellow, IEEE*

Abstract—**The mapping problem arises when the dependency structure of a parallel algorithm differs from the processor interconnection of the parallel computer or when the number of processes generated by the algorithm exceeds the number of processors available. The mapping problem (also known as task allocation) has been widely studied. We propose a new generalized mapping strategy that uses a combination of graph theory, mathematical programming, and heuristics. The key difference between our strategy and those proposed previously is the interdependence between the algorithm and the architecture. We use the knowledge from the given algorithm and the architecture to guide the mapping. The approach begins with a graphical representation of the parallel algorithm (*problem graph*) and the parallel computer (*host graph*). Using these representations, we generate a new graphical representation (*extended host graph*) on which the problem graph is mapped. We use an accurate characterization of the communication overhead in our objective functions to evaluate the optimality of the mapping. An efficient mapping scheme is developed which uses two levels of optimization procedures. The objective functions include minimizing the communication overhead and minimizing the total execution time which includes both computation and communication times. The mapping scheme is tested by simulation and further confirmed by mapping a real world application onto actual distributed environments.**

Index Terms—**Deadlock, feasibility, mapping, objective functions, scheduling, strongly connected components.**

I. Introduction

THE notion that a cooperating collection of loosely coupled processors could function as a more powerful general purpose computing facility has existed for quite some time. If properly designed and planned, such a collection of processors provides a more economical and reliable approach than that of centralized processing systems. Much work has been focused on the problem of cooperation among distributed resources of a system, resulting in a myriad of techniques and methodologies. Most of the proposed strategies apply to specific architectures and algorithms. On the other hand, little research attempts a generalized approach to the above problem. While the idea of distributed computing is tantalizing, various practical and theoretical problems must be solved to realize the idea's potential. The major problems encountered are due to the interprocessor communication and delay because of dependency between subtasks. The mapping problem arises when the dependency structure of a parallel algorithm differs from the processor interconnection of the parallel computer (topological variation), or when the number of processes generated by the algorithm exceeds the number of processors available (cardinality variation).

The mapping problem, as described above, has been described a number of times and in a number of different ways in the literature [1]–[10]. The mapping problem can be considered as a distributed scheduling problem or as a resource allocation problem. An implicit distinction often exists between the terms scheduling and allocation. However, it can be argued that these are merely alternative formulations of the same problem, with allocation posed in terms of resource allocation (from the resources' point of view) and scheduling viewed from the consumer's point of view [5]. In this sense, allocation and scheduling are merely two terms describing the same general mechanisms but from different viewpoints. The mapping problem incorporates both these viewpoints.

In order to complete a task in a minimum execution time, it is desirable to take advantage of parallel processing. This type of problem, usually referred to as the minimum execution time (schedule length) multiprocessor scheduling problem, has been studied extensively [11]. The above problem, however, is extremely difficult to solve and generally intractable. It is well known that some simplified subproblems constructed from the original scheduling problem by imposing a variety of constraints still fall in the class of NP-hard problems [11]–[14], [8], [15]. The obvious approach then is to concentrate on the development of polynomial time algorithms that provide near optimal solutions. A new generalized mapping strategy is proposed that optimizes a set of accurately specified objective functions for real-time applications in a distributed computing environment.

A. Problem Statement

This paper focuses on the problem of optimally allocating processes (commonly referred to as task allocation) and scheduling them. Henceforth, we use the terms process and tasks interchangeably. Optimal task allocation in a distributed computing environment requires the optimal assignment of processors to computations and communication resources to the implementation of dependency relations between the unit of computation able to be scheduled, i.e., tasks, in order to minimize certain objective functions. A distributed computing environment has conflicting requirements [16]:

Manuscript received October 2, 1990; revised January 17, 1992.

V. Chaudhary is with the Parallel and Distributed Computing Laboratory, Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202.

J. K. Aggarwal is with the Computer and Vision Research Center, Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712-1084.

IEEE Log Number 9205854.

- While minimizing interprocessor communication tends to assign the entire computation to a single processor, load balancing tries to distribute the computation evenly among the processors.
- While real-time constraint uses as many processors as possible to maximize parallel execution, the precedence relationships limit parallel execution.
- The saturation effect suggests the use of fewer processors since inefficiency increases with the number of processors.

A parallel computation can be represented by a directed acyclic graph [9], [17], [18]. We represent a parallel computation by a directed static graph, the *problem graph* $S = (V, E)$, whose vertices $v^i \in V$ represent processes and whose edges $e^{ij} \in E$ represent communication paths. Note that this graph is distinct from the data dependency graph, since it could be cyclic. Edges of the problem graph are assigned weights to indicate the traffic intensity along that edge. This problem graph is a restricted model in that it does not allow new nodes or edges to be created during run time. This restriction avoids ambiguity in determining the computation and communication requirements of the problem graph. We also assume that the subtasks are nonpreemptive.

The system architecture specifies the actual hardware structure of the multiprocessor. The architecture consists of a network of processors, each of which is composed of a *processing node* and a *communication node*. The network is an interconnection between processor communication nodes. It performs the routing of data involved in the processor communication, and is known prior to the mapping. Fig. 3 illustrates an example of a host architecture. The traffic routing occurs at the communication node and is independent of the processing node. This separation of communication performed by the communication nodes allows the processors to achieve computational efficiency. The communication is packet switched. In addition, each processor has a queue which is used for both mapping and packet switching. The interconnection network structure of the host architecture is described by an undirected *host graph* $H = (\mathcal{P}, \mathcal{E})$. $\mathcal{P} = \{p^1, p^2, \cdots, p^n\}$ is the set of nodes and $\mathcal{E} = \{(p^i, p^j) \mid (p^i, p^j)$ is a data link$\}$ is the set of edges. The nodes of the host graph represent the processors and the edges represent links. Each edge can be weighted on its data capacity or length.

The basic problem we are trying to solve is to find a mapping of S onto H which optimizes certain objective functions. The objective functions usually calculate the cost of the mapping process based on certain cost functions. These costs must be minimized to obtain an optimal mapping. The various costs that need to be considered include the following:

- The execution cost of each process on each of the (heterogeneous) processors.
- The (interprocessor) communication cost incurred between processes when they are assigned to different processors.

The above discussion assumes that the processors could be heterogeneous. The generalized approach suggested here is a substantial advancement in scope over the current state-of-the-art. Most researchers have dealt with methodologies applicable only to specific configurations of resources or specific problems [19]–[22], [1], [23], [10].

The research defined and proposed in this paper is an attempt at a generalized approach to mapping processes onto processors. The proposed mapping is applicable to a broad class of application programs. Since the problem is NP-hard and no polynomial time algorithms exist, we have to rely on efficient heuristics. Most heuristics used for optimizing the mappings work only for restricted cases. We concentrate on developing heuristics which give optimal solutions for a wide spectrum of problems.

B. Approach

Most of the work done in the area of the cooperation of distributed resources uses one of three techniques: graph theoretic, integer programming, and heuristics. While the graph theoretic method is attractive in its simplicity, it has several limitations. First, the basic min-cut solution provides for a minimum cost allocation between two to three processors. In general, an extension of this method to an arbitrary number of processors requires an intractable n-dimensional min-cut algorithm. It does not provide mechanisms for representing resource constraints. Queueing delays introduced due to cardinality variations are also difficult to represent.

The integer programming approach formulates the model as an optimization problem, and then solves it via a mathematical programming technique. The approach is a flexible technique because it allows constraints to be introduced into the model appropriate to the application. Its shortcomings include the representation of the effects of the current system state in the real-time constraint, and the representation of the effects of precedence relations in the data flow among the subtasks. Unlike the first two approaches, heuristic approaches aim only to find a suboptimal solution. Yet, heuristic approaches are faster, more extensible, and simpler than optimal solution techniques. In fact, in some cases heuristic techniques may be the only available tools for solving difficult problems.

We use a combination of the graph theoretic, mathematical programming, and heuristic approaches. This enables us to easily represent all constraints in our model. Unlike previously proposed strategies, we formulate an accurate set of objective functions to evaluate the optimality of the solutions. We use the concept of pseudo processors, and derive conditions to prevent deadlock due to dependency relations among the subtasks. The key difference between this strategy and those proposed previously is the interdependence between the algorithm and the architecture. The knowledge from the given algorithm and the architecture guides the mapping.

The approach begins with a graphical representation of the parallel algorithm (*problem graph*) and the parallel computer (*host graph*). Using these representations, a new graphical representation (*extended host graph*) is generated onto which the problem graph is mapped. An accurate characterization of the communication overhead is used in our objective function to evaluate the mapping's optimality. An efficient mapping scheme is developed which uses two levels of optimization procedures. A combination of the graph theoretic, mathemat-

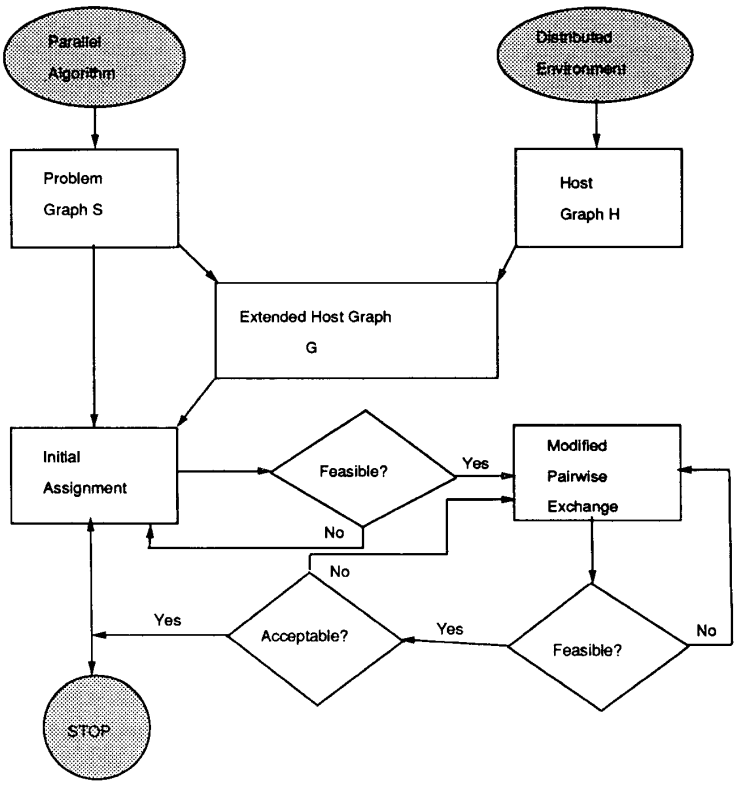


Fig. 1. Flowchart outlining the entire mapping scheme.

ical programming, and heuristic approaches is used, enabling us to easily represent numerous constraints in our model. The objective functions include minimizing the communication overhead and minimizing the total execution time, which includes both computation and communication times. The mapping scheme is tested by simulation and further confirmed by mapping a real world application onto actual distributed environments. The flowchart in Fig. 1 outlines the entire mapping scheme.

The rest of the paper is organized as follows. Section II briefly reviews previous work in related areas. The various mapping schemes, task allocation strategies, and multiprocessor scheduling techniques are grouped, depending on their proximity to each other. Section III begins with the description of the mapping model followed by properties of this model. Section IV presents a description of the objective functions and their evaluations. Section V describes the mapping strategy and its computational complexity. Section VI summarizes the performance results of the simulation and implementation of the mapping scheme. Section VII proposes a traffic scheduling scheme. The paper concludes with comments on the superiority of the proposed strategy and its extensions.

II. A Survey of Related Research

The multiprocessor scheduling problem is extremely difficult to solve and generally intractable [11]–[14], [8], [15]. Even the simplified subproblems constructed from the original scheduling problem by imposing a variety of constraints still fall in the class of NP-hard problems [24]. The difficulty of solution varies with the inclusion or exclusion of preemption, the number of parallel processors, precedence constraints, etc. Surveys of the rapidly expanding area of deterministic scheduling theory and task allocation are given in [8], [7], [5], and [25].

An efficient $O(n)$ algorithm was developed by [26] where the task processing times are equal and the task graph is tree shaped. If arbitrary precedences are allowed, then Coffman and Graham [27] proposed an $O(n^2)$ algorithm for two processors. If any of these restrictions are relaxed, the problem becomes NP-hard [11], [15]. Other scheduling problems are discussed in [24] and [19].

We now classify the various strategies for multiprocessor scheduling, task mapping, and resource allocation under a common, uniform set of terminology [5]. Broadly, the various strategies can be classified as being either static or dynamic. In the case of static scheduling, the entire information regarding the processes in the host system, as well as the processes involved in a job, is assumed to be available *a priori* [28], [29], [7], [30]. As the name suggests, dynamic scheduling is the inverse of static scheduling.

In the case that all the information regarding the state of the host system, as well as the processes, is known, an optimal assignment can be made based on some objective function [4], [31]–[34], [23]. In the event that these problems

are computationally infeasible, suboptimal solutions may be tried [35]–[38]. Within the realm of suboptimal solutions, the heuristic algorithms represent the category of static algorithms that make a realistic assumption about *a priori* knowledge concerning process and host system characteristics [16], [39]–[41]. The distinguishing feature about heuristic algorithms is their use of special parameters which affect the system in indirect ways. Usually, the parameter being monitored is correlated to the system performance in an indirect way but is easier to monitor, i.e., clustering [9], [42].

Regardless of whether a static solution is optimal or suboptimal-approximate, there are four basic categories of task mapping algorithms: Solution space enumeration and search [23], graph theoretic [3], [43], [44], mathematical programming [4], [31]–[34], and queueing theoretic [21], [45], [46].

In the case of dynamic solutions, a more realistic assumption is made that very little *a priori* knowledge is available about the resource needs of a process. Unlike the static case, no decision is made for a process before it is executed [42], [47]. Since it is the responsibility of the processes to decide where a process is to be executed, it is critical to decide where such decisions are made. Mapping schemes can be classified based on the locality of the responsibility of the process's scheduling, i.e., whether it physically resides in a single processor [29] or whether the work involved in the decision making is physically distributed among the processors [48].

Since the problem is NP-hard, several heuristic algorithms have been proposed in the past. Most of these previous approaches focused primarily on specific mapping strategies for particular multiprocessor architectures. Some strategies attempt to take advantage of hardware characteristics, such as the interconnection network of architectures. Since none of these strategies are general purpose, they apply to a limited class of multiprocessors, e.g., tightly-coupled homogeneous architectures [49], loosely-coupled homogeneous architectures [19], loosely-coupled heterogeneous architectures [19], or multicomputers connected in a point-to-point fashion [20].

Several simplifying assumptions are common. Bokhari [2] describes a mapping scheme assuming no cardinality variation. The objective function is the number of edges of the problem graph that fall on the edges of the system graph. That is, the objective function takes into account only the matched edges. However, the unmatched edges may, in some cases, determine the system's performance. The problem graph edges are also assumed to be identical, although, in general, they could have different traffic intensities, represented as weights.

Another simplifying assumption is made in the quadratic assignment problem [51]. The objective function is the sum of products of the weights of problem edges and the distances of the corresponding system edges for all problem edges, i.e., the sum of communication overheads of all problem edges, which seems to be a reasonable measure. However, this measure does not specify exactly what is to be minimized (maximized) in parallel processing applications. Moreover, the actual distance of the system edge is not really independent of the problem graph unless the problem edges share none of the system edges.

McDowell and Appelbe [52] discuss the problem of assigning processes to the processors interconnected as a ring. The problem graphs are restricted to binary trees, and a heuristic algorithm is suggested to minimize the communication delay. A tight, necessary condition for finding assignments of program fragments to linearly connected processors that require no communication delays is presented.

Sahni [53] presents the scheduling of tasks on multipipeline and multiprocessor computers. In this paper, the class of computers considered is not general. Lee and Aggarwal [10] and Bianchini and Shen [54] both assume that the number of tasks is less than the number of processors and, hence, only consider the topological variation. Further, [54] assumes that processor allocation has already been done. Berman and Snyder [1] use edge grammars to abstract graphs from the system but only consider a restricted class of interconnection structures. The mapping strategy proposed by Kim and Browne [9] uses the abstraction of the system graph without any knowledge about the problem graph, and vice versa. Furthermore, the merging of clusters resulting in the reduction of resource utilization may lead to a worse load balancing.

For brevity and completeness, we merely mention some of the other mapping schemes: [55]–[65], [47], [66]–[68], [30], [69]–[74], [21], [75]–[77], [34], [78]–[80].

From the above review, it is apparent that a myriad of multiprocessor scheduling strategies exist which can be applied to specific architectures. On the other hand, little research attempts a generalized approach to multiprocessor scheduling applicable to multiprocessors regardless of the underlying architectural characteristics.

III. The Mapping Model

We illustrate a problem graph $S = (V, E)$ by the directed graph in Fig. 2. The weights on the arcs denote the traffic intensity. The problem graph is described by a *problem matrix* PM. The element (i, j) is denoted by π_{ij}, whose magnitude is the weight for the problem edge e^{ij} and whose sign indicates the direction of the communication. If π_{ij} is positive, then the communication size is $\mid \pi_{ij} \mid$ from v^i to v^j. If π_{ij} is negative, then the communication of the same size is in the reverse direction. If there is no communication between v^i and v^j, π_{ij} is zero. The problem matrix corresponding to the problem graph of Fig. 2 is given below.

Definition A physical processor p^k corresponds to a set of *pseudo processors* $\{p_i^k \mid i \geq 1\}$. The queues in p^k are $q^{ki}, i \geq 1$, i.e., the incoming packets for p_i^k will reside in q^{ki}.

The above definition involves certain assumptions. First, it assumes that the packets are consumed eventually. Second, it assumes that a finite number of unused queues are released and allocated to other pseudo processors. Finally, it assumes that the number of queues can be less than the number of pseudo processors.

Definition: The interconnection network structure of the host architecture is described by an undirected *host graph* $H = (\mathcal{P}, \mathcal{E})$. $\mathcal{P} = \{p^1, p^2, \cdots, p^n\}$ is the set of nodes and $\mathcal{E} = \{(p^i, p^j) \mid (p^i, p^j)$ is a data link$\}$ is the set of edges.

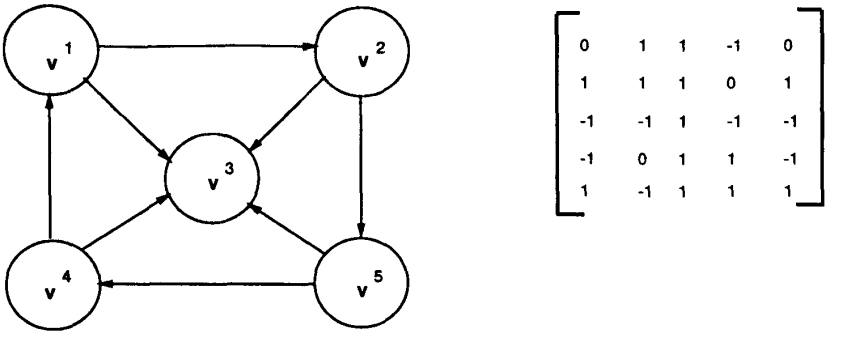


Fig. 2. The problem graph and its corresponding problem matrix.

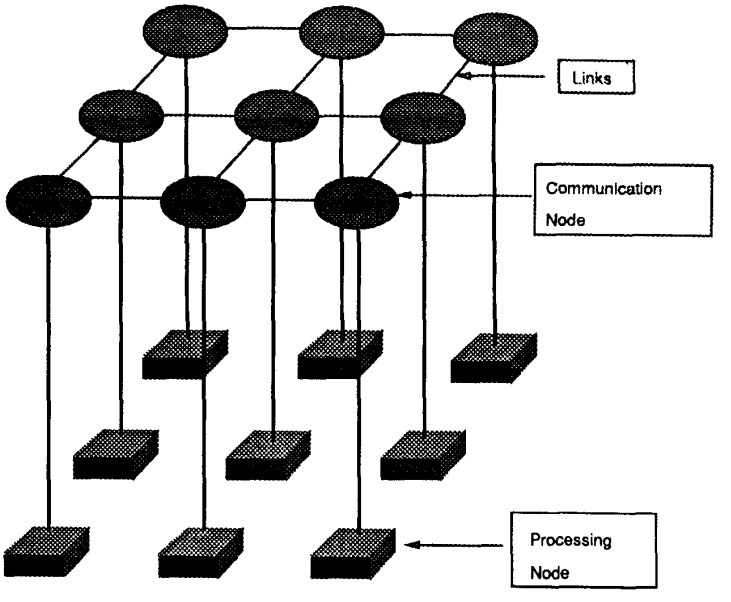


Fig. 3. An example of a host architecture: A mesh network of nine processors.

The nodes of the host graph represent the processors and the edges represent links. Each edge can be weighted on its data capacity or length. A *host matrix* HM describes the host graph. The element (i, j) is denoted by h_{ij}, whose magnitude is the weight of the host edge (p^i, p^j). Fig. 4. gives an example of a host graph and the corresponding host matrix.

Definition: An *extended host graph* $G = (\bar{P}, \bar{E})$ where

$$\bar{P} = \bigcup_{i=1}^{N} \mathcal{P}_i, N \geq 1$$

$$\bar{E} = (\bigcup_{i=1}^{N} \mathcal{E}_i) \bigcup (\bigcup_{i \neq j} \mathcal{E}_{(ij)})$$

$$\mathcal{P}_i = \{p_i^1, p_i^2, \ldots, p_i^n\}$$

$$\mathcal{E}_i = \{(p_i^l, p_i^m) \mid (p^l, p^m) \in \mathcal{E}\}$$

$$\mathcal{E}_{(ij)} = \{(p_i^l, p_j^m) \mid (p^l, p^m) \in \mathcal{E} \vee (l = m \wedge j > i)\}.$$

The nodes $\mathcal{P}_i$ of G represent the pseudo processors of H. If there is a link between processors p^l and p^m in H, then there is a directed edge from p_i^l to p_j^m in G. Note that p_i^l can communicate with p_j^m if p^l is executing the process of p_i^l and if p_j^m has not completed its execution.

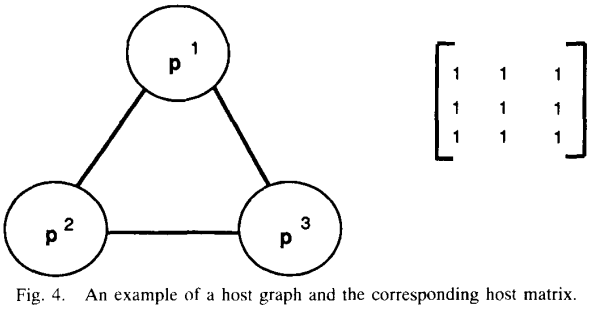


Fig. 4. An example of a host graph and the corresponding host matrix.

An *extended matrix* EM describes the extended host graph. The element (i, j) is denoted by m_{ij}, whose magnitude is the weight of the extended host edge. Note that we can have numerous extended host graphs (matrices) depending

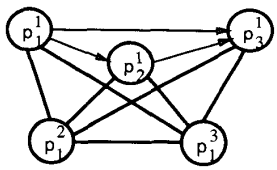


Fig. 5. An example of an extended host graph and the corresponding extended matrix.

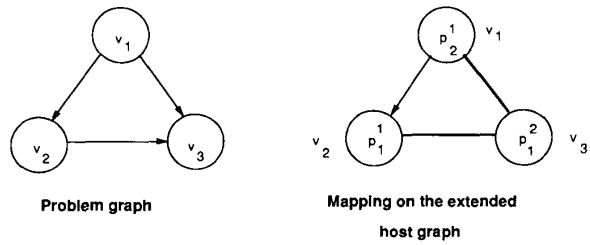


Fig. 6. An example of a mapping leading to processor deadlock.

on the particular problem and host graphs. Fig. 5 shows an extended host graph and the corresponding extended matrix, corresponding to the problem graph of Fig. 2 and the host graph of Fig. 4.

A. Properties of the Model

The mapping of a parallel algorithm on a parallel architecture can be described by a one-to-one mapping M from the vertices of a problem graph S onto the nodes of the extended host graph G [2].

Given M, if the allocated pseudo processors $\bigcup_{i=1}^{N} p_i^k$ for p^k are ordered as $p_{l_1}^k, p_{l_2}^k, \cdots, p_{l_m}^k$ where $l_r < l_{r+1}$, $1 \leq r < m$, then the processes associated with these pseudo processors will be executed by p^k in that order, i.e., the lower subscript has a higher priority. Hence, a directed edge from p_i^k to p_j^k in G will force $i < j$ and thus p_i^k can communicate with p_j^k but not vice versa.

Consider the case when the host graph has $\mid H \mid$ processors and the problem graph has $\mid S \mid$ processes. Depending on the values of $\mid H \mid$ and $\mid S \mid$, we will either have one extended host graph or several extended host graphs.

- If $\mid H \mid \geq \mid S \mid$, then every host processor is itself a pseudo processor, i.e., we have only one extended host graph.
- If $\mid H \mid < \mid S \mid$, then at least one processor will have more than one pseudo processor, i.e., there will be more than one extended host graph. The next proposition derives an expression to evaluate the number of these extended host graphs.

Proposition 1: Given d processors and $d+n$ processes, the number of ways in which the processes can be allocated to the processors such that each processor has at least one process is

$$\frac{(n+d-1)!}{n!(d-1)!}.$$

In order to cut down on this combinatorial explosion of the number of extended host graphs, we restrict the extended host graph by imposing certain constraints. We split the processors into pseudo-processors in the ratio of their computational power.

In the next section we deal with the feasiblility of mappings introduced due to the priorities assigned to pseudo-processors.

Definition: A mapping is *feasible* if the parallel algorithm executed on the parallel machine according to this mapping terminates.

1) Processor Deadlock: We make certain assumptions about the host architecture which do not restrict the generality of the mapping scheme described in this paper. First, we assume that each processor is capable of computing its task. Second, we assume that the interconnection between processors is loss free, i.e., no packets are lost in communication. Thus, a pseudo processor will wait for any packets sent from another pseudo processor and will eventually receive them.

It can be easily shown that not all mappings of S onto H will yield a feasible mapping. If there is a limit on the number of queues per processor, then merely sending packets numbering more than the number of queues to any particular processor will result in a deadlock (q-deadlock). But, a deadlock could result even if we do not constrain the number of queues. Fig. 6 illustrates this by an example. Consider a host with two processors. The problem graph S has three vertices. To take care of the cardinality problem, one of the processors will correspond to two pseudo processors. The host graph H, with these three pseudo processors, is then constructed. If we construct a mapping with p_2^1 onto v^1, p_1^1 onto v^2, and p_1^2 onto v^3, then it is clear that this will lead to a deadlock. The p-deadlock is due to the violation of the data dependence presented by the problem graph.

We shall now formally derive a condition that will guarantee a p-deadlock free mapping. We say that the process v^k *depends* on v^l if there is a path from v^l to v^k. Also, p_t^s is *busy* if p^s is executing the process of p_t^s.

Lemma 1: A mapping is p-deadlock free iff there is no such list of S-nodes $v^{k_1}, v^{k_2}, \cdots, v^{k_m}, v^{l_1}, v^{l_2}, \cdots, v^{l_m}$ mapped to pseudo processors $p_{t_1}^{s_1}, p_{t_2}^{s_2}, \cdots, p_{t_m}^{s_m}, p_{r_2}^{s_2}, p_{r_3}^{s_3}, \cdots, p_{r_{[m+1,m]}}^{s_{[m+1,m]}}$, respectively, such that v^{k_j} depends on v^{l_j} and $t_j > r_j$, $1 \leq j \leq m$, where $[i,m] \equiv (i) \bmod m$.

Proof: First we assume that the mapping is not p-deadlock free. Hence, a p-deadlock occurs. Thus, for all busy pseudo processors $p_{t_j}^{s_j}$, we can find $p_{t_{j+1}}^{s_{j+1}}$ such that v^{k_j} of $p_{t_j}^{s_j}$ depends on v^{l_j} of $p_{t_{j+1}}^{r_{j+1}}$ and $p_{t_{j+1}}^{r_{j+1}}$ cannot be allocated to the processor occupied by $p_{t_{j+1}}^{s_{j+1}}$, i.e., $t_{j+1} > r_{j+1}$. Next, we check whether the processor $p^{s_{j+1}}, j \geq 1$ is occupied by $p_{t_x}^{s_x}$, $1 \leq x \leq j$. Since the number of busy processors are finite, we can find such a pseudo processor $p_{t_x}^{s_x}$. Thus, the list of S-nodes v^{k_x}, $v^{k_{x+1}}, \cdots$, $v^{k_{j+1}}$, v^{l_x}, $v^{l_{x+1}}$, $\cdots$, $v^{l_{j+1}}$ contradicts the condition of Lemma 1.

We now assume that such a list of S-nodes exists. Since the pseudo processors $p_{r_2}^{s_2}, p_{r_3}^{s_3}, \cdots, p_{r_{[m+1,m]}}^{s_{[m+1,m]}}$ cannot execute their processes, $p_{t_1}^{s_1}$, $p_{t_2}^{s_2}, \cdots$, $p_{t_m}^{s_m}$ cannot receive the packets. Hence, these pseudo processors will be busy, implying that a p-deadlock will occur. □

Consider the special case when $m = 1$, i.e., there are only two processes. If process v^k depends on v^l and these are mapped to pseudo processors p_i^s and p_j^s, respectively, then $i > j$, i.e., process v^l is to execute before v^k to prevent a p-deadlock. Note that a p-deadlock occurs because of the priority assigned to the pseudo-processors, i.e., they need to execute sequentially within a processor. Hence, we are allocating and scheduling processes simultaneously.

Processes v^k and v^l are *strongly connected* if there is a path from v^k to v^l and from v^l to v^k. In the problem graph S, the S-nodes can be partitioned into disjoint Strongly Connected Components (SCC) [81].

Once we have derived the conditions to check if a mapping is p-deadlock free, the next question is whether such a mapping exists. If so, then given the problem graph and the host graph, is it possible to say that a p-deadlock free mapping exists? The necessary and sufficient condition to answer the above question is stated next as a theorem.

Theorem 1: Given a problem graph S, there is a p-deadlock free mapping for a host graph H iff the number of processors in H is at least equal to the maximum number of nodes in an SCC of S.

Proof: Suppose that a mapping exists such that the number of pseudo processors in H is greater than or equal to the number of nodes in an SCC of S. Order the SCC's C_1,C_2,$\cdots$,C_n such that if a node in C_i depends on a node in C_j, then $i < j$. Assigning processes in C_x to pseudo processors in the set $\bigcup \mathcal{P}_x$ satisfies the condition of Lemma 1. Hence, this mapping is p-deadlock free.

Now assume that the number of processors in H is less than the maximum number of nodes in an SCC of S. Hence, for every mapping we have to assign two nodes of the SCC to some p^x. Since these two nodes depend on each other, we can always find two nodes v^k and v^l mapped to p_i^x and p_j^x such that v^k depends on v^l and $i > j$. Thus, the corresponding mapping is not p-deadlock free. □

2) Queue Deadlock: As the previous subsection states, limiting the number of queues can lead to a deadlock. We shall now derive an expression for the minimum number of queues required to avoid a queue deadlock (q-deadlock). We first illustrate with an example in which the q-deadlock is induced by a particular mapping.

Example: Consider the case when the number of S-nodes is equal to the number of H-nodes and when there are two SCC's, C_1 and C_2 in S. One node in C_1 depends on a node in C_2, and each processor in H has only one queue. If we have a mapping (maybe p-deadlock free) in which any processor is assigned with two processes, v^k and v^l, the mapping is not q-deadlock free. This happens when the process v^k sends a packet to process v^l (since the pseudo-processors share the queues of the processor in which they reside).

Theorem 2: Given a problem graph S, if the number of queues in each processor of the host architecture H is greater than or equal to the number n of strongly connected components in S, then a p-deadlock free mapping is also q-deadlock free.

Proof: If the mapping is p-deadlock free, then any two nodes in an SCC must be assigned to distinct processors (Lemma 1). Hence, each SCC will have at most one process assigned to each processor. Thus, each processor will be allocated at most n processes for any p-deadlock free mapping. □

There are some interesting consequences of the above theorem. If S has only one SCC, then S is itself strongly connected. Hence, each processor requires only one queue to support any p-deadlock free mapping. If the number of SCC's, n, is equal to the number of S-nodes and the S-nodes form a path, then by Theorem 2, n queues are required for q-deadlock free mapping. But, a q-deadlock free mapping for such an S can be obtained by a maximum of two queues. Hence, this gives us a better bound on the maximum number of queues required for any p-deadlock free mapping to be q-deadlock free.

A procedure to evaluate a better bound on the maximum number of queues required for any p-deadlock free mapping to be q-deadlock free is proposed. It is not the best bound though. For a given problem graph S, the procedure Q_min() calculates a better bound than Theorem 2.

Definition: The maximum independent strongly connected component set $MISC(S)$ in a directed graph S is a maximum set of strongly connected components of S, i.e.,

$$MISC(S) = \{C_i \mid 1 \leq i \leq n\}$$

such that for any node v^k, if $v^k \in C_i$, then no node in C_j, $j \neq i$, depends on v^k. The graph $S^{'}$ is defined as a subgraph obtained from S by removing all the nodes in $MISC(S)$.

Q_min (S: problem graph): Integer;
begin
  0 **Q_min**= $\{\mid MISC(S) \mid +$**Q_recurse**$(S^{'})\}$;
end

Q_recurse (S: problem graph): Integer;
begin
  0 **If** $S^{'} = \phi$ **then Q_recurse** $=\mid MISC(S) \mid$;
  1 **If** the cardinality of an SCC in $MISC(S) > 1$,
    then Q_recurse $= (\mid MISC(S) \mid +$**Q_recurse**$(S^{'}))$;
    else Q_recurse $= (\mid MISC(S) \mid +$**Q_recurse**$(S^{'}) - 1)$;
end

Example: Consider the problem graph S as shown in Fig. 7(a). The maximum SCC set of S is $\{\{n_1\},\{n_2\}\}$. Fig. 7(b) shows the maximum independent SCC set of $S^{'}$, i.e., $\{\{n_3\}\}$. Fig. 7(c) shows the maximum independent SCC set of $S^{''}$, i.e., $\{\{n_4\},\{n_5\}\}$. Fig. 7(d) shows the maximum independent SCC set of $S^{'''}$, i.e., $\{\{n_6\}\}$. Since Q_recurse($S^{'''}$) returns 1, Q_recurse($S^{''}$) returns 2, and Q_recurse($S^{'}$) returns 2, the procedure Q_min(S) returns 4. This bound on the number of queues necessary in each node for any p-deadlock free mapping to be q-deadlock free and derived using the procedure Q_min() is lower than the bound derived using Theorem 2. It can be easily seen that the bound derived using the theorem is 6.

The correctness of the procedures Q_min and Q_recurse can be easily established. If a node in the maximum SCC set of $S^{'}$ is assigned to the processor allocated to some process v^k, then

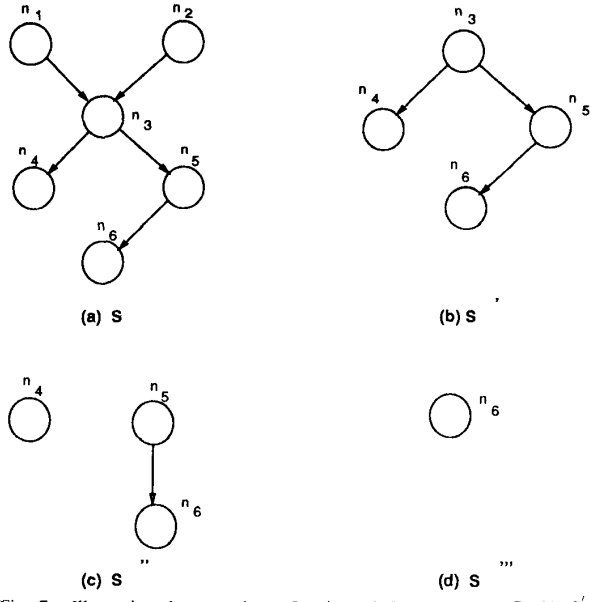


Fig. 7. Illustrating the procedures Q_min and Q_recurse. (a) S, (b) S', (c) S'', and (d) S'''.

the recently assigned process can be executed and can send messages to other processes only after v^k is completed. Since a process is assigned a queue only after receiving a message, the worst case occurs when one of the nodes v^k in $MISC(S')$ is assigned to a processor that has not been assigned any process; and the node starts executing by sending a message to all the nodes in the maximum SCC set of S''. If any SCC, C_i in $MISC(S')$ has more than one node, then v^k may be in this component and the other nodes in $MISC(S')$ can be assigned to the processors p^k that have been assigned the maximum number of processes among all processors; otherwise, the only node in C_i cannot be used to increase the queue requirement of the processor p^k, as done in step 2 of the procedure Q_recurse.

B. Constraints in a Distributed Environment

In a general distributed computing environment, processes will exist that cannot execute on certain processors, two processes need to execute on the same processor, or a process might need more than one processor at the same time to execute. Thus, the additional constraints may be incorporated in the mapping model:

- We define a binary *exclusion matrix* EM that describes the computability of a particular task on a particular processor. If the element (i, j) is zero, then the process i cannot execute on processor j. Otherwise, the value of the element (i, j) is 1. This restriction could result from several reasons. The obvious reasons include 1) the type of computation required, and 2) the amount of memory required.
- We define an *interference matrix* IM that describes the degree of incompatibility between two processes. The value of the element (i, j) ranges between 0 and 1. The closer the value is to zero, the more incompatible are the two processes. As an example, a pair of tasks that are both highly CPU bound will have more interference costs than a pair in which one is CPU bound and the other is IO bound. Adding interference costs should increase the concurrency between the processes.
- We define a *simultaneous matrix* SM that describes the processors needed by a process for its execution. The processors needed simultaneously for the execution of a process i are found by checking the columns of the matrix SM corresponding to row i, i.e., (i, j), where j ranges for all the processors.

IV. Objective Functions

The mapping problem essentially involves two distinct procedures. First, we formulate an objective function to accurately measure what we need to optimize. Then, we propose a mapping scheme to optimize the formulated objective function. We now discuss the various objective functions and their evaluations.

A. Communication Overheads

Minimizing the communication overhead is especially important when mapping parallel algorithms onto parallel computers [10], [2]. Chaudhary and Aggarwal [82]–[84] have shown that the communicational complexity of certain vision algorithms dominates the computational complexity for most distributed memory architectures. In other words, the communication overhead increases at a rate higher than the rate of decrease in the computation time. This leads to a decrease in processor efficiency [85] and throughput with an increase in the number of processors. We now propose an objective function that minimizes the communication overhead, with a constraint that the number of processors may be less than the number of processes.

If the number of processes in S is more than the number of processors in H, then by the Pigeonhole principle [86], a processor is assigned more than one process. The processor is split into pseudo-processors such that each process is assigned to one pseudo-processor. In fact, more than one processor may be split into several pseudo-processors, each with a process assigned to it. Since any two pseudo-processors p_i^k and p_j^k share the same communication node, we assume that the communication overhead for these pseudo-processors is zero. Hence, we can take advantage of the above to reduce the communication overhead by mapping pseudo-processors requiring much communication among them onto the same processor.

We use the terms defined by Lee and Aggarwal [10]. A *stage* is a time interval during which the computation for a process is carried out. A *phase* is a time interval during which the communication for a problem edge is carried out. A *step* is a time interval for the communication through a link. A phase, in general, may be comprised of a set of steps if it takes several links to realize a problem edge.

Due to precedence relationships, certain processes can execute concurrently while others cannot. Accordingly, certain communications can occur in the same phase while others

need distinct phases. Hence, the set of problem edges E can be sorted into subsets E_k according to the phase in which they are required, shown below:

$$E_k = \{\bar{e}_{kj}\}, 1 \leq k \leq N_p, 1 \leq j \leq L_k$$

where N_p is the number of phases (subsets) and L_k is the number of problem edges in the subset E_k. $\bar{e}_{kj}$ is to be distinguished from the problem edge e^{kj}. A problem edge might appear in more than one such subset, each with a different weight associated with it. Thus, the messages required by the process v^i can only be generated and sent out by another process v^k in different phases. Also, two problem edges from different subsets may share links in the system without increasing the communication overhead, since they do not need the links simultaneously. Thus, they can be treated independently when being mapped onto the host edges. Due to cardinality variation, the computation and communication of messages of some of the processes has to be delayed. Hence, the subsets E_k are determined by the given problem graph and the particular mapping.

Before defining the objective functions related to the communication overhead, let us examine the communication overhead of a problem edge in more detail. First, the frequency of use of a problem edge indicates the traffic intensity along the problem edge. The weight assigned to the problem edge is proportional to the traffic intensity. Second, the *nominal distance* between the host nodes, i.e., the shortest distance between the host nodes, cannot be used directly. The nominal distance D_{il} is the length of the shortest path between the host nodes, p^i and p^l. If $(p^i, p^l) \in \mathcal{E}$, i.e., there is an edge between p^i and p^l, then the nominal distance D_{il} is one or else it is equal to the number of host edges in the shortest path between p^i and p^l. Consider a case involving more than one problem edge mapped onto a single host edge. Then, the communication of some of the problem edges will be delayed unless the host edge comprises an adequate number of multiple links. Thus, the communication overhead of the problem edge e^{kj} may be distinct from the nominal distance D_{il} when the problem edge e^{kj} is mapped onto the host edge (p^i, p^l). Hence, given the problem and host graphs, the communication overhead c_{kj} of the problem edge (p^i, p^l) depends on the particular mapping and the communication control parameters of the host system. To obtain an accurate characterization of the communication overhead, the actual communication overhead of the problem edge should be used instead of the nominal distance.

B. Minimizing the Schedule Length

In a heterogeneous computing environment where the computing power of the individual processors can vary tremendously, the computation time of the processes on various processors will greatly influence the task performance. A very common objective function is to minimize the schedule length (or makespan) defined as the sum of all computations and communications between the processes.

The computation time of every process on each processor is represented by a *computation matrix comp*. Depending on the particular mapping x, the computation time of the process i can be easily found from the computation matrix *comp*, i.e., if the mapping x maps the process i onto processor j, then the computation time is $comp(i, j)$. This computation time is also represented as $comp_x(i)$. The communication time between processes i and j is computed as described in the previous section.

C. The Objective Functions

We define an objective function based on the communication overhead $\bar{c}_{kj}$ of the problem edge $\bar{e}_{kj}$ for a mapping D as follows:

$$OF_1 = \sum_{k=1}^{N_e} \left(\max_{j=1}^{L_k} \bar{c}_{kj} \right).$$

In OF_1, some problem edges in the same subset E_k are required simultaneously but the subsets are required in a sequence, i.e., $N_e > 1$ and $L_k > 1$ for some k. There are two special cases of OF_1.

First, consider the case when $L_k = 1, \forall k$. This amounts to sequential processing since no two problem edges are processed in the same phase, i.e., none of the problem edges are processed simultaneously. The objective function OF_1^a for this special case can be represented as follows:

$$OF_1^a = \sum_{k=1}^{N_e} \bar{c}_{k1}.$$

Second, consider the case when $N_e = 1$, i.e., all the problem edges are processed simultaneously. This implies total parallelism. The objective function OF_1^b for this special case can be represented as follows:

$$OF_1^b = \max_{j=1}^{L_1} \bar{c}_{1j}.$$

It is easy to see that OF_1^b may decrease while OF_1^a increases.

Before we compute the objective functions, we define certain matrix representations used. A *nominal distance matrix* D represents the nominal distances in a host graph. The element (i, j) is the nominal distance D_{ij} for the host edge (p^i, p^j). An *extended nominal distance matrix* D' represents the nominal distance in an extended host graph. Note that the nominal distances for two pseudo processors in the same host processor are the same. Figs. 8(c) and (d) show an extended host graph and its corresponding extended nominal distance matrix.

An *assignment matrix* A describes a particular mapping. This matrix is obtained by permuting the columns and rows of the problem matrix according to the mapping. For example, performing the mapping indicates that the second and third columns and the second and third rows are exchanged. Fig. 8(e) shows the resulting assignment matrix.

A *communication overhead matrix* CM is obtained from the nominal distance and assignment matrices. The element (i, j) indicates the communication overhead (denoted by c_{ij}) of the problem edge e^{ij} for a particular mapping. For a problem edge

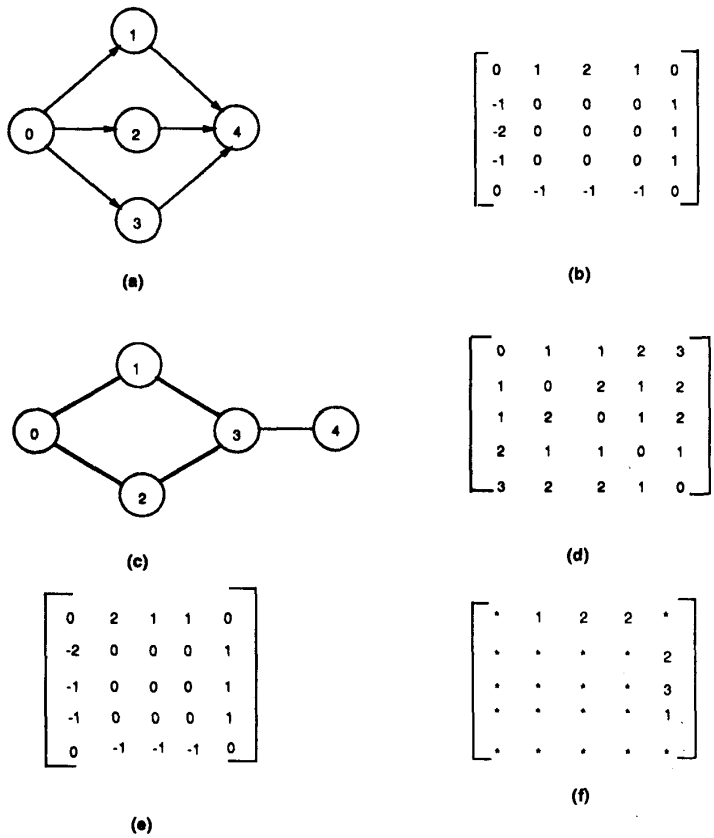


Fig. 8. Calculating the communication overhead. (a) The problem graph, (b) the problem matrix, (c) the extended host graph, (d) the extended nominal distance matrix, (e) the assignment matrix, and (f) the communication overhead. The *s indicate that the corresponding problem edges are not used.

with a negative π_{ij}, c_{ij} is undefined since it is redundant. Fig. 8(f) gives the communication overhead matrix for the mapping (1 2), where the star indicates that the corresponding problem edge is not used. The following sections describe in detail the procedures to evaluate c_{ij}.

The objective function for minimizing the schedule length is given as follows:

$$OF_2 = \min_x \left(\sum_i comp_x(i) + \sum_i \sum_{j \neq i} c_{ij} \right).$$

In OF_2, the first term is the summation of the computation times of the processes and the second term is the summation of the communication times between these processes. The objective is to find a mapping x that minimizes this sum. The next section describes the evaluation of the communication overhead.

D. Computing the Communication Overhead

The communication mode can be classified into two classes: synchronous and asynchronous. In a synchronous mode of communication, the steps for all the links are synchronized; whereas in an asynchronous mode of communication, communication for a problem edge can occur at any time.

1) Synchronous Communication: We calculate the communication overhead $\bar{c}_{kj}$ by the following procedure:

OF_Calculate_sync (S: problem graph; D: mapping);

begin

0 Sort E into subsets E_k according to the phases when the problem edges are used.

1 **For** each E_k **do**

1.1 $\bar{E}_k = E_k$. Assume that a problem edge $\bar{e}_{kj}$ is mapped onto a path between pseudo processors p_l^r and p_m^s.

1.2 **For** each problem edge $\bar{e}_{kj}$ **do**

1.2.1 **If** $r = s$ **then** $\bar{c}_{kj} = 0$ and delete $\bar{e}_{kj}$ from $\bar{E}_k$.

od

1.3 **For** each problem edge $\bar{e}_{kj}$ in E_k **do**

1.3.1 Find the nominal distance D'_{rs} of the corresponding host edge (p^r, p^s) by superimposing A onto D'.

1.3.2 **If** the length of the path in the host graph H is greater than 1, **then** the path in the host graph is divided into a sequence of links.

od

1.4 **For** each problem edge $\bar{e}_{kj}$ in E_k **do**

1.4.1 Assign the packets to p_l^s such that the number of packets is equal to the weight of the edge $\bar{e}_{kj}$.

1.4.2 Assign the data structures SOURCE, DESTINATION, and STEP in each packet with p_l^s, p_m^r, and 0, respectively.

od

1.5 **Repeat**

1.5.1 STEP := STEP + 1

1.5.2 Move packet(s) to the next node according to the implementation of the particular system.

1.5.3 **If** a packet reaches its destination, **then** Remove the packet.

If it is the last packet from the source, **then** $\bar{e}_{kj}$ = STEP of the last packet from p_l^s to p_m^r

1.5.4 **else** add it to the queue of the current node.

Until all the packets reach their destination.

2 Calculate the OF.

end

The decomposition of the problem edge ([1.3.2]) is determined *a priori* by the host's routing parameters. The choice of the packets to be moved ([1.5]) is determined by the routing scheme of the particular host system. Finally, the number of packets ([1.4.1]) and the increment ([1.5.1]) may be adjusted according to the particular system.

The computational complexity of the procedure is dominated by loop [1.5]. The number of iterations of this loop varies with the assignment and the routing rules. The number is bounded by $\mid V \mid D_m^{'}\pi_m$ computations where $\mid V \mid$ is the number of problem nodes, $D_m^{'}$ is the diameter of the extended host graph, i.e., the maximal extended nominal distance, and π_m is the maximum weight in a problem.

2) Asynchronous Communication: The procedure, OF_Calculate_sync, used to evaluate the communication overhead for the synchronous mode of communication may not give a precise evaluation of the communication overhead for an asynchronous mode of communication. We present another procedure, OF_Calculate_async(), which approximates the communication overhead for an asynchronous mode of communication.

OF_Calculate_async (S: problem graph; D: mapping);

begin

0 Sort E into subsets E_k according to the phases when the problem edges are used.

1 **For** each E_k **do**

1.1 $\bar{E}_k = E_k$. Assume that a problem edge $\bar{e}_{kj}$ is mapped onto a path between processors p_l^r and p_m^s.

1.2 **For** each problem edge $\bar{e}_{kj}$ **do**

1.2.1 **If** $r = s$ **then** $\bar{e}_{kj} = 0$ and delete $\bar{e}_{kj}$ from $\bar{E}_k$.

od

1.3 **For** each problem edge $\bar{e}_{kj}$ **do**

1.3.1 Find the nominal distance $D_{rs}^{'}$ of the corresponding host edge (p^r, p^s) by superimposing A onto $D^{'}$.

1.3.2 The communication overhead $\bar{e}_{kj} = \pi_{kj} D_{rs}^{'}$.

od

2 Calculate the OF.

end

The sorting of the problem edges in the procedure OF_Calculate_async is carried out only when some problem edges are distinguishable in time from others and hence can be partitioned into different sets. Otherwise, the problem edges will all belong to just one set. Thus, the computational complexity of the procedure is bounded by [1.3], which is executed $\mid V \mid$ times.

The procedure OF_Calculate_sync gives a very accurate evaluation of the communication overhead but is very expensive to evaluate. Since step [1.5] in OF_Calculate_sync simulates the routing on the target host, it is unsuitable for very large applications. In contrast, the procedure OF_Calculate_async is easy to evaluate but is not as accurate as OF_Calculate_sync. Thus, one can also use OF_Calculate_async to evaluate the communication overhead for a synchronized mode of communication when the priority is the time for evaluation.

V. Mapping Scheme

Once the objective function for a particular mapping is available, it must be optimized. This optimization of the mapping from S to H involves two steps. We first try to evaluate a good initial assignment, and then improve the mapping incrementally by a modified pairwise exchange. This approach leads more efficiently to an optimal solution, although it does not guarantee it. First, evaluation of a good initial assignment can save a great deal of computation in the optimization performed later. Since we have a reasonably good initial assignment, we could find an optimal solution without going through an exhaustive pairwise exchange (which requires an exponential order of computation).

A. Initial Assignment

We present a procedure which attempts to achieve a very small value of the objective function rather than to rely on a random initial assignment. If the mapping obtained by this algorithm is not optimal or does not satisfy certain constraints, we use the modified pairwise exchange method described later. We define the communication intensity of v^k to be the sum of the weights of all the edges incident on it. The degree of a node v is denoted by $d(v)$. The choice of the S-node to be assigned is based on the communication intensity and connectivity of S whereas the choice of the corresponding H-node is determined by a certain measure derived from the objective function.

Init_assign (S: problem graph; H: host graph);

begin

0 $V = \{v^k \mid v^k \in S\}$

1 Assign tasks using the exclusion matrix EM.

2 Check for the interference matrix IM and the simultaneous matrix SM.

3 Make the extended host graph G.

4 Find the process v^k with the largest communication intensity from V. In case of a tie, choose one arbitrarily.

5 If there are n SCC's in S whose nodes are such that v^k *depends* on them, then assign v^k to p^i_{n+1} such that $d(p^i)$ is as close to $d(v^k)$ as possible.

6 $V = V - v^k$.

7 **While** $\mid V \mid > 0$ **do**

7.1 Find v^l with the largest communication intensity from the nodes adjacent to S-nodes which are already assigned.

7.2 Assign v^l to p^j_m such that a certain measure of the objective function is minimized and the assignment is a *feasible* mapping.

7.3 V = $V - v^l$.

od

end

The time complexity of the above procedure is determined by subprocedure 7.2, which is repeated $n - 1$ times. This subprocedure has a complexity of $O(n^3)$ due to the procedure to check the feasibility of a particular mapping. Hence, the execution of Init_assign takes $O(n^4)$ time.

B. Modified Pairwise Exchange

The procedure Init_assign does not guarantee an optimal solution. Hence, we may use the pairwise exchange of two problem nodes to improve the mapping. This pairwise exchange could be done iteratively until we exhaust all the possibilities. Unfortunately, the method is exponential in computational complexity. Instead, we propose a modified pairwise exchange scheme in which we consider only a selected set until a certain criteria is satisfied. This technique is expected to give a reasonable solution efficiently since the initial assignment scheme reduces the search space considerably. But, there is still a possibility that the solution is a local optima. One may use another distinct initial assignment to get out of this local optima if the solution is unsatisfactory. The algorithm for the whole mapping, including the modified pairwise exchange, is given below.

Mod_pair_xchange (S: problem graph; H: host graph);

begin

0 Init_assign(S,H);

1 Evaluate objective function.

2 **If** mapping is satisfactory **then goto end**.

3 **Repeat**

3.1 find v^x for the exchange according to certain measures derived from the objective function used.

3.2 $\forall v^y \in \{V - v^x\}$ **do**

3.2.1 exchange(v^x, v^y) temporarily

3.2.2 **If** Is_feasible($\tilde{S}$,H) **then** evaluate the objective function.

od

3.3 Determine v^k which gives the best objective function.

3.4 **If** the new objective function is better than the old objective function **then** exchange(v^x, v^k).

Until ¬satisfactory.

end

The complexity of the above algorithm is given by $O(M * N * F)$ where M is the number of iterations of the **while** loop. Step 4.2 takes $O(N)$ time and evaluating the OF takes $O(F)$ time.

C. Feasibility

This section proposes a procedure, Is_feasible, to check the feasibility of a mapping and to prove its correctness.

Is_feasible (S: problem graph; G: extended host graph): Boolean;

begin

0 Construct $\tilde{G} = (P, \tilde{E})$. For any two processes v^k and v^l which are mapped to p^r_i and p^s_j, **if** there is an edge from v^k to v^l in S (or $r = s \wedge i < j$) **then** there is an edge $(p^s_i, p^r_j) \in \tilde{E}$.

1 Compute the adjacency matrix $\tilde{G}_a$ of $\tilde{G}$.

2 Compute the transitive closure $\tilde{G}^c_a$ of $\tilde{G}_a$.

3 $\forall p^t$, **if** there is an edge $(p^s_i, p^r_j) \in \tilde{E}$ and if the item of $\tilde{G}^c_a$ in row p^t_j and column p^t_i has a path from p^t_j to p^t_i in $\tilde{G}$ **then Is_feasible** = false. **Goto end**.

4 **Is_feasible** = true.

end

The complexity of Is_feasible is $O(n^3)$, which is essentially the complexity of computing the transitive closure in step 3. The correctness of Is_feasible can be easily shown using the previous lemmas and theorems.

1) The Correctness of Feasibility: By Lemma 1, if a mapping is not p-deadlock free, we can find a list of S-nodes v^{k_1}, $v^{k_2}, \cdots, v^{k_m}, v^{l_1}, v^{l_2}, \cdots, v^{l_m}$ mapped to $p^{s_1}_{t_1}, p^{s_2}_{t_2}, \cdots, p^{s_m}_{t_m}, p^{s_2}_{r_2}, p^{s_3}_{r_3}, \cdots, p^{s_{[m+1,m]}}_{r_{[m+1,m]}}$, respectively, such that v^{k_j} *depends* on v^{l_j} and $t_j \geq r_j, 1 \leq j \leq m$. Hence, we can find a cycle $p^{s_{[m+1,m]}}_{r_{[m+1,m]}}, p^{s_1}_{t_1}, p^{s_2}_{r_2}, p^{s_2}_{t_2}, p^{s_3}_{r_3}, \cdots, p^{s_m}_{t_m}, p^{s_{[m+1,m]}}_{r_{[m+1,m]}}$ in $\tilde{G}$ and thus step 4 returns Is_feasible = false.

Conversely, if step 4 returns Is_feasible = false, then a cycle K in $\tilde{G}$ exists. A series of continuous nodes in K, $p^{i_1}_{j_1}, p^{i_2}_{j_2}, \cdots, p^{i_n}_{j_n}$ is a *sequence* if $i_1 = i_2 = \cdots = i_n$ $j_1 < j_2, < \cdots < j_n$ and if it is maximal. In the above sequence, $p^{i_1}_{j_1}$ is the *initial* node and $p^{i_n}_{j_n}$ is the *terminal* node. K can be expressed as a series of sequences K_1, K_2, $\cdots$, K_m whose initial nodes are $p^{s_1}_{t_1}, p^{s_2}_{t_2}, \cdots, p^{s_m}_{t_m}$ and whose terminal nodes are $p^{s_m}_{r_m}, p^{s_1}_{r_1}, \cdots, p^{s_{m-1}}_{r_{m-1}}$. This list of pseudo processors forming K violates Lemma 1. Thus, the mapping is infeasible. □

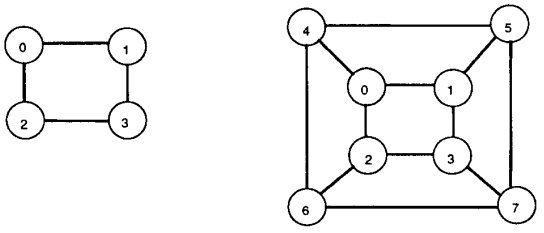


Fig. 9. 4-node and 8-node hypercubes as host graphs.

VI. Implementation Results

The mapping scheme was tested using several synthetic problem graphs and host graphs, and also on a real-time stereo in a distributed environment. The synthetic examples were a simulation of real situation problems and hosts. The next two sections summarize the results of both these implementations.

A. Simulation

To compare our strategy with previous strategies we applied the mapping scheme on the problem graphs used by Lee and Aggarwal [10]. We used 4-node and 8-node hypercubes to introduce the cardinality variation which is not dealt in [10]. Fig. 9 illustrates the host graphs used for the simulation. The use of hypercubes trivializes the evaluation of the shortest path in the host graph. We evaluate the shortest path by complementing the less significant bit first. It is also assumed that each link of the host can independently send or receive messages.

The problem graphs considered [10] are those that would appear in real applications. A synchronous mode of communication is assumed. The problem graphs for the 4-node and 8-node host graphs are given in Fig. 10 and Fig. 11, respectively. The weight for each edge is given and the number within the parenthesis indicates the phase in which the edge is required. The computational requirement at each node in the problem graph is the same, and so is the computability of each node in the host graph. Thus, these examples do not fully exploit the generality of the mapping scheme but are used for comparison.

The examples considered here do not have any constraints involved. Furthermore, the uniformity of the computational requirement of the problem nodes and the computability of the host nodes leads to the splitting of each host processor into two pseudo-processors. Thus, the modified pairwise exchange involves only the changes of assignments of the processes onto the pseudo-processors. There is no change in the extended host graph. Fig. 12 gives an example of the extended host graph for the first set problem nodes and the 4-node host graph.

The mappings were tested with the objective function as OF_1. The implementations exactly follow the procedures explained earlier. The results are tabulated in Tables I and II for the 4-node host and in Tables III and IV for the 8-node host. The number of iterations in the modified pairwise exchange procedure is represented by N. The results obtained indicate that the proposed mapping strategy takes care of both the cardinality variation and the topological variation. The objective functions and the number of iterations in the modified pairwise exchange indicate that our strategy works well and gives near optimal results (if not optimal in most cases).

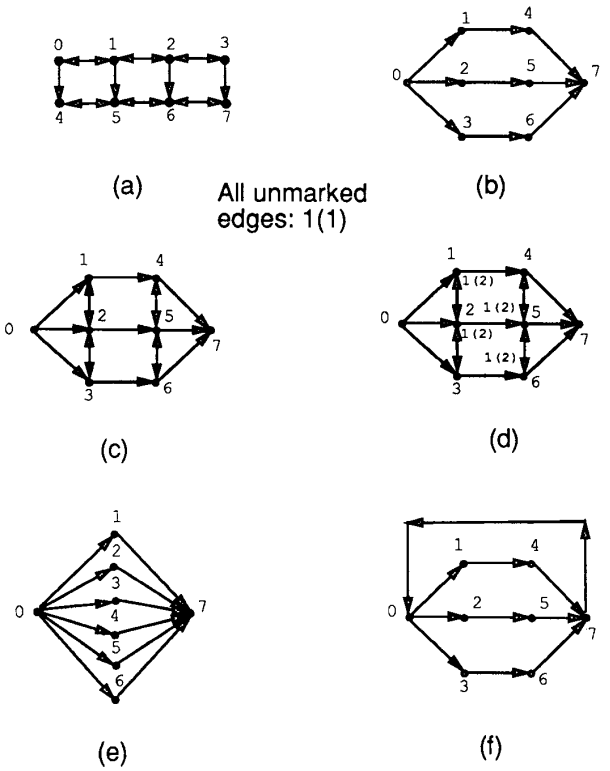


Fig. 10. The problem graphs for the 4-node hosts: (a) S_1, (b) S_2, (c) S_3, (d) S_4 (e) S_5, (f) S_6.

B. Real-Time Stereo

To further evaluate the mapping scheme, a significantly large problem was considered to be mapped in a distributed environment with several constraints. The problem was to determine the three-dimensional position of object points in a scene using a stereo algorithm [87].

The stereo algorithm was split into processes as shown in Fig. 13. This splitting into processes was done arbitrarily. However, the processes are logically independent and data dependence between the processes is very streamlined. The processes are implemented in C and Fortran. Two CCD cameras are used to input the stereo images. Since we use only one frame grabber the images are extracted in three steps: first the camera grabs the left image; then it snaps the image and; finally it saves the image. This is repeated for the right camera. The left and right images are then processed independently. This processing involves low level image processing. It is followed by matching the left and right processed images to determine the three-dimensional position of object points in a scene. The low level processing involves convolution with a Laplacian filter (LPCN) followed by extracting zero-crossings (EDGE) and thresholding these edges (TEDGE). The left and right images are displayed before and after the low

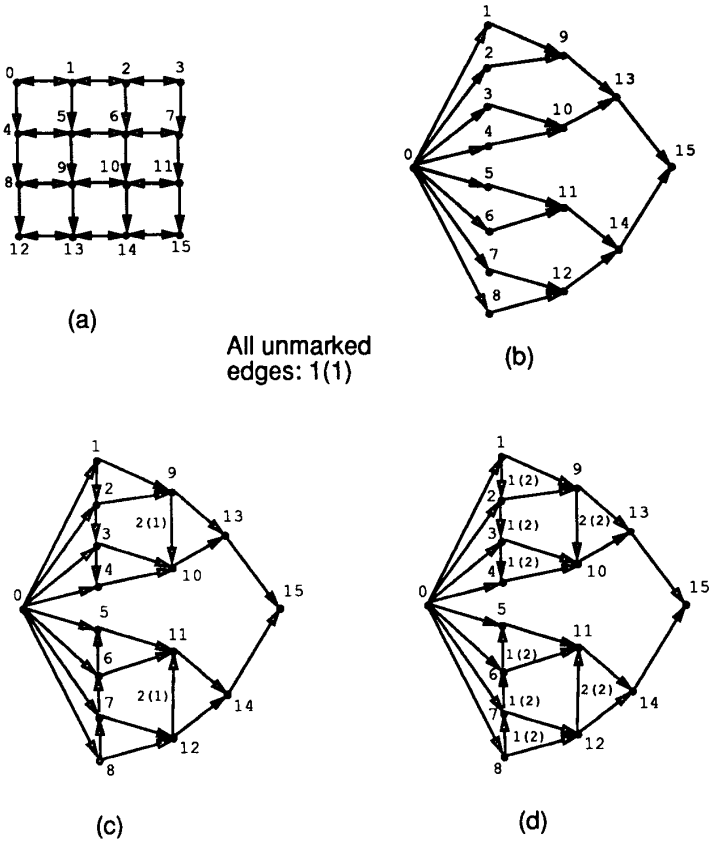


Fig. 11. The problem graphs for the 8-node hosts: (a) S_7, (b) S_8, (c) S_9, (d) S_{10}.

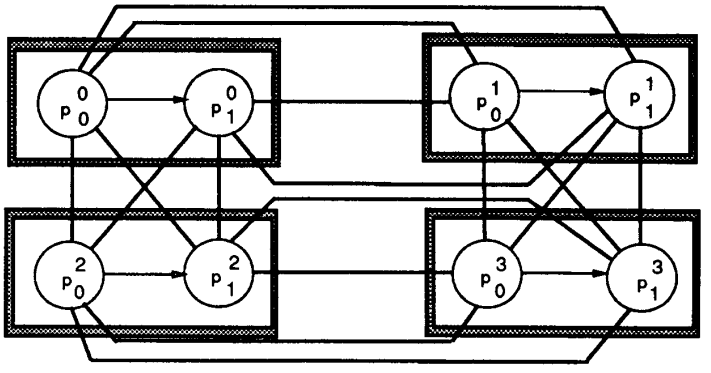


Fig. 12. An example of the extended host graph for the 4-node host graph.

level processing (DISPLAY processes). After independently processing the left and right images, they are matched using a relaxation algorithm (RELAXATION). The final diparity map is then displayed. The images are gray level images of size 256×256. Thus, the communication volume of every edge (except the edge between RELAXATION and DISPLAY5) in the problem graph is 256 Kbits. The communication volume between RELAXATION and DISPLAY5 is 512 Kbits. All the DISPLAY processes take approximately 1.8 s.

The distributed environments (Fig. 14) are comprised of four processors connected in two different ways. ZEUS is a Sun 4 capable of 10 MIPS and 1 MFLOPS. TOMJR is an IBM RT/PC II with 4.4 MIPS and 1 MFLOPS. AMAZON is an IBM RS/6000 (520) with 27.5 MIPS and 7.4 MFLOPS. GANGES is an IBM RS/6000 (530) with 34.5 MIPS and 10.9 MFLOPS. The interconnection between them as shown in Fig. 14 is via Ethernet at 10 MB/s. There are certain constraints introduced in the mapping. The processes GRAB,

TABLE I
INITIAL ASSIGNMENTS FOR THE 4-NODE HOST

S	Initial Mapping $p_0^0\ p_1^0\ p_0^1\ p_1^1\ p_0^2\ p_1^2\ p_0^3\ p_1^3$	OF_1
S_1	(1 2 5 6 0 3 4 7)	2
S_2	(0 1 2 4 3 6 5 7)	1
S_3	(2 5 0 1 3 4 7 6)	5
S_4	(2 5 0 1 3 4 7 6)	4
S_5	(0 1 2 7 5 3 4 6)	5
S_6	(0 2 7 5 1 3 4 6)	4

TABLE II
FINAL ASSIGNMENTS FOR THE 4-NODE HOST AFTER MODIFIED PAIRWISE EXCHANGE

S	Modified Pairwise Exchange $p_0^0\ p_1^0\ p_0^1\ p_1^1\ p_0^2\ p_1^2\ p_0^3\ p_1^3$	OF_1	N
S_1	(1 2 5 6 0 3 4 7)	2	0
S_2	(0 1 2 4 3 6 5 7)	1	0
S_3	(2 1 0 5 3 4 7 6)	3	1
S_4	(5 3 6 0 4 2 7 1)	2	8
S_5	(4 1 2 7 0 3 5 6)	3	2
S_6	(1 2 4 5 0 3 7 6)	3	2

TABLE III
INITIAL ASSIGNMENTS FOR THE 8-NODE HOST

S	Initial Mapping $p_0^0\ p_1^0\ p_0^1\ p_1^1\ p_0^2\ p_1^2\ p_0^3\ p_1^3\ p_0^4\ p_1^4\ p_0^5\ p_1^5\ p_0^6\ p_1^6\ p_0^7\ p_1^7$	OF_1
S_7	(5 6 9 10 1 2 13 14 4 7 8 11 0 3 12 15)	2
S_8	(0 1 13 9 3 2 10 7 5 11 4 14 6 8 15 12)	4
S_9	(0 2 10 9 12 6 11 7 3 1 4 13 5 8 14 15)	7
S_{10}	(0 2 10 9 12 6 11 7 3 1 4 13 5 8 14 15)	5

TABLE IV
FINAL ASSIGNMENTS FOR THE 8-NODE HOST AFTER MODIFIED PAIRWISE EXCHANGE

S	Modified Pairwise Exchange $p_0^0\ p_1^0\ p_0^1\ p_1^1\ p_0^2\ p_1^2\ p_0^3\ p_1^3\ p_0^4\ p_1^4\ p_0^5\ p_1^5\ p_0^6\ p_1^6\ p_0^7\ p_1^7$	OF_1	N
S_7	(5 6 9 10 1 2 13 14 4 7 8 11 0 3 12 15)	2	0
S_8	(0 1 7 9 3 2 10 13 5 11 4 14 6 8 15 12)	3	1
S_9	(9 1 10 2 12 8 11 3 7 0 6 13 5 4 14 15)	5	5
S_{10}	(1 2 10 9 12 6 11 7 3 0 4 13 5 8 14 15)	4	1

SNAP, and SAVE can only be done on ZEUS (since the frame grabber is in ZEUS). DISPLAY5 is done only on TOMJR. RELAXATION is executed only on TOMJR for the first distributed environment [Fig. 14(a)]. Finally, none of the processes other than GRAB, SNAP, and SAVE can be executed on ZEUS. The computation times of the various processes on the processors are given in Table V.

The mapping for the first distributed environment starts with the assignment of tasks using the exclusion matrix. The initial assignment included GRAB, SNAP, and SAVE mapped onto ZEUS and RELAXATION and DISPLAY5 mapped onto TOMJR. Since no other process can be mapped onto ZEUS, the remaining 10 processes need to be mapped onto TOMJR, GANGES, and AMAZON. Because of the relative computational powers of TOMJR, AMAZON, and GANGES the extended host graph formed is as shown in Fig. 15. Fig. 15 also gives the mapping of the processes onto the pseudo-processors. Fig. 16 gives the mapping of the processes onto the processors for the first distributed environment.

The mapping for the second distributed environment is similar to the first distributed environment except that RELAXATION is not constrained to be executed on TOMJR. Moreover, GANGES and AMAZON can communicate with each other. Fig. 17 shows the extended host graph and the mappings of processes onto pseudo-processors. Note that GANGES has the largest number of pseudo processors since it is computationally the most powerful machine. Fig. 18 gives the mapping of the processes onto the processors for the

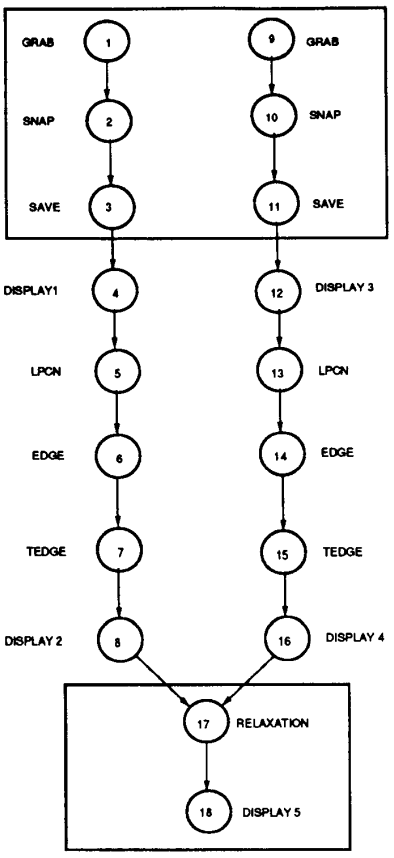


Fig. 13. The stereo problem graph. The processes within the dotted lines indicate constraints.

TABLE V
THE COMPUTATION TIMES IN SECONDS OF THE VARIOUS PROCESSES ON THE PROCESSORS. THE ROWS INDICATE THE PROCESSORS AND THE COLUMNS INDICATE THE PROCESSES

	5	6	7	13	14	15	17
1	0.1	0.07	0.08	0.1	0.07	0.08	2.2
2	0.11	0.07	0.08	0.11	0.07	0.08	2.4
3	0.94	0.72	0.85	0.94	0.72	0.85	6.9

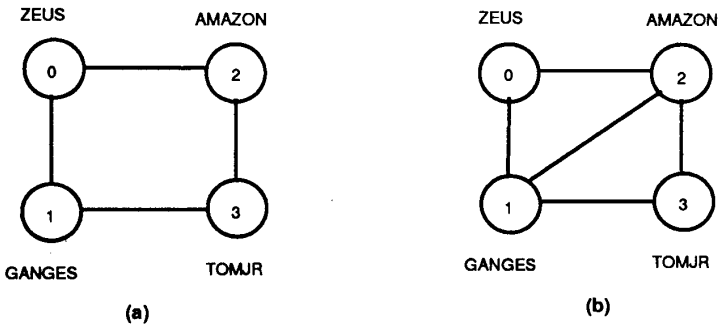


Fig. 14. (a) The first distributed environment and (b) the second distributed environment.

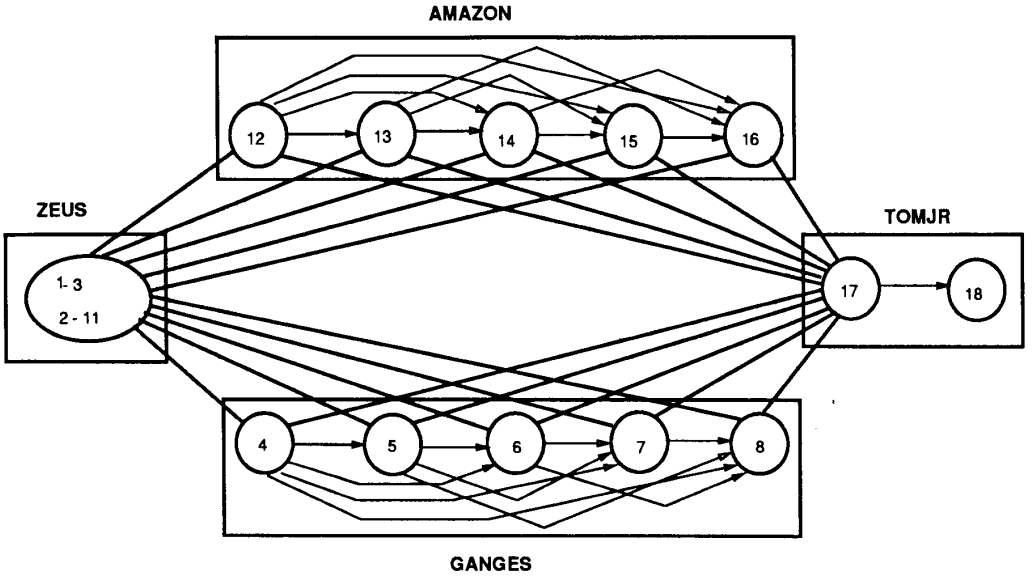


Fig. 15. The extended host graph with the processes mapped onto the pseudo-processors for the first distributed environment.

second distributed environment. In both examples of the distributed environments, no pairwise exchange was performed.

Both these mappings were implemented using OF_2 on the respective distributed environments and real-time stereo was achieved.

VII. Traffic Scheduling

Having done the mapping of processors onto the pseudo processors to optimize certain objective functions, the traffic splitting ability of the communication node (packet switching) can be used to optimize the traffic volume through the physical links by using different routes. By routing traffic along different paths, the total traffic volume can be increased to maximize the network throughput. When determining traffic paths, both traffic volume and delay must be considered. We shall assume that the network delay due to the transfer of messages is negligible in comparison to the actual time of transfer of the messages [54].

We start with sorting the pseudo processors into subsets according to the phase in which they communicate, as shown

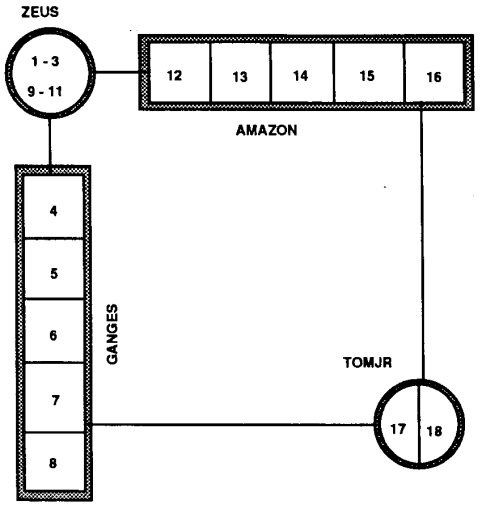


Fig. 16. The final mapping of the processes onto the processors for the first distributed environment.

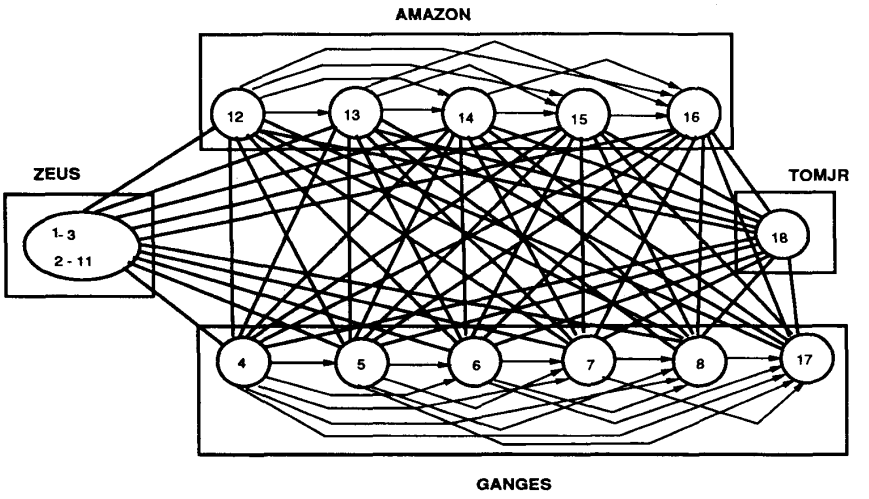


Fig. 17. The extended host graph with the processes mapped 7 onto the pseudo-processors for the second distributed environment.

below:

$$P_k = \{\bar{p}_k^j\}, 1 \leq k \leq N_p, 1 \leq j \leq L_k$$

where N_p is the number of phases (subsets) and L_k is the number of pseudo processors in the subset P_k that communicates with other pseudo processors in the same subset P_k. $\bar{p}_k^j$ is to be distinguished from the pseudo processor p_k^j. The communication volume for each pair of pseudo processors in the same phase is known. The purpose of traffic scheduling is to maximize the total network traffic.

Optimal traffic scheduling can be obtained by optimizing the traffic schedule for each phase. Consider the case where a pseudo processor $\bar{p}_{k_1}^i = p_l^m$ communicates with a pseudo processor $\bar{p}_{k_2}^j = p_q^r$. This creates two possibilities:

1) If $m = r$, then the pseudo processors p_l^m and p_q^r share the same communication node. Thus, the communication cost between them can be ignored.
2) If $m \neq r$, then the communication of the message from $\bar{p}_{k_1}^i$ to $\bar{p}_{k_2}^j$ will use some path from the processor p_m to the processor p_r in the host graph.

Thus, for each phase k, a multiple commodity flow network problem can be generated. The flow volume between processors p^m and p^r is equal to the communication flow between the pseudo processors p_l^m and p_q^r. The multiple commodity flow network problem can be solved by the interprocessor traffic scheduling algorithm proposed by Bianchini and Shen [54].

VIII. Conclusion

A generalized mapping scheme for a distributed computing environment is proposed. The proposed strategy uses the knowledge from the given algorithm and the given architecture to guide the mapping. The concept of pseudo processors can help us understand and describe the feasibility of a mapping, p-deadlock, and q-deadlock. An accurate characterization of the communication overhead is used as our objective function to evaluate the optimality of the mapping. The two level optimization schemes described here illustrate the use of pseudo processors in reducing the communication overhead for a generalized system.

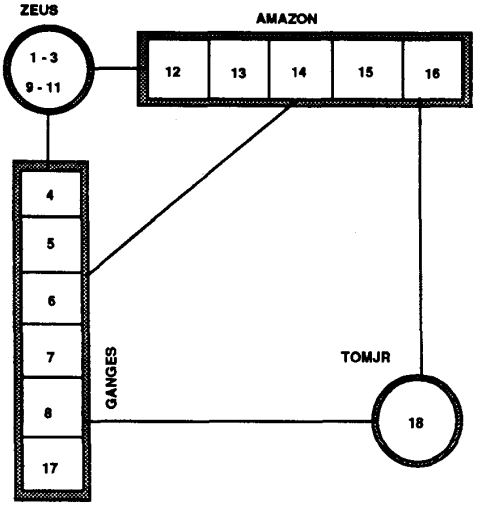


Fig. 18. The final mapping of the processes onto the processors for the second distributed environment.

The proposed scheme is a substantial advancement in scope over the current state-of-the-art. Conditions are derived to prevent deadlock and to check the feasibility of the mapping. The mapping model is very general and can incorporate almost every practical constraint. A two level mapping optimization technique is used to arrive at extremely good results. The key difference between our strategy and those proposed previously is the interdependence between the algorithm and the architecture. We use the knowledge from the given algorithm and the given architecture to guide the mapping.

The proposed strategy was tested in both a simulated environment and a real distributed environment. The results

achieved in both implementations indicate the efficacy of the proposed mapping scheme.

REFERENCES

[1] F. Berman and L. Snyder, "On mapping parallel algorithms into parallel architectures," in *Proc. Int. Conf. Parallel Processing,* Aug. 1984, pp. 307–309

[2] S. H. Bokhari, "On the mapping problem," *IEEE Trans. Comput.*, vol. C-30, no. 3, pp. 207–214, Mar. 1981.

[3] ——, "Dual processor scheduling with dynamic reassignment," *IEEE Trans. Software Eng.*, vol. SE-5, no. 4, pp. 326–334, July 1979.

[4] ——, "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system," *IEEE Software Eng.*, vol. SE-7, no. 6, pp. 335–341, Nov. 1981.

[5] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Software Eng.*, vol. 14, no. 2, pp. 42–45, Feb. 1988.

[6] ——, "Design of a loosely-coupled distributed multiprocessing network," in *Proc. Int. Conf. Parallel Processing,* Aug. 1984, pp. 42–45.

[7] M. J. Gonzales, "Deterministic processor scheduling," *ACM Comput. Surveys,* vol. 9, no. 3, Sept. 1977.

[8] E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Recent developments in deterministic sequencing and scheduling: A survey," in *Deterministic and Stochastic Scheduling,* M. A. H. Dempster *et al.*, Eds. Reidel, 1982

[9] S. J. Kim and J. C. Browne, "A general approach to mapping of parallel computations upon multiprocessor architectures," in *Proc. Int. Conf. Parallel Processing,* Aug. 1988, pp. 1–8.

[10] S. Y. Lee and J. K. Aggarwal, "A mapping strategy for parallel processing," *IEEE Trans. Comput,* vol. C-36, no. 4, pp. 433–442, Apr. 1987.

[11] E. G. Coffman, *Computer and Job-Shop Scheduling Theory.* New York: Wiley, 1974.

[12] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.,* vol. 10, pp. 384–393, 1975.

[13] K. Vairavan and R. A. DeMillo, "On the computational complexity of a generalized scheduling problem," *IEEE Trans. Comput.*, vol. C-25, pp. 1067–1073, Nov. 1976.

[14] D. Bernstein, M. Rodeh, and I. Gertner, "On the complexity of scheduling problems for parallel/pipelined machines," *IEEE Trans. Comput.*, vol. C-38, no. 9, pp. 1308–1313, Sept. 1989.

[15] J. K. Lenstra and A. H. G. R. Kan, "Complexity of scheduling under precedence constraints," *Oper. Res.*, vol. 26, pp. 22–35, Jan. 1978.

[16] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE Comput. Mag.*, vol. 15, pp. 50–56, June 1982.

[17] J. C. Browne, "Formulation and programming of parallel computations: A unified approach," in *Proc. Int. Conf. Parallel Processing,* Aug. 1985, pp. 624–631

[18] ——, "Framework for formulation and analysis of parallel computation structures," *Parallel Comput.*, vol. 3, pp. 1–9, 1986.

[19] L. D. Wittie and A. M. Tilborg, "MICROS, a distributed operating system for MICRONET, a reconfigurable network computer," *IEEE Trans. Comput.*, vol. C-29, no. 12, pp. 1133–1144, Dec. 1980.

[20] R. M. Bryant and R. A. Finkel, "A stable distributed scheduling algorithm," in *Proc. Int. Conf. Distributed Comput. Syst.*, Apr. 1981, pp. 314–323.

[21] T. C. K. Chou and J. A. Abraham, "Load balancing in distributed systems," *IEEE Trans. Softwre Eng.*, vol. SE-8, no. 4, pp. 401–412, July 1982.

[22] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer—Designing an MIMD shared memory parallel computer," *IEEE Trans. Comput.*, vol. C-32, no. 2, pp. 175–189, Feb. 1983.

[23] C. C. Shen and W. H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Trans. Comput.*, vol. C-34, no. 3, pp. 197–203, Mar. 1985.

[24] Kashahara and Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Trans. Comput.*, vol. C-33, no. 11, pp. 1023–1029, Nov. 1984.

[25] W. W. Chu, L. J. Hooloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Comput. Mag.*, vol. 13, no. 11, pp. 57–69, Nov. 1980.

[26] T. C. Hu, "Parallel sequencing and assembly line problem," *Oper. Res.*, vol. 9, pp. 841–848, Nov. 1961.

[27] E. G. Coffman and R. L. Graham, "Optimal scheduling for two-processor systems," *Acta Informatica,* vol. 1, pp. 200–213, 1972.

[28] A. K. Jones *et al.*, "StarOS, a multiprocessor operating system for the support of task forces," in *Proc. 7th Symp. Oper. Syst. Principles,* Dec. 1979, pp. 117–127

[29] J. Ousterhout, D. Scelza, and P. Sindhu, "Medusa: An experiment in distributed operating system structure," *Commun. ACM,* vol. 23, no. 2, pp. 92–105, Feb. 1980.

[30] J. A. Stankovic *et al.*, "A review of current research and critical issues in distributed system software," *IEEE Computer Society Distributed Process. Tech. Committee Newslett.*, vol. 7, pp. 14–47, Mar. 1985.

[31] K. W. Doty, P. L. McEntire, and J. G. O'Reilly, "Task allocation in a distributed computer system," in *Proc. IEEE InfoCom,* 1982, pp. 33–38.

[32] A. Gabrielian and D. B. Tyler, "Optimal object allocation in distributed computer systems," in *Proc. Int. Conf. Distributed Comput. Syst.,* May 1984, pp. 84–95.

[33] P. Y. R. Ma, E. Y. S. Lee, and J. Tsuchiya, "A task allocation model for distributed computing systems," *IEEE Trans. Comput.*, vol. C-31, no. 1, pp. 41–47, Jan. 1982.

[34] L. M. Ni and K. Hwang, "Optimal load balancing for a multiple processor system," in *Proc. Int. Conf. Parallel Processing,* 1981, pp. 352–357.

[35] J. A. Bannister and K. S. Trivedi, "Task allocation in fault-tolerant distributed systems," *Acta Informatica,* vol. 20, pp. 261–281, 1983.

[36] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," in *Proc. Int. Conf. Distributed Comput. Syst.,* May 1984, pp. 30–39.

[37] ——, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.*, vol. C-37, no. 11, pp. 1384–1397, Nov. 1988.

[38] C. C. Price and S. Krishnaprasad, "Software allocation models for distributed computing systems," in *Proc. Int. Conf. Distributed Comput. Syst.*, May 1984, pp. 40–48.

[39] C. P. Kruskal and A. Weiss, "Allocating independent subtasks on parallel processors, extended abstract," in *Proc. Int. Conf. Parallel Processing,* Aug. 1984, pp. 236–240.

[40] M. O. Ward and D. J. Romero, "Assigning parallel executable intercommunicating subtasks to processors," in *Proc. Int. Conf. Parallel Processing,* Aug. 1984, pp. 392–394.

[41] V. Chaudhary and J. K. Aggarwal, "Generalized mapping of parallel algorithms onto parallel architectures," in *Proc. Int. Conf. Parallel Processing,* Aug. 1990, pp. 137–141.

[42] J. A. Stankovic and I. S. Sidhu, "An adaptive bidding algorithm of processes, clusters, and distributed groups," in *Proc. Int. Conf. Distributed Comput. Syst.,* May 1984, pp. 49–59.

[43] H. S. Stone, "Critical load factors in two-processor distributed systems," *IEEE Trans. Software Eng.*, vol. SE-4, no. 3, pp. 254–258, May 1978.

[44] H. S. Stone and S. H. Bokhari, "Control of distributed processes," *IEEE Comput. Mag.*, vol. 11, pp. 97–106, July 1978.

[45] L. Kleinrock, *Queueing Systems, Vol. 2: Computer Applications.* New York: Wiley, 1976.

[46] L. Kleinrock and A. Nilsson, "On optimal scheduling algorithms for time-shared systems," *J. ACM,* vol. 28, no. 3, pp. 477–486, July 1981.

[47] J. A. Stankovic *et al.*, "An evaluation of the applicability of different mathematical approaches to the analysis of decentralized control algorithms," in *Proc. IEEE COMPSAC,* Nov. 1982, pp. 62–69.

[48] P. H. Enslow Jr., "What is a 'distributed' data processing system," *IEEE Comput. Mag.*, vol. 11, no. 1, pp. 13–21, Jan. 1978.

[49] Bolt Beranek and Newman Inc., "Butterfly (TM) parallel processor overview," Cambridge, MA, June 1985.

[50] H. Forsdick, R. Schantz, and R. Thomas, "Operating systems for computer networks," *IEEE Trans. Comput.*, no. 11, Jan. 1978.

[51] M. Hanan and J. M. Kurtzberg, "A review of the placement and quadratic assignment problems," *SIAM Rev.*, vol. 14, pp. 324–342, Apr. 1972.

[52] C. E. McDowell and W. F. Appelbe, "Processor scheduling for linearly connected parallel processors," *IEEE Trans. Comput.*, vol. C-35, no. 7, pp. 632–638, July 1986.

[53] S. Sahni, "Scheduling multipipeline and multiprocessor computers," *IEEE Trans. Comput.*, vol. C-33, no. 7, pp. 637–645, July 1984.

[54] R. P. Bianchini, Jr. and J. P. Shen, "Interprocessor traffic scheduling algorithm for multiple-processor networks," *IEEE Trans. Comput.*, vol. C-36, no. 4, pp. 396–409, Apr. 1987.

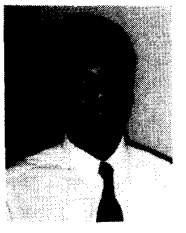
[55] G. R. Andrews, D. P. Dobkin, and P. J. Downey, "Distributed allocation with pools of servers," in *Proc. ACM SIGACT-SIGOPS Symp. Principles Distributed Comput.*, Aug. 1982, pp. 73–83.

[56] L. M. Casey, "Decentralized scheduling," *Australian Comput. J.*, vol. 13, pp. 58–63, May 1981.

[57] C. Gao, J. W. S. Liu, and M. Railey, "Load balancing algorithms in homogeneous distributed systems," in *Proc. Int. Conf. Parallel Processing,* Aug. 1984, pp. 302–306.

[58] D. Klappholz and H. C. Park, "Parallelized process scheduling for a tightly-coupled MIMD machine," in *Proc. Int. Conf. Parallel Process-*

ing, Aug. 1984, pp. 315–321.
[59] V. M. Lo, "Task assignment to minimize completion time," in *Proc. Int. Conf. Distributed Comput. Syst.,* May 1985, pp. 329–336.
[60] S. Majumdar and M. L. Green, "A distributed real time resource manager," in *Proc. IEEE Symp. Distributed Data Acquisition, Comput. Contr.,* 1980, pp. 185–193.
[61] C. V. Ramamoorthy *et al.*, "Optimal scheduling strategies in a multiprocessor system," *IEEE Trans. Comput.,* vol. C-21, no. 2, pp. 137–146, Feb. 1972.
[62] K. Ramamrithan and J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time system," in *Proc. Int. Conf. Distributed Comput. Syst.,* May 1984, pp. 96–107.
[63] J. Reif and P. Spirakis, "Real-time resource allocation in a distributed system," in *Proc. ACM SIGACT-SIGOPS Symp. Principles Distributed Comput.,* Aug. 1982, pp. 84–94.
[64] T. G. Saponis and P. L. Crews, "A model for decentralized control in a fully distributed processing system," in *Proc. COMPCON,* 1980, pp. 307–312
[65] J. A. Stankovic, "The analysis of a decentralized control algorithm for job scheduling utilizing bayesian decision theory," in *Proc. Int. Conf. Parallel Processing,* 1981, pp. 333–337.
[66] ——, "A heuristic of cooperation among decentralized controllers," in *Proc. IEEE INFOCOM,* Apr. 1983, pp. 331–339.
[67] ——, "An application of bayesian decision theory to decentralized control of job scheduling," *IEEE Trans. Comput.,* vol. C-34, no. 2, pp. 117–130, Feb. 1985.
[68] ——, "Stability and distributed scheduling algorithms," in *Proc. ACM Nat. Conf.,* vol. 2, Mar. 1985.
[69] P. Tang, P. C. Yew, Z. Fang, and C. Q. Zhu, "Deadlock prevention in processor self-scheduling for parallel nested loops," in *Proc. Int. Conf. Parallel Processing,* Aug. 1987, pp. 11–18.
[70] S. P. Lo and V. D. Gligor, "Properties of multiprocessor scheduling algorithms," in *Proc. Int. Conf. Parallel Processing,* Aug. 1987, pp. 867–870.
[71] C. E. Houstis, "Allocation of real-time applications to distributed systems," in *Proc. Int. Conf. Parallel Processing,* Aug. 1987, pp. 863–866.
[72] K. M. Baumgartner and B. W. Wah, "Load balancing protocols on a local computer system with a multiaccess network," in *Proc. Int. Conf. Parallel Processing,* Aug. 1987, pp. 851–858.
[73] D. C. Gerogiannis and S. C. Orphanoudakis, "Efficient embedding of interprocessor communication in parallel implementations of intermediate level vision tasks," in *Proc. Int. Conf. Comput. Vision Pattern Recognition,* June 1990, pp. 368–372.
[74] M. Yasberi, J. C. Browne, and D. P. Agrawal, "Analysis and design of parallel algorithms and implementations for some image processing operations," in *Proc. Int. Conf. Parallel Processing,* Aug. 1987, pp. 901–908.
[75] T. C. K. Chou and J. A. Abraham, "Load redistribution under failure in distributed systems," *IEEE Trans. Comput.,* vol. C-32, no. 9, pp. 799–808, Sept. 1983.
[76] Y. C. Chow and W. H. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput,* vol. C-28, no. 5, pp. 354–361, May 1979.
[77] R. Manner, "Hardware task/processor scheduling in a polyprocessor environment," *IEEE Trans. Comput.,* vol. C-33, no. 7, pp. 626–636, July 1984.
[78] L. M. Ni and K. Abani, "Nonpreemptive load balancing in a class of local area networks," in *Proc. Comput. Networking Symp.,* Dec. 1981, pp. 113–118.
[79] M. L. Powell and B. P. Miller, "Process migration in DEMOS/MP," in *Proc. 9th Symp. Oper. Syst. Principles (OS Rev.),* vol. 17, Oct. 1983, pp. 110–119.
[80] J. A. Stankovic, "Simulation of three adaptive, decentralized controlled, job scheduling algorithms," *Comput. Networks,* vol. 8, no. 3, pp. 199–217, June 1984.
[81] N. Deo, *Graph Theory with Applications to Engineering and Computer Science.* New Delhi, India: Prentice-Hall of India, 1987.
[82] V. Chaudhary and J. K. Aggarwal, "On the complexity of parallel image component labeling," in *Proc. Int. Conf. Parallel Processing,* Aug. 1991, pp. 183–187.
[83] ——, "Parallelism in computer vision - A review," in *Parallel Algorithms for Machine Intelligence and Computer Vision,* V. Kumar and P. Gopalakrishnan and L. Kanal, Eds. New York: Springer-Verlag, 1990, pp. 271–309.
[84] ——, "Parallelism in low-level vision - A review," in *Data Analysis in Astronomy III,* V. De Gesu, L. Scarsi, P. Crane, J. H. Friedman, S. Levialdi, and M. C. Maccarone. New York: Plenum, 1988, pp. 255–270.
[85] M. H. Sunwoo, B. S. Baroody, and J. K. Aggarwal, "A parallel algorithm for region labeling," in *Proc. IEEE Workshop Comput. Architecture for Pattern Anal. Mach. Intell.,* 1987, pp. 27–34.
[86] I. Herstein, *Abstract Algebra.* New York: Wiley, 1984, 212 pp.
[87] Y. C. Kim and J. K. Aggarwal, "Positioning three-dimensional objects using stereo images," *IEEE J. Robot. Automat.,* vol. RA-3, no. 4, pp. 361–373, Aug. 1987.



Vipin Chaudhary (S'89–M'92) received the B.Tech. (Hons.) degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, in 1986, the M.S. degree in computer science, and the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin, in 1989 and 1992, respectively.

He is currently an Assistant Professor of Electrical and Computer Engineering, Wayne State University. From January 1992 to May 1992, he was a postdoctoral researcher with the Computer and Vision Research Center, Department of Eletrical and Computer Engineering, The University of Texas at Austin. His current research interests include algorithms, architectures, interconnection networks, and task assignment for parallel and distributed systems, performance evaluation, computer vision, and computer graphics.

Dr. Chaudhary was awarded the President of India Gold Medal for academic excellence in 1986. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

J. K. Aggarwal (S'62–M'65–SM'74–F'76) has served on the faculty of the University of Texas at Austin College of Engineering since 1964 and is now the Cullen Professor of Electrical and Computer Engineering and Director of the Computer and Vision Research Center. His research fields include computer vision, parallel processing of images, and pattern recognition. He is the author or editor of 6 books and 20 book chapters; author of 130 journal papers, and 11 reprinted articles, as well as numerous proceedings papers, and technical reports.

Dr. Aggarwal served as Chairman of the IEEE Computer Society Technical Committee on Pattern Analysis and Machine Intelligence (1987–1989); Director of the NATO Advanced Research Workshop on Multisensor Fusion for Computer Vision, Grenoble, France (1989); and was Chairman of the Computer Vision Program Committee, 10th International Conference on Pattern Recognition, Atlantic City, NJ (1990). Currently, he serves as IEEE Computer Society representative on International Association for Pattern Recognition and is Treasurer of International Association for Pattern Recognition.