Minimizing the Application Execution Time Through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System

Min Tan, *Student Member*, *IEEE*, Howard Jay Siegel, *Fellow*, *IEEE*, John K. Antonio, *Member*, *IEEE*, and Yan Alexander Li, *Member*, *IEEE*

Abstract—In a heterogeneous computing (HC) environment consisting of different types of machines, an application program is decomposed into subtasks, each of which is computationally homogeneous. The goal is to execute subtasks on the machines in such a way that the total program execution time is minimized. A mathematical framework is presented that models the matching of subtasks to machines, scheduling of subtasks' computation, scheduling of intermachine communication steps, and selection of sources of shared data items for intermachine communication (data relocation). The goal of this work is to generate a provably optimal scheme for communicating shared data among subtasks as an enhancement to any given matching and scheduling. Initially, it is assumed that at any instant in time, only one machine is being used for program execution and only one subtask is being executed. Based on this assumption, a polynomial algorithm is introduced to optimize scheduling and data relocation with respect to any given matching of subtasks to machines. The data relocation scheme is then extended to reduce intermachine data communication time in an HC environment with a given matching and scheduling of subtasks' computation where: 1) multiple subtasks' computations can be performed concurrently on different machines; 2) subtask computation stteps can be overlapped with other subtasks' communication steps for intermachine data transfers; and 3) machines in the HC suite are interconnected by a shared-bus type of network.

Index Terms—Data distribution, data relocation, data-reuse, distributed computing, heterogeneous computing, multiple data-copies, optimization, scheduling.

1 Introduction

A single application program often requires many different types of computation that result in different needs for machine capabilities. Heterogeneous computing (HC) is the effective use of the diverse hardware and software components in a heterogeneous suite of machines connected by a high-speed network to meet the varied computational requirements of a given application [5], [9], [12], [16], [17], [24]. The results of this research may be applied to target HC platforms that may include high-performance parallel and vector machines as well as a cluster of different (or similar) types of workstations.

The goal of HC is to decompose an application program into subtasks, each of which is computationally homogeneous. Then, each subtask is assigned to the machine where it is best suited for execution. In general, each subtask is assigned to one of the machines in the heterogeneous suite in a way that minimizes the total execution time (consisting of both subtasks' computation times and intermachine communication times) of the application program. The subtask assignment problem is referred to as matching in HC

There are a variety of mathematical formulations for matching, collectively called selection theory, that have been proposed to choose the appropriate machine for each subtask of an application program (e.g., [4], [8], [15], [22]). A collection of algorithms, called graph-based algorithms in this paper (e.g. [2], [15], [19], [21]), have been developed to solve matching-related problems based on a subtask flow graph that describes the data dependencies among subtasks of an application program. As shown in Fig. 1a, each vertex of the subtask flow graph represents a subtask. Let $S[k]$ denote the kth subtask. Let a data item be a block of information that can be transferred between subtasks. There is an edge from $S[k]$ to $S[j]$ labeled with the name of the data item that $S[k]$ transfers to $S[j]$ during execution. An extra vertex labeled *Source* denotes the locations where the initial data elements of the program are stored. The purpose of selection theory formulations and graph-based algorithms is to find the matching scheme that minimizes the total execution time of the application program. For this paper, it is assumed that matching (i.e., assignment of subtasks to machines) is static and has already been done.

- *M. Tan and H.J. Siegel are with the Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285. E-mail: {mtan, hj}@ecn.purdue.edu.*
- *J.K. Antonio is with the Department of Computer Science, Texas Tech University, Lubbock, TX 79409-3104. E-mail: antonio@cs.ttu.edu.*
- *Y.A. Li is with Intel Corp., 2200 Mission College Blvd., Santa Clara, CA 95054-1537. E-mail: ali2@mipos2.intel.com.*

Manuscript received 17 May 1996; revised December 1996.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number 100205.0.

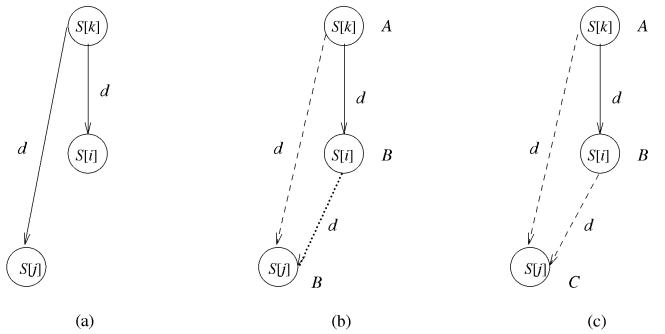


Fig. 1. Data-distribution situations in HC: (a) subtask flow graph, (b) data-reuse, (c) the multiple data-copies situation.

A data item can be an integer, an array of characters, or a large file, such as a multispectral image. Based on static (compile time) analysis, a given subtask may need as input one or more data items generated (or modified) by one or more other subtasks. Using information from the subtask flow graph, a data item is denoted by the ordered pair (s, d), where $s \geq 0$ is the number of the subtask that generates the needed value of d upon completion of execution of that subtask. For example, (3, x) represents the value of variable x generated by subtask $S[3]$ upon completion of its execution. In the notation (s, d), $s = -1$ if the needed value of d is an initial input to the program. Two data items are the same if and only if they are both associated with the same variable name in an application program and the corresponding value of the data is generated by the same subtask (which implies that the two data items have the same value).

In general, most of the graph-based algorithms for matching-related problems assume that the pattern of data transfers among subtasks is known a priori and can be illustrated using some type of graph (e.g., [2], [14], [15], [19], [21]). Thus, no matter which machine is used for executing each subtask of a specific application program, the locations (subtasks) from which each subtask obtains its corresponding input-data items are determined by the subtask flow graph and independent of any particular matching scheme between machines and subtasks.

The above traditional formulation is refined in this paper by considering two data-distribution situations called data-reuse and multiple data-copies. It is assumed that each subtask $S[i]$ keeps copies of all its input and output-data items on the machine to which $S[i]$ is assigned by the matching scheme. The issues on how to free the memory space for storing the copies of data items after they are forwarded to other subtasks are discussed in Section 3. Data-reuse arises when two subtasks, $S[i]$ and $S[j]$, need the same data item from $S[k]$ (as in the example subtask flow graph in Fig. 1a). For any data item $e = (k, d)$, $\underline{e}$ represents the value of the associated data and $\lfloor e \rfloor$ (or $\lfloor d \rfloor$) represents the size of the associated data. As shown in Fig. 1b, suppose the particular matching scheme is the one that assigns $S[k]$ to machine A, and both $S[i]$ and $S[j]$ to machine B. Furthermore, assume, for this example, that the subtasks are executed in the order k, i, then j. In this case, there is no need to transfer data item e from $S[k]$ to $S[j]$, as shown by the dashed line in Fig. 1b, because e is already on machine B due to the data transfer of e from $S[k]$ to $S[i]$ completed earlier (solid line in Fig. 1b). If a traditional graph-based formulation were used to compute intersubtask communication cost, then the impact of data-reuse is ignored. The techniques derived in this paper account for (and take advantage of) the data-reuse situation.

The multiple data-copies situation arises when two subtasks, $S[i]$ and $S[j]$, need the same data item $e = (k, d)$ from $S[k]$, where $S[i]$, $S[j]$, and $S[k]$ are assigned to three different machines in the HC system. In the example in Fig. 1c, the matching scheme assigns $S[k]$ to machine A, $S[i]$ to machine B, and $S[j]$ to machine C. Therefore, $S[j]$ can obtain data item e from either machine A or machine B (shown by the two dashed lines). The choice that results in the shortest time should be selected. Retrieving the needed data item from the selected source is referred to as data relocation. In general, when using information only from the subtask flow graph, the possibility of multiple sources of a needed data item due to a specific matching scheme is not considered; however, the techniques developed in this paper do optimize with respect to data relocation options.

When a subset of subtasks can be executed in any order, and the multiple data-copies situation is considered, varying the order of the execution of these subtasks (while maintaining the data dependencies among all subtasks) can impact the execution time of the application program. Determining the order for executing the subtasks is referred to as scheduling in this paper. Thus, matching determines on which machine each subtask should be executed, while scheduling determines when to execute a subtask on the machine to which it is assigned [16].

The intermachine communication time between subtasks can be substantial in an HC system. Thus, this intermachine communication time can be a major factor in degrading the performance of an HC system. Taking the effects produced by data-reuse and multiple data-copies into account can potentially decrease this time and, hence, the total execution time of the application program. This paper focuses on methods for minimizing the communication time of an application program for a given matching scheme. In particular, the impact of scheduling and data relocation schemes on the communication time of the subtasks are examined.

In Section 2, a mathematical model for matching, scheduling, and data relocation in HC is presented. Section 3 introduces an extension to the usual scheduling methodology, in which temporally interleaved execution of the atomic input operations of different subtasks (TIE) is described. When considering multiple data-copies, this extension to scheduling can decrease the execution time of an application program.

Issues related to scheduling and data relocation for HC systems are presented in Section 4. It is assumed that at any instant in time during the execution of a specific application program, only one machine is being used for program execution and only one subtask is being executed. In practice, concurrent execution of multiple subtasks is possible. However, the simplifying assumption of the subtasks and communication steps being totally ordered in time (i.e., no two

subtasks' computation or intermachine communication can be executed concurrently with respect to one another) is used in Section 4 as a step toward solving the more general problem. (Note that each individual subtask can be executed on a parallel machine and exploit the parallelism available.) This simplifying assumption is used by many other researchers as well (e.g., [2], [5], [19], [21]).

Section 4 presents a procedure for representing the data relocation scheme by using a directed acyclic graph associated with the application program. A minimum spanning tree based algorithm (referred to as the TIE algorithm) is described. For a given matching, the TIE algorithm finds the optimal scheduling scheme for the execution of subtasks and the optimal data relocation scheme for each subtask. Both data-reuse and multiple data-copies are considered in the TIE algorithm. The correctness of the TIE algorithm is proved mathematically and an example is also provided.

Based on this TIE algorithm, a refinement procedure for the data relocation scheme and the scheduling of intermachine data transfer steps is introduced in Section 5 for HC systems where multiple subtasks' computation and a single intermachine communication are performed simultaneously whenever possible. Such a situation may occur in practice if the machines in the suite share a single communication bus.

The contributions of this paper can be summarized as follows. In order to use an HC system, a scheme to match subtasks to machines must be determined (where the generation of such a matching may involve consideration of subtask scheduling and the movement of data items shared by subtasks). Because the matching problem is, in general, NP-complete [7], heuristics are used to matching solutions. It is not the goal of our paper to provide a heuristic for matching and scheduling subtask computation and compare its performance with other heuristics in the literature (e.g., [1], [18]). A survey of matching and scheduling related heuristics is provided in [16]. Instead, the goal of this work is to reduce the communication time (and, hence, the execution time) of an application program on an HC system by varying data relocation and scheduling of intermachine data transfers, using a given matching. If the totally ordered model used in Section 4 is assumed, then a provably optimal new scheduling (for both computation and communication) and data relocation scheme is derived for the given matching. If the concurrent computation model introduced in Section 5 is applicable, then the technique presented in Section 4 can be employed to enhance the scheduling of intermachine communication and data relocation for the given matching and scheduling of subtask computation.

Although both the totally ordered model used in Section 4 and the concurrent computation model used in Section 5 have certain limitations, practical HC applications do exist that use the above HC models. In [13], an HC application involving the simulation of mixing in turbulent convection in three dimensions was developed on an HC system consisting of Thinking Machines' CM-200 and CM-5, a CRAY 2, and a Silicon Graphics VGX workstation. The required calculations for the simulation were divided into three subtasks and totally ordered in time (i.e., follows the model in Section 4). All three subtasks were parallel programs and were executed on the CM-5, the CRAY 2, and the CM-200. The final results are displayed on the Silicon Graphics VGX workstations. An example of the concurrent computation HC model in Section 5 is a cluster of heterogeneous workstations connected by Ethernet, a perceivably popular and economical HC system [5], [24].

The focus of this paper is to demonstrate the effect of data relocation and the scheduling of communication steps on decreasing the intermachine communication time for an HC application executed on certain chosen classes of HC systems. Furthermore, often research on problems using restricted models can later be built upon to find techniques for solving problems associated with more general models.

2 A Mathematical Model

A mathematical model for matching, scheduling, and data relocation in HC is formalized in this section. This model can describe an HC system constructed by a suite of high-performance machines and/or a cluster of workstations. This mathematical model assumes that all machines in the HC suite are under the control of these matching, scheduling, and data relocation schemes. While the model is described in terms of a single application program, that application program could actually be a collection of unrelated programs, each decomposed into subtasks. It is assumed that the matching, scheduling, and data relocation are done statically (e.g., at compile time) for a production job that will make repeated use of the HC suite. That is, the time for determining matching, scheduling, and data relocation does not impact the response time for program execution, and it is worthwhile to invest resources in attempting to optimize the use of the HC suite by this application program because it will be executed frequently. The model serves as the mathematical basis for the TIE algorithm presented in Section 4. All notation developed in the remaining sections is summarized in the glossary of notation at the end this paper.

1) An application program $\underline{P}$ is composed of a set of subtasks $\underline{S} = \{S[0], S[1], ..., S[n-1]\}$, where $\underline{n}$ is the number of subtasks in P.
2) Suppose that $\underline{NI[i]}$ is the number of input-data items required by $S[i]$ and $\underline{NG[i]}$ is the number of output-data items generated by $S[i]$. There are two sets of data items associated with each $S[i]$. One is the input-data set $\underline{I[i]} = \{Id[i, 0], Id[i, 1], ..., Id[i, NI[i]-1]\}$, the other is the generated output-data set $\underline{G[i]} = \{Gd[i, 0], Gd[i, 1], ..., Gd[i, NG[i]-1]\}$. Each $Id[i, j]$ and $Gd[i, j]$ is a data item (i.e., a two-tuple as defined in Section 1). The program structure of P is specified by a subtask flow graph.

In this paper, the subtask flow graph of any application program P is assumed to be acyclic. A cycle in a graph represents a loop containing one or more subtasks. If the number of iterations is known, the loop can be unrolled. If the number of iterations is data dependent (i.e., nondeterministic), the loop can be unrolled using a statistical approach (such as that presented in [21]). While the concept presented

here can be applied as a heuristic without unrolling loops, the optimality proofs presented in Section 4 require the acyclic property.

3) An HC system consists of a heterogeneous suite of machines $\underline{M} = \{M[0], M[1], ..., M[m-1]\}$, where $\underline{m}$ is the number of machines in the system.
4) Each $\underline{S[i]}$ of the application program P can be executed by any of the machines $\underline{M[j]}$ in the HC system. There is a computation matrix $\underline{C} = \{C[i, j]\}$ associated with S and M, where $\underline{C[i, j]}$ denotes the computation time of $S[i]$ on machine $M[j]$ [10], [23]. The computation matrix C is assumed to be known. It can be computed from empirical information or by applying two characterization techniques in HC, namely task profiling and analytical benchmarking (see [16] for a survey of these techniques).
5) Suppose that a set of initial data elements $\{d_0, d_1, ..., d_{Q-1}\}$ are required for executing the application program P, where $\underline{Q}$ is the number of initial data elements for P. An initial-data matrix $\underline{H} = \{H[k, j]\}$ is defined, where $H[k, j]$ ($0 \le k < Q$ and $0 \le j < m$) represents the *smallest* communication time for machine $M[j]$ to obtain the initial data element d_k from one of the devices where d_k is stored before the execution of P. Initial data element d_k is also denoted as data item $(-1, d_k)$.
6) The communication function matrix $\underline{D(|e|)} = \{D[s, r](|e|)\}$, for $0 \le s, r < m$, where $D[s, r](|e|)$ denotes the communication time for transferring data item e (of size $|e|$) from machine $M[s]$ to machine $M[r]$ [10], [11]. $D[s, r](|e|)$ includes all the various hardware and software related components of the intermachine communication process, e.g., network latency and the time for data format conversion between $M[s]$ and $M[r]$ when necessary. Having $s = -1$ indicates that $e = (-1, d)$, and d is one of the initial data elements of P, and there exists k ($0 \le k < Q$) such that $d = d_k$ and $D[s, r](|e|) = H[k, r]$.
7) The matching associated with the application program P is defined by an assignment function $\underline{Af}$, such that $Af : S \rightarrow M$. If $Af(i) = j$, then $S[i]$ is assigned to be executed on machine $M[j]$. The assignment function Af corresponds to the matching problem discussed in Section 1. $\underline{NS[j]}$ is defined as the number of subtasks assigned to be executed on machine $M[j]$. Thus,

$$\sum_{j=0}^{m-1} NS[j] = n.$$

8) A scheduling function $\underline{Sf}$ is associated with the application program P. Sf indicates the order for performing a subtask's computation with respect to the other subtasks' computation of the entire application program. For the material presented in Section 4, where the subtasks' computation and communication for the application program P are assumed to be totally ordered in time, $Sf(i) = k$ means that $S[i]$ is the kth ($0 \le k < n$) subtask of the entire application program whose computation is performed. The scheduling function Sf corresponds to the scheduling problem discussed in Section 1.
9) The set of data-source functions is $\underline{DS} = \{DS[0], DS[1], ..., DS[n-1]\}$, where $DS[i](j) = k$ ($0 \le i, k < n$) means that $S[i]$ obtains the input-data item $Id[i, j]$ from $S[k]$. If $DS[i](j) = -1$, then $Id[i, j] = (-1, d_x)$ and $S[i]$ obtains the associated data from the "closest" device where d_x is initially stored. The set of data-source functions DS corresponds to the data relocation problem discussed in Section 1. For different scheduling functions (as well as assignment functions), with consideration of the data-reuse and multiple data-copies situations, there are different sets of choices for the data-source functions. Thus, the communication time of an application program P depends on both Sf and DS.

3 Impact of Multiple Data-Copies and Temporally Interleaved Input Operations

In this section, both the data-reuse and multiple data-copies situations are considered. Furthermore, data-reuse is viewed as a special case of having multiple data-copies.

It can be shown that, when data-reuse is considered and the multiple data-copies situation is not, the communication time of any application program P depends only on the assignment function Af when the subtasks' computation and communication are assumed to be totally ordered in time [20]. But when one considers the multiple data-copies situation, the communication time of P, in general, also depends on the scheduling function Sf and the set of data-source functions DS. Each scheduling function Sf defines a set of possible choices for DS.

Recall from Section 1 that the size of a data item $e = (k, d)$ is denoted by $|e|$ (or $|d|$). To show the effect of utilizing the multiple data-copies, consider an HC system with four machines connected by the network illustrated in Fig. 2 (vertices denote machines and edges denote communication links). The number in a box with each link represents the communication cost for obtaining the corresponding data item. The order for executing the data transfers is indicated by the numbers in the circles. An initial data element d_0 is stored on machine $M[0]$, and is the required input-data item for both $S[0]$ and $S[1]$. Assume that $Af(0) = 1$ and $Af(1) = 2$. If $S[0]$'s computation is performed before $S[1]$'s computation, with respect to the data transfers illustrated by the arrows in Case 1 of Fig. 2, the total communication time for transferring d_0 is 205. If $S[0]$'s computation is performed after $S[1]$'s computation, with respect to the data transfers illustrated by the arrows in Case 2 of Fig. 2, the total communication time for transferring d_0 is 305. Hence, depending on which scheduling function (and, in general, which set of data-source functions) is chosen, the communication time of an application program P may be different.

It is assumed, without loss of generality, that all input-data items are received for a subtask prior to that subtask's computation. For an arbitrary $S[i]$, there are $NI[i]$ necessary operations for obtaining the input-data items in $I[i]$. These operations are defined as the atomic input operations of $S[i]$. The scheduling function Sf only represents the order for performing the subtasks' computation, *not* the order for performing the atomic input operations. Most of the existing algorithms for matching and scheduling for HC only

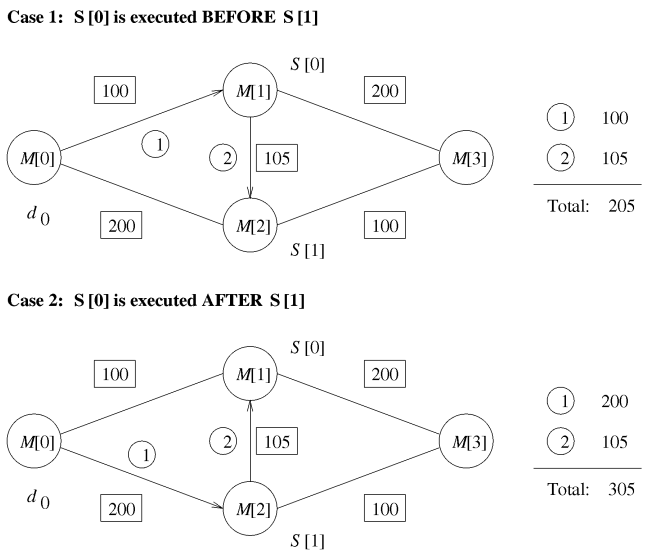
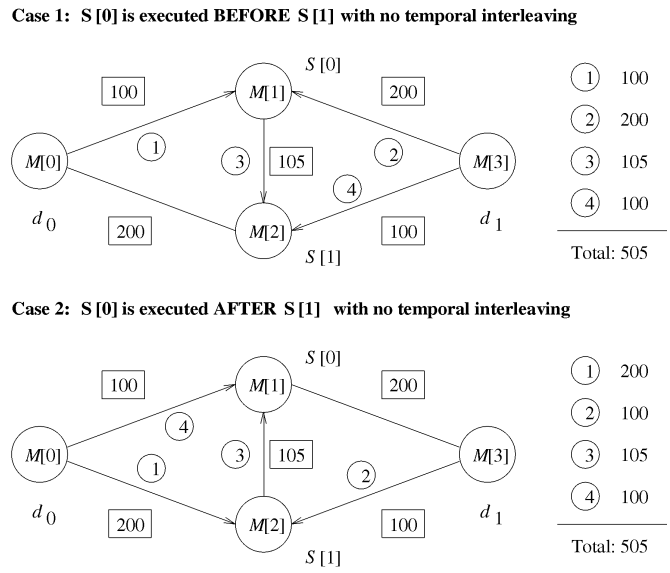


Fig. 2. Network of four machines with the initial data element d_0 on $M[0]$.



Case 3: Temporally interleaved execution of atomic input operations

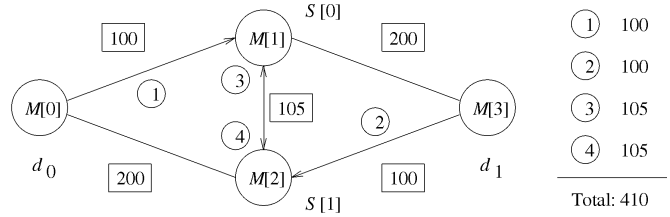


Fig. 3. Network of four machines with initial data elements on $M[0]$ and $M[3]$.

allow consecutive ordering of the atomic input operations of each subtask. This means that if subtask $S[i_1]$'s computation is performed before $S[i_2]$'s computation according to the scheduling scheme, then all atomic input operations of $S[i_1]$ must be performed *before* the atomic input operations of $S[i_2]$ are done. The temporally interleaved execution of atomic input operations for different subtasks (TIE) allows some of the atomic input operations of $S[i_1]$ to be performed *after* some atomic input operations of $S[i_2]$ are executed even if $Sf(i_1) < Sf(i_2)$. The effective use of TIE can result in a smaller execution time than that associated with considering all $NI[i]$ atomic input operations of $S[i]$ to be indivisible. This is true because TIE gives more options for choosing the set of data-source functions for $S[i]$.

As an example, for the same HC system and the same assignment function Af described in Fig. 2, assume now that two input-data items $(-1, d_0)$ and $(-1, d_1)$ are required input-data items of both $S[0]$ and $S[1]$. As shown in Fig. 3, these data items are initially stored on $M[0]$ and $M[3]$, and it is assumed that $|d_0| = |d_1|$. If $S[0]$'s computation is performed before $S[1]$'s computation, the total communication time for transferring d_0 and d_1 is 505 based on the data transfers illustrated by the arrows in Case 1 of Fig. 3. If $S[0]$'s computation is performed after $S[1]$'s computation, the total communication time for transferring d_0 and d_1 is also 505 based on the data transfers illustrated by the arrows in Case 2 of Fig. 3. But, if TIE is allowed, suppose the atomic input operation for $S[0]$ to obtain d_0 is performed first, then the atomic input operation for $S[1]$ to obtain d_1 is performed second, followed by the atomic input operation for $S[0]$ to obtain d_1 and the atomic input operation for $S[1]$ to obtain d_0. In this case, the total communication time for transferring d_0 and d_1 to both $S[0]$ and $S[1]$ is 410.

A set of ordering functions $\underline{Order} = \{Order[i] \mid 0 \le i < n\}$ is associated with the application program P. $Order[i]$ defines the order for performing the atomic input operations of subtask $S[i]$ with respect to all of the atomic input operations for the entire program. If $Order[i](j) = k$, where $0 \le j < NI[i]$ and

$$0 \le k < \sum_{i=0}^{n-1} NI[i],$$

then the jth atomic input operation of $S[i]$ (to obtain the input-data item $Id[i, j]$) is the kth atomic input operation to be performed during the execution of P. Depending upon the network topology and the machine features of the HC system, it is possible in the general case for certain machines in the HC suite to fetch multiple data items from different sources in an overlapping fashion. In such a case, the values of $Order[i]$ are based on the increasing order of the starting times of the atomic input operations of all the subtasks. In contrast with the scheduling function Sf, the set of ordering functions $Order$ schedules the atomic input operations for all subtasks rather than scheduling subtasks' computation. $Order$ and Sf together specify the order of the execution steps (i.e., both the communication and computation steps) of the application program P on an HC system.

The usual definition of scheduling implicitly assumes that the atomic input operations (corresponding to communication) and computation of $S[i]$ are performed indivisiblely. However, interleaved communication (i.e., the atomic input operations of subtasks) may result in smaller total communication time. Thus, extending the standard definition of scheduling to include TIE generally allows for enhanced performance of the HC system. The set of ordering

functions, *Order*, defines the interleaving of the execution of atomic input operations for the subtasks in a program and is used in conjunction with the regular scheduling function *Sf*. As stated in Section 1, it is assumed that each subtask $S[i]$ can keep a copy of each of its individual input-data items and output-data items for forwarding to other subtasks. Based on *Order* and *DS*, it can be determined when a subtask no longer needs to retain a copy of a given data item, so it can be deleted from storage.

4 Totally Ordered Subtasks and Communication Steps

4.1 Generation of Spanning Trees

For the type of HC systems presented in this section, the subtasks' computation and communication of *P* are assumed to be totally ordered in time. For a given computation matrix *C* and communication function matrix $D(|e|)$, the total execution time of the application program *P* associated with an assignment function *Af*, a valid scheduling function *Sf*, and a set of data-source functions *DS* is defined by the following formula:

$$\begin{aligned}&\text{Execution\_time}_P(Af, Sf, DS) = \\ &\text{Computation\_time}_P(Af, Sf, DS) + \\ &\text{Communication\_time}_P(Af, Sf, DS).\end{aligned}$$

such that,

$$\begin{aligned}\text{Computation\_time}_P(Af, Sf, DS) &= \sum_{i=0}^{n-1} C\big[i, Af(i)\big] \\ &= \text{Computation\_time}_P(Af)\end{aligned}$$

and

$$\begin{aligned}&\text{Communication\_time}_P(Af, Sf, DS) = \\ &\sum_{i=0}^{n-1} \sum_{j=0}^{NI[i]-1} D\Big[Af\big(DS[i](j)\big), Af(i)\Big]\Big(\big|Id[i, j]\big|\Big).\end{aligned}$$

This shows that $\text{Computation\_time}_P$ is actually independent of *Sf* and *DS*. Although the dependence of $\text{Communication\_time}_P$ on *Sf* is not explicitly shown in the above equation, the possible sets of data-source functions *DS* depend on *Sf* (see Definition 9 in Section 2). Thus, $\text{Communication\_time}_P$ does indeed depend on *Sf*.

When using TIE, the concept of a *valid* set of data-source functions *DS* for the atomic input operations of the application program *P* can be defined according to the properties of a constructed graph, defined below as *Gr*[*Af, DS*]. There may be many such valid sets, each corresponding to a unique graph, and each resulting in a $\text{Communication\_time}_P$ that may be different from the others. An invalid *DS* would correspond to a set of data-source functions that does not result in an operational program (e.g., in Fig. 3, the case where $S[0]$ receives d_0 from $S[1]$, $S[1]$ receives d_0 from $S[0]$, and neither receives d_0 from $M[0]$ is not valid). A procedure for generating a graph, denoted as *Gr*[*Af, DS*], corresponding to a particular *DS* and assignment function *Af* is described below. Based on the properties of this graph, the validity of a set of data-source functions *DS* can be determined.

Generation of *Gr*[*Af, DS*]

Step 1: A *Source* vertex is generated that represents the locations for all the initial data elements (which may be on different devices / machines).

Step 2: For each $S[i]$, $NI[i] + 1$ vertices, one for each of the $NI[i]$ atomic input operations and one for all of the generated output-data items of $S[i]$, are created. These are the set of input-data vertices, labeled $V[i, j]$ $(0 \le j < NI[i])$ and the output-data vertex $V_g[i]$. *V* is a set that contains all the above vertices associated with the application program *P* in Steps 1 and 2.

Step 3: Let *W* denote the maximum communication time necessary to transfer any data item from an initial source or machine in the heterogeneous suite to any other machine (this can be determined from *H* and *D* defined in Section 2).

Step 4: For each input-data vertex $V[i_1, j_1]$, let $DS[i_1](j_1) = i_2$, where $-1 \le i_2 < n$.

Case A: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from the *Source* vertex if $Id[i_1, j_1] = (-1, d_k)$ and d_k is one of the initial data elements.

In this case, $i_2 = -1$ and there exists k $(0 \le k < Q)$ such that $Id[i_1, j_1] = (-1, d_k)$. Add a directed edge with weight $H[k, Af(i_1)]$ from the *Source* vertex to $V[i_1, j_1]$.

Case B: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from the subtask that generates $Id[i_1, j_1]$.

In this case, $0 \le i_2 < n$ and there exists j_2 such that $Id[i_1, j_1] = Gd[i_2, j_2]$. Add a directed edge with weight $D[Af(i_2), Af(i_1)](|Id[i_1, j_1]|)$ from $V_g[i_2]$ to $V[i_1, j_1]$.

Case C: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from one of the other subtasks that have obtained that input-data item already.

In this case, $0 \le i_2 < n$ and there exists j_2 such that $Id[i_1, j_1] = Id[i_2, j_2]$. Add a directed edge with weight $D[Af(i_2), Af(i_1)](|Id[i_1, j_1]|)$ from $V[i_2, j_2]$ to $V[i_1, j_1]$.

Step 5: For every $0 \le i < n$, a directed edge with weight $W + 1$ (i.e., a weight greater than any possible communication time) is added from $V[i, 0]$ to $V_g[i]$. Thus, $V_g[i]$ also has exactly one parent vertex, i.e., $V[i, 0]$.

For any input-data vertex $V[i_1, j_1]$ $(0 \le i_1 < n$ and $0 \le j_1 < NI[i_1])$, exactly one case above (A, B, or C) can occur. Thus, any vertex $V[i_1, j_1]$ has exactly one parent vertex. Also, the weight of the edge to $V[i_1, j_1]$ from its unique parent vertex is the communication time for $S[i_1]$ to obtain $Id[i_1, j_1]$ with respect to a given *Af* and *DS*.

As an example, suppose that a specific application program *P* is illustrated by the subtask flow graph shown in Fig. 4 with corresponding parameters listed in Table 1, when d_0 and d_1 are the names of initial data elements of *P*; X_0, X_1, Y, Z_0, and Z_1 are the names of generated data items of *P*; and a is an arbitrary constant. Note that initial data elements are named with lower case letters and generated data items with upper case letters.

TABLE 1
PARAMETERS FOR THE SUBTASK FLOW GRAPH SHOWN IN FIG. 4

subtask	no. inputs	input data items	no. outputs	output data items	data size
$S[0]$	$NI[0] = 1$	$Id[0, 0] = (-1, d_0)$	$NG[0] = 2$	$Gd[0, 0] = (0, X_0)$ $Gd[0, 1] = (0, X_1)$	$\|d_0\| = 2a$ $\|d_1\| = 6a$ $\|X_0\| = 8a$ $\|X_1\| = 3a$ $\|Y\| = 5a$ $\|Z_0\| = 4a$ $\|Z_1\| = a$
$S[1]$	$NI[1] = 2$	$Id[1, 0] = (-1, d_0)$ $Id[1, 1] = (0, X_0)$	$NG[1] = 1$	$Gd[1, 0] = (1, Y)$	
$S[2]$	$NI[2] = 2$	$Id[2, 0] = (0, X_0)$ $Id[2, 1] = (-1, d_1)$	$NG[2] = 2$	$Gd[2, 0] = (2, Z_0)$ $Gd[2, 1] = (2, Z_1)$	
$S[3]$	$NI[3] = 3$	$Id[3, 0] = (-1, d_1)$ $Id[3, 1] = (1, Y)$ $Id[3, 2] = (2, Z_0)$	$NG[3] = 0$		
$S[4]$	$NI[4] = 2$	$Id[4, 0] = (0, X_1)$ $Id[4, 1] = (2, Z_1)$	$NG[4] = 0$		
$S[5]$	$NI[5] = 2$	$Id[5, 0] = (0, X_1)$ $Id[5, 1] = (2, Z_0)$	$NG[5] = 0$		

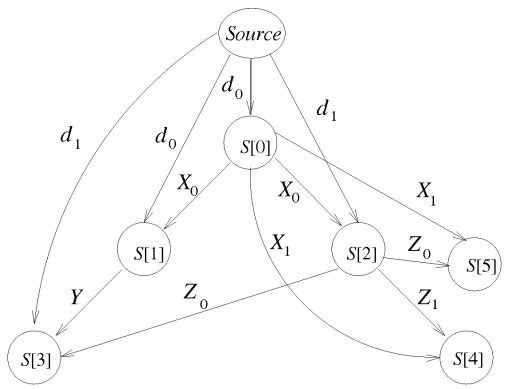


Fig. 4. Subtask flow graph for the example application program

Consider, for ease of presentation, an HC system with four machines connected by the very simple linear array network illustrated in Fig. 5c. Here, $D[s, r](|d|) = |s - r| |d| L$, where $0 \le s, r < 4$ and L is the length of the physical link between the neighboring machines in the linear array network. This model for D is an oversimplified example; any appropriate equation that represents the communication costs of the network in the HC system can be used. The result of applying the set of data-source functions defined by the subtask flow graph in Fig. 4 is shown in Figs. 5a and 5b. The solid lines in Fig. 5a show the directed edges added by applying Steps 4 and 5. The assignment function Af for this current example is shown in Fig. 5c: $Af(0) = 1$, $Af(1) = 2$, $Af(2) = 2$, $Af(3) = 1$, $Af(4) = 3$, and $Af(5) = 0$. W is $24aL$ according to Step 3.

If $Gr[Af, DS]$ generated above by Steps 1 to 5 with respect to any given Af, DS, and P is a tree (denoted as *Tree*$[Af, DS]$) with the *Source* vertex being the root of the tree, then the corresponding DS is defined as a valid set of data-source functions for atomic input operations of the application program P. Any DS that is not such a tree is invalid. The DS defined in Fig. 5b is a valid set of data-source functions (i.e., the solid edges in Fig. 5a form a tree).

The reason for this definition is that, for a *valid* set of data-source functions DS, $Gr[Af, DS]$ must be an acyclic graph. Otherwise deadlock arises in the application program P, which makes P unschedulable (recall the earlier example of an invalid DS). Because a $Gr[Af, DS]$ generated with respect to a *valid* DS is acyclic and each vertex (except the *Source* vertex) of $Gr[Af, DS]$ has exactly one parent vertex, from basic graph theory [3], $Gr[Af, DS]$ is a tree with the *Source* vertex as the root of the tree. Thus, the validity of the corresponding DS can be determined according to whether the $Gr[Af, DS]$ generated by above Steps 1 to 5 is a tree. Furthermore, with an arbitrary assignment function Af and a valid set of data-source functions DS, the weight of the edge to $V[i_1, j_1]$ ($0 \le i_1 < n$ and $0 \le j_1 < NI[i_1]$) from its unique parent vertex is the communication time for $S[i_1]$ to obtain $Id[i_1, j_1]$ with respect to the given Af and DS. Thus, the communication time for the application program P is only a function of Af and DS (DS must be valid) and

$$\text{Communication\_time}_P(Af, DS) = \text{Weight}\left(Tree[Af, DS]\right) - n(W + 1),$$

where Weight(x) is the sum of the weights on all edges of tree x. For the application program P specified by Fig. 4, with respect to the given assignment function Af and the given valid data-source functions DS as defined in Fig. 5b, $\text{Communication\_time}_P(Af, DS) = 67aL$.

To determine a set of ordering functions, *Order*, corresponding to a valid DS for executing the atomic input operations of different subtasks, a directed edge with weight zero from $V[i_1, j_1]$ to $V_g[i_1]$ is added to the *Tree*$[Af, DS]$ for every i_1 and j_1 except $j_1 = 0$ (i.e., $0 \le i_1 < n$ and $1 \le j_1 < NI[i_1]$). These directed edges are illustrated by the dashed lines shown in Fig. 5a for the example application program P.

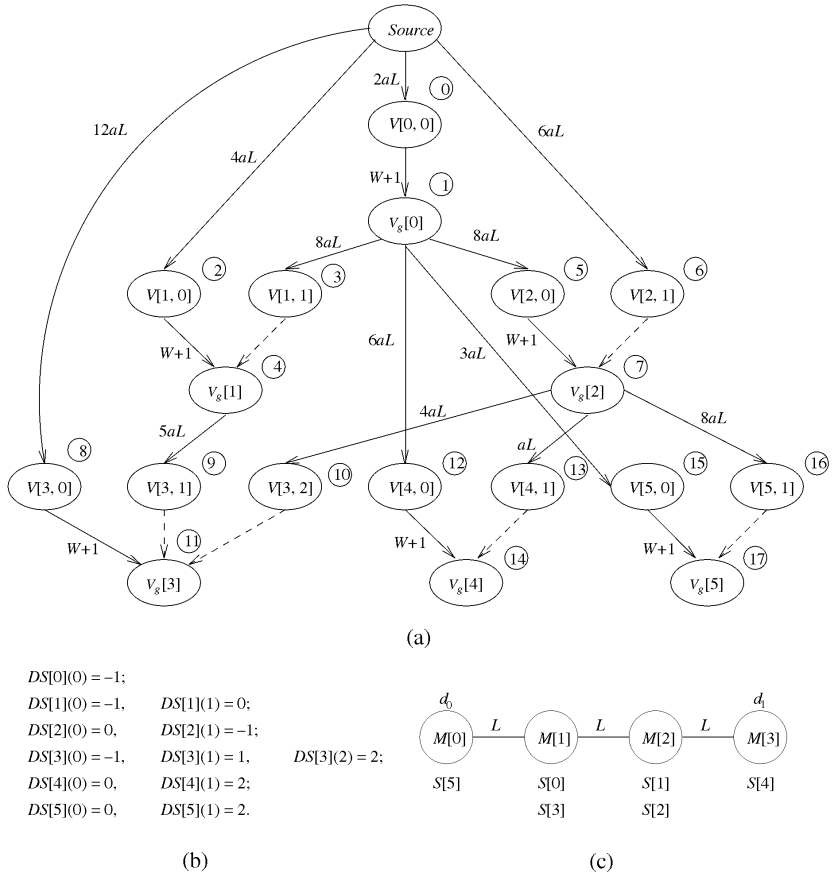


Fig. 5. Generating a spanning tree with respect to the set of data-cource functions associated with the subtask flow graph: (a) the spanning tree (solid lines), (b) the set of data-source functions, (c) the linear array network and the matching scheme.

After adding these zero-weight edges, the tree becomes a directed acyclic graph (DAG). One possible set of ordering functions *Order* corresponding to *DS* can be determined by applying a topological sort algorithm [6] to this generated DAG. For the example application program *P*, the numbers in the circles in Fig. 5a indicate one ordering for the execution of the corresponding atomic input operations and subtask execution of *P* as determined by one particular topological sort.

It is stated at the beginning of this section that Communication_time$_P$ is a function of *Af*, *Sf*, and *DS*. If TIE is allowed, because *Order* is a generalization of *Sf*, Communication_time$_P$ depends on *Af*, *Order*, and a valid *DS*. *Order* must be one of the sets of ordering functions, generated by the topological sort described above, corresponding to a valid *DS*. Otherwise, the scheduling scheme and the data relocation scheme are incompatible with one another (i.e., *Order* and *DS* collectively cannot result in an operational program). If $Order_1$ and $Order_2$ are two sets of ordering functions, then, because Communication_time$_P$(*Af*, *DS*) = Weight(*Tree*[*Af*, *DS*]) − $n(W + 1)$, Communication_time$_P$(*Af*, $Order_1$, *DS*) = Communication_time$_P$(*Af*, $Order_2$, *DS*). Thus, if TIE is allowed and the corresponding *DS* is a valid set of data source functions for the atomic input operations of the application program *P*, Communication_time$_P$ is a function of *Af* and *DS* only (assuming a valid *Order* is used, based on a topological sort of the generated DAG). Because the computation time for *P* is a function of only *Af*, the total execution time for *P* is a function of *Af* and *DS*. The objective of matching, scheduling, and data relocation for HC in Subsection 4.2 is to find an assignment function *Af** and a valid set of data-source functions *DS**, such that

$$\text{Execution\_time}_P(Af^*, DS^*) = \min_{Af, DS}\{\text{Execution\_time}_P(Af, DS)\}.$$

4.2 Finding the Optimal Set of Data-Source Functions

4.2.1 Description of the Algorithm

For a given assignment function *Af*, a minimum spanning tree based algorithm is presented for finding a corresponding optimal valid set of data-source functions *DS**, such that for any other valid set of data-source functions *DS*,

$$\text{Execution\_time}_P(Af, DS^*) \le \text{Execution\_time}_P(Af, DS).$$

A directed graph *Dg* (see Fig. 6) corresponding to a specific assignment function *Af* can be generated by connecting the vertices in *V* as follows (recall that *V* is a set that contains all the vertices generated for any specific application program *P* according to Steps 1 and 2 described in Subsection 4.1). Fig. 6, which is based on the example program shown in Fig. 4, uses the same machine and assignment function as in Fig. 5c, and has all the same vertices as in Fig. 5a.

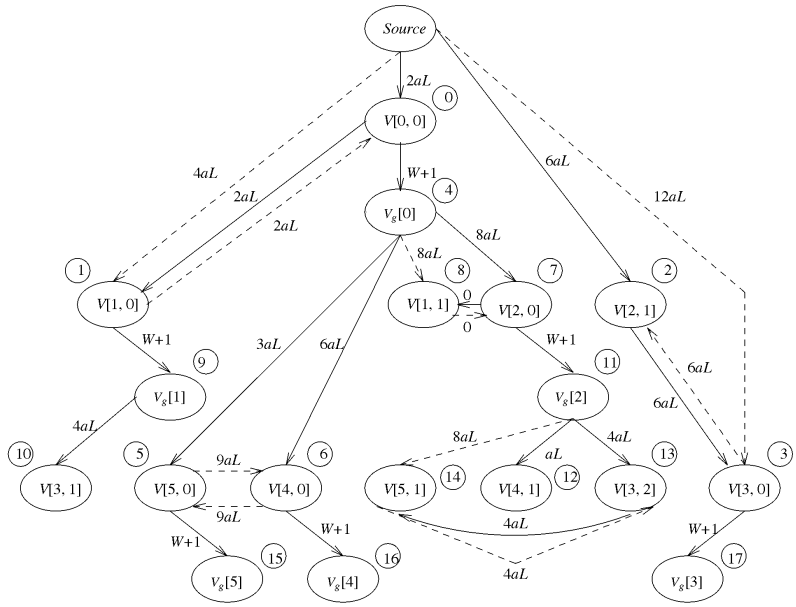


Fig. 6. Generating a minimum spanning tree for the example application program and its corresponding valid data-source functions.

1) For every i_1, j_1, i_2, and j_2, where $0 \le i_1, i_2 < n$, $0 \le j_1 < NI[i_1]$, $0 \le j_2 < NI[i_2]$, and $i_1 \ne i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, a directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ with weight $D[Af(i_1), Af(i_2)](|e|)$ and a directed edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ with weight $D[Af(i_2), Af(i_1)](|e|)$ are added.
2) For every i_1, j_1, i_2, and j_2, where $0 \le i_1, i_2 < n$, $0 \le j_1 < NG[i_1]$, and $0 \le j_2 < NI[i_2]$, such that $Gd[i_1, j_1] = Id[i_2, j_2] = e$, a directed edge from $V_g[i_1]$ to $V[i_2, j_2]$ with weight $D[Af(i_1), Af(i_2)](|e|)$ is added.
3) For every i, j, and k, such that $Id[i, j] = (-1, d_k)$, where $0 \le i < n$, $0 \le j < NI[i]$, and $0 \le k < Q$, a directed edge from the *Source* vertex to $V[i, j]$ with weight $H[k, Af(i)]$ is added.

All the edges generated in 1), 2), and 3) are called fetch edges. For the example application program P illustrated by the subtask flow graph in Fig. 4, with the linear network of four machines as the heterogeneous suite and the assignment function defined in Fig. 5c, the edges (both solid lines and dashed lines) of Dg in Fig. 6 (except the ones with weight $W + 1$) are fetch edges.

4) For every $0 \le i < n$, a directed edge from $V[i, 0]$ to $V_g[i]$ with weight $W + 1$ is added.

All the edges generated in 4) are called activate edges. There are a total of n activate edges with total weight $n(W + 1)$. Notice that the weight of an activate edge is larger than the weight of any fetch edge because of the definition of W. The edges of Dg shown in Fig. 6 with weight $W + 1$ are activate edges.

For given system parameters D and H, the directed graph Dg can be generated by knowing only P and Af. After generating Dg corresponding to a specific Af, a modified version of Prim's algorithm [6], referred to as the TIE algorithm in this paper, is applied to find a minimum spanning tree $MST[Af]$ of Dg. The *Source* vertex is the root of the minimum spanning tree. Suppose A is a set that contains the vertices that have been added to the tree, and T is the partial tree generated during the execution of the TIE algorithm. The order of execution for the atomic input operation that corresponds to any vertex $V[i, j]$ ($0 \le i < n$ and $0 \le j < NI[i]$) in V is $Order^*[i](j)$. The TIE algorithm is described as follows.

Step 1: Let $A = \{Source\}$, $T = \{Source\}$, and $Counter = 0$.

Step 2: Case A: If the set of cut edge(s) between A and $V - A$ (a cut edge is an edge that connects a vertex in A and a vertex in $V - A$) contains fetch edge(s), then find a cut edge that has the smallest weight (there might be several, in which case an arbitrary minimum weight edge is chosen). Include that edge in T and move the corresponding vertex $V[i, j]$ that is currently in $V - A$ into A and T. Increment $Counter$ by 1 and set $Order^*[i](j) = Counter$. Because the set of cut edges between A and $V - A$ contains fetch edges and the weight of an activate edge is greater than the weight of any fetch edge, no activate edge can be chosen.

Case B: If the set of cut edges between A and $V - A$ contains only activate edges, these edges will connect to a subset of the V_g vertices. Let this subset be denoted as $\{V_g[i_0], V_g[i_1], ..., V_g[i_j], ..., V_g[i_{u-1}]\}$, where $1 \le u \le n$, $0 \le j < u$, $0 \le i_j < n$, and u is the number of activate edges in that set. It can be shown that there will exist at least one j ($0 \le j < u$) such that all $V[i_j, k]$ ($0 \le k < NI[i_j]$) are already contained in A by previous iterations of the TIE algorithm. Any such $V_g[i_j]$ is defined as a ready-to-execute vertex. Given that the application program is valid and that the set of cut edge(s) between A and $V - A$ only contains activate edges, there is at least one subtask $S[i_j]$ such that all of its input-data vertices $V[i_j, k]$ are already in A. Otherwise, P is not a valid program because it would allow deadlock. Include a ready-to-execute vertex $V_g[i_j]$ in A and T (if there are sev-

eral, an arbitrary one of the ready-to-execute vertices is chosen), and its corresponding activate edge (i.e., the edge from $V[i_j, 0]$ to $V_g[i_j]$) in T. Because all $V_g[i]$ ($0 \le i < n$) are included in the $MST[Af]$ after they become ready-to-execute vertices, $S[i]$ generates all of its output-data items after it obtains all of its input-data items. Unlike Prim's algorithm, the TIE algorithm uses two classes of edges and places a ready-to-execute vertex into T and A. In all other respects, the algorithms are the same. Because each activate edge is the only edge entering a computation vertex (i.e., V_g vertex), all activate edges will eventually become part of the minimum spanning tree. Hence, this modification to Prim's algorithm to create the TIE algorithm still generates a minimum spanning tree.

Step 3: If $A = V$, terminate the algorithm, otherwise, execute Step 2 again.

For the application program P illustrated by the subtask flow graph in Fig. 4, with the linear network of four machines as the heterogeneous suite and the same assignment functions defined in Fig. 5c, the solid lines in Fig. 6 show the $MST[Af]$ corresponding to Af after applying the TIE algorithm to Dg. This $MST[Af]$ was generated by knowing only Af, $I[i]$, and $G[i]$ (for given system parameters D and H).

The optimal valid set of data-source functions DS^* for atomic input operations of the application program P that corresponds to the minimum spanning tree $MST[Af]$ generated above can be determined as follows:

1) If, in $MST[Af]$, the parent vertex of $V[i_1, j_1]$ is $V[i_2, j_2]$, then $DS^*[i_1](j_1) = i_2$.
2) If, in $MST[Af]$, the parent vertex of $V[i_1, j_1]$ is $V_g[i_2]$, then $DS^*[i_1](j_1) = i_2$.
3) If, in $MST[Af]$, the parent vertex of $V[i_1, j_1]$ is the *Source* vertex, then $DS^*[i_1](j_1) = -1$.

Because $MST[Af]$ is a tree, every vertex except the *Source* vertex has exactly one parent vertex, and the value of $DS^*[i_1](j_1)$ for each $0 \le i_1 < n$ and $0 \le j_1 < NI[i_1]$ is unique. The numbers in the circles in Fig. 6 indicate the order in which vertices were added to the minimum spanning tree, which is the order for executing their corresponding atomic input operations and subtask computation. The set of ordering functions, $Order^*[i](j)$, generated by the TIE algorithm corresponds to this order except that the computation vertices (i.e., V_gs) are not included.

For the complexity analysis of the TIE algorithm, suppose that $|\underline{E}|$ is the number of edges in Dg and $|\underline{V}|$ is the number of vertices in Dg. If a Fibonacci heap is used to implement the priority queue in the TIE algorithm, as was done in Prim's algorithm [6], the worst case asymptotic complexity of the algorithm for finding DS^* is $O(|E| + |V| \lg |V|)$. For Dg,

$$|V| = \sum_{i=0}^{n-1} \left(NI[i] + 1\right) + 1 = \sum_{i=0}^{n-1} NI[i] + n + 1.$$

Each vertex $V[i, j]$ is connected to at most n other vertices in Dg. This corresponds to the case where $S[i]$ can obtain its required input-data item $Id[i, j]$ from all the other subtasks in P and from the source where the initial data elements are stored. Each vertex $V_g[i]$ is connected only to $V[i, 0]$. Thus,

$$|E| \le n\sum_{i=0}^{n-1} NI[i] + n.$$

Let

$$\underline{\varepsilon} = \sum_{i=0}^{n-1} NI[i]$$

be the number of edges in the subtask flow graph. Then $|V| = \varepsilon + n + 1$ and $|E| \le n\varepsilon + n$. The worst case asymptotic complexity of the TIE algorithm in terms of ε and n is $O[n\varepsilon + (n + \varepsilon)\lg(n + \varepsilon)]$, where n is the number of subtasks in P.

4.2.2 Proof of Correctness of the Algorithm

It is shown in Subsection 4.1 that with an arbitrary assignment function Af, any valid set of data-source functions DS for atomic input operations of the application program P corresponds to a spanning tree of Dg (denoted as $Tree_P[Af, DS]$). The weight of $Tree_P[Af, DS]$ (denoted as $\text{Weight}(Tree_P[Af, DS])$) is $\text{Communication\_time}_P(Af, DS) + n(W + 1)$.

Thus,

$$\begin{aligned}&\text{Execution\_time}_P(Af, DS) =\\ &\text{Computation\_time}_P(Af) +\\ &\text{Communication\_time}_P(Af, DS) =\\ &\text{Computation\_time}_P(Af) +\\ &\text{Weight}\left(Tree_P[Af, DS]\right) - n(W + 1).\end{aligned}$$

Because

$$\begin{aligned}&\text{Execution\_time}_P(Af, DS^*) =\\ &\text{Computation\_time}_P(Af) +\\ &\text{Weight}\left(Tree_P[Af, DS^*]\right) - n(W + 1) =\\ &\text{Computation\_time}_P(Af) +\\ &\text{Weight}\left(MST[Af]\right) - n(W + 1)\end{aligned}$$

and

$$\text{Weight}\left(MST[Af]\right) \le \text{Weight}\left(Tree_P[Af, DS]\right),$$

it is true that

$$\text{Execution\_time}_P(Af, DS^*) \le \text{Execution\_time}_P(Af, DS).$$

For the application program P illustrated by the subtask flow graph in Fig. 4, if the set of data-source functions DS is determined directly from the subtask flow graph provided, then Execution_time_P is $C[0, 1] + C[1, 2] + C[2, 2] + C[3, 1] + C[4, 3] + C[5, 0] + 67aL$. After applying the algorithm presented in Subsection 4.2.1 and using DS^*, then Execution_time_P is $C[0, 1] + C[1, 2] + C[2, 2] + C[3, 1] + C[4, 3] + C[5, 0] + 47aL$.

5 An Extension to the Application of the TIE Algorithm

In this section, the TIE algorithm is extended for use in a more general computing paradigm than that of Section 4. Here, the computation of multiple subtasks can occur con-

currently on different machines and the subtask computation steps can be overlapped with other subtasks' communication steps for intermachine data transfers. It is assumed that the heterogeneous suite of machines are connected by a single shared bus, such as an Ethernet. Because all machines in the suite use a single shared bus, the communication steps of the entire application program can be modeled as totally ordered in time [5]. Thus, only one atomic input operation can be executed at any instant in time during the execution of a specific application program *P*. Furthermore, as discussed in Section 1, research using this restricted HC model may help toward solving similar problems with a more general HC model.

Recall that in Section 4.2, a total ordering among both the computation and communication steps of subtasks was assumed. In contrast, in this section, it is assumed that multiple subtasks' computation and a single intermachine communication can be performed concurrently. An extension to the application of the TIE algorithm is presented in this section to generate a new set of data-source functions, $\underline{DS'}$, and a new set of ordering functions, $\underline{Order'}$, when *Af* and *Sf* are specified (and fixed), and the initial *DS* and *Order* functions are provided. The total execution time is generally decreased when the new *DS′* and *Order′* functions are applied. Thus, the TIE algorithm is used as a polynomial time heuristic to improve the initially given data relocation and ordering schemes for intermachine communication, with the given known matching and scheduling of subtask computation.

As defined in 6) of Section 2, $D[s, r](|e|)$ includes all the various hardware and software related components of the intermachine communication process for transferring data item *e*, e.g., network latency and the time for data format conversion between $M[s]$ and $M[r]$ when necessary. With the shared bus based network model, the times for transferring the same data item between different pairs of source and destination machines can differ based on an individual machine's network interface hardware features (e.g., data transfer rate for its I/O port and time for data format conversion between two different machines). Thus, the rationale for varying data relocation and scheduling intermachine communication to decrease data transfer times of application programs still applies in this shared-bus environment.

Given *Af*, *Sf*, *DS*, and *Order*, with known values of *C*, *D*, and *H* defined in Section 2, the starting time $\underline{ST(v)}$ and the finishing time $\underline{FT(v)}$ of each atomic input operation (associated with an input-data vertex $v \in V$) and computation step (associated with an output-data vertex $v \in V$) of *P* can be determined by topological-sort based algorithms. The basic idea is to improve the data relocation schemes for all subtasks, which are computed in the order specified by the initial *Sf*. The extension of the application of the TIE algorithm can be described by following steps:

Step 1: The $\underline{Source[0]}$ vertex represents the locations of all the initial data elements (which may be on different devices/machines). Define

$$\underline{t_0} = \min_{0 \le i < n} \left\{ ST\left(V_g[i]\right) \right\},$$

which is the starting time of the earliest subtask whose computation is performed. Then, define the set of input-data vertices $V[0] = \{V[i, j] \mid FT(V[i, j]) \le t_0\}$. By applying steps 1 and 3 (not 2, because there is no generated data items before t_0) described at the beginning of Subsection 4.2.1, a directed graph $\underline{dg[0]}$ consisting of vertices $V[0] \cup \{Source[0]\}$ can be generated by knowing only *P* and *Af*. The TIE algorithm described in Subsection 4.2.1 can be used to generate a minimum spanning tree of $dg[0]$. The corresponding modified *DS′* and *Order′* can be derived for the atomic input operations specified by the vertices in $V[0]$ in the same way that *DS** and *Order** were derived in Subsection 4.2.1.

Step 2: Iterate the general procedure described in Step 1 $n - 1$ times, where *n* is the number of subtasks in *P*. For the *w*th ($1 \le w < n$) iteration, $S[j_w]$ denotes the *w*th subtask to be computed (according to $ST(V_g[i])$, $0 \le i < n$). The $\underline{Source[w]}$ vertex represents the locations for all the initial data elements, all the input-data items, and all the generated data items of all subtasks $S[k]$ ($k = j_q$, where $0 \le q < w$) whose computation has been performed up to this step. Let $\underline{t_w} = ST(V_g[j_w])$ be the starting time of $S[j_w]$'s computation. Then, define the set of input-data vertices $\underline{V[w]} = \{V[i, j] \mid t_{w-1} < FT(V[i, j]) \le t_w\}$. By applying steps 1 and 3 (recall that the locations of all previously generated data items up to *w*th iteration are specified by $Source[w]$) described at the beginning of the Subsection 4.2.1, a directed graph $\underline{dg[w]}$ can be generated by knowing only *P* and *Af*. The TIE algorithm described in Subsection 4.2.1 can be used to generate a minimum spanning tree of $dg[w]$. The corresponding *DS′* and *Order′* can be derived for the atomic input operations specified by the vertices in $V[w]$ in the same way that *DS** and *Order** were derived in Subsection 4.2.1.

Because of the proved optimality of the TIE algorithm shown in Subsection 4.2.2, after the *w*th iteration of above Step 2, the starting time of subtask $S[j_w]$ ($0 \le w < n$) using *DS′* and *Order′* can be no later than that associated with the initial values for these functions. Thus, the total execution time is decreased (or the same) for *P* with respect to the given *Af*, *Sf*, *DS*, and *Order*.

Consider an example for illustrating the functionality of the above steps for improving the data relocation and its corresponding ordering scheme. Assume an HC application, illustrated by the subtask flow graph in Fig. 7 with four subtasks (d_0 and d_1 are the names of initial data elements and X_0 is the name of a generated data item of $S[0]$). Assume that four subtasks are assigned to three machines, let $Af(0) = 1$, $Af(1) = 2$, $Af(2) = 2$, and $Af(3) = 1$, and $C[0, 1] = 3$, $C[1, 2] = 10$, $C[2, 2] = 4$, and $C[3, 1] = 24$. The relevant values of *H* and *D* are as follows: $H[0, 1] = 2$, $H[0, 2] = 4$, $H[1, 2] = 6$; and $D[1, 2](|d_0|) = D[2, 1](|d_0|) = 2$, $D[1, 2](|X_0|) = 8$, and $D[2, 2](|X_0|) = 0$.

Fig. 8a shows the Gantt chart of the application program illustrated by Fig. 7 with the data relocation using the inter-subtask data transfer patterns illustrated by the subtask flow graph. In this example, $S[0]$'s computation is assumed to be performed before $S[3]$'s computation on $M[1]$ and $S[1]$'s computation is assumed to be performed before $S[2]$'s computation on $M[2]$ according to the known *Sf*. The corresponding ordering scheme for the communication

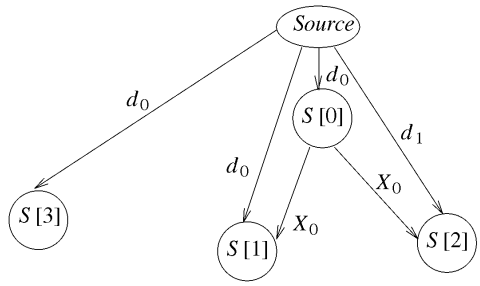


Fig. 7. Subtask flow graph for the example in Section 5.

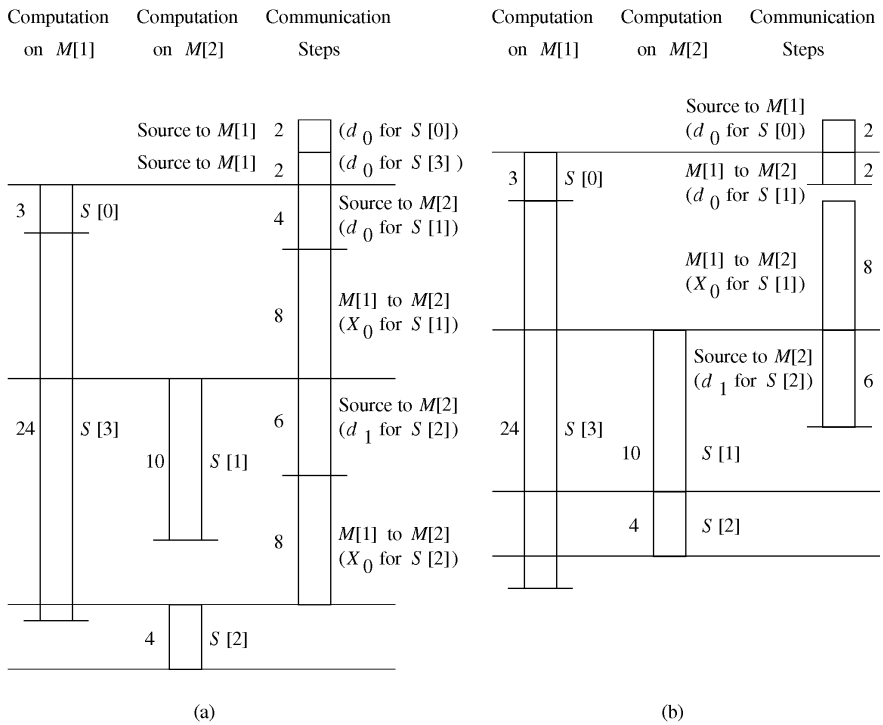


Fig. 8. An example for the extension to the application of the TIE algorithm: (a) Gantt chart with respect to the initial *DS* and *Order*, (b) Gantt chart with respect to *DS′* and *Order′*.

steps is illustrated by the bars on the right side of Fig. 8a. The total execution time is 34. By applying Steps 1 and 2 described above, an improved data relocation scheme is determined (i.e., $DS'[0](0) = -1$, $DS'[1](0) = 0$, $DS'[1](1) = 0$, $DS'[2](0) = 1$, $DS'[2](1) = -1$, $DS'[3](0) = 0$) and its corresponding ordering scheme *Order′* (shown by the bars on the right side of Fig. 8(b)). The total execution time is decreased to 29 using *DS′* and *Order′*. The corresponding Gantt chart is shown by Fig. 8b. Note that the movement of d_0 from the *Source* to $S[0]$ on $M[1]$ also provides d_0 to $S[3]$ (on $M[1]$). Similarly, the movement of X_0 from $S[0]$ on $M[1]$ to $S[1]$ on $M[2]$ also provides X_0 to $S[2]$ (on $M[2]$).

To the best of the authors' knowledge, there is no other work presented in the open literature that utilizes the data relocation and the scheduling of the communication steps to decrease the intermachine communication time of HC applications with a given known matching and scheduling of the subtask computation. Thus, there is no related work with which to make a direct comparison of the approach presented in this section.

Recall that, in this paper, no complete new matching and scheduling subtask computation is generated. Techniques are provided for enhancing the results generated by any given matching and scheduling heuristics, and the optimality of these enhancement techniques are proven mathematically. There is no matching and scheduling heuristic for HC that is widely accepted by the research community as a benchmark to use with the new enhancements as a basis.

Furthermore, even if such a benchmark heuristic existed, there is no generally accepted set of HC benchmark application programs for evaluating heuristics (and enhanced heuristics). It is not clear what characteristics a "typical" HC application would exhibit. Determining a representative set of HC application program benchmarks remains an unsolved challenge for the scientific community in the research field of HC and is outside the scope of this paper.

Due to the above problems, it is difficult to use experimentation to demonstrate the effectiveness of the TIE algorithm from Section 4 and its extension from Section 5. Thus,

in this paper, a theoretical approach is adopted to show the soundness of the TIE algorithm and its extension analytically (versus experimentally). For the totally ordered model used in Section 4, the TIE algorithm is proven to be optimal. For the concurrent computation model used in Section 5, the refinement procedure based on the TIE algorithm is also proven to decrease the total execution time of an HC application. (In pathological worst cases, the TIE algorithm may not be able to improve a particular matching and scheduling, but it will never increase the total execution time.) The exact amount of improvement for a given HC application achieved by varying data relocation and the scheduling of subtasks' communication steps depends on the subtask flow graph of the particular application program, the particular underlying HC system used, and the particular given matching and scheduling of subtasks' computation generated by some other existent heuristic.

6 Summary

In an HC system, the subtasks of an application program P must be assigned to a suite of heterogeneous machines to utilize computational resources effectively (the matching problem). The execution time of P is impacted by the order of execution of subtasks' computation and atomic input operations (the scheduling problem), and the scheme for distributing the initial data elements and the generated data items of P to different subtasks (the data relocation problem).

The intermachine communication time in an HC system can have a significant impact on overall system performance, so techniques that can be used to reduce this time are important. The contributions of this paper are on the techniques for varying scheduling and data relocation schemes to decrease intermachine communication time. In order to use an HC system, a matching scheme must be determined (where the generation of such a matching may involve consideration of scheduling and data relocation). When the subtasks' computation and communication are assumed to be totally ordered in time, the TIE algorithm presented in Section 4 can find in polynomial time the optimal scheduling and data relocation with respect to a given matching. Based on this TIE algorithm, a refinement procedure for the data relocation and the scheduling of intermachine data transfer steps (corresponding to a given matching and scheduling of computation) is presented in Section 5 for HC systems where multiple subtasks' computation and a single intermachine communication are performed simultaneously whenever possible.

With the presence of data-dependent conditional constructs in the subtask flow graph, the post-conditional locations of the input-data items and output-data items of the subtasks inside the "then" and "else" clauses cannot be determined at compile time (i.e., their locations will depend on the value of the conditional and how the clauses are executed at run time). Future work includes applying the methodologies and concepts developed in Sections 4 and 5 to include the data-dependent conditional constructs in the subtask flow graph.

To limit the scope of this paper, the subtasks' computation and communication of a specific application program P were assumed to be totally ordered in time in Section 4 for the presentation of the TIE algorithm and its optimality proof. Section 5 describes one of the possible extensions to the application of the TIE algorithm, where multiple subtasks' computation and a single intermachine communication are performed simultaneously whenever possible. Future research includes applying the concepts developed here to exploit the impact of data-reuse and multiple data-copies on a more general HC environment, in which multiple subtasks' computation and multiple intermachine communication steps can be performed concurrently whenever possible.

Glossary of Notation

Af assignment function (assigns subtasks of application program P to machines)
$C[i, j]$ computation time of subtask i on machine j
Dg directed graph (corresponding to program P) showing data transfer options based on a given Af
d_k kth initial data element of the application program P
$DS[i](j)$ source of the jth input-data item for subtask i
$D[s, r](|e|)$ time for transferring data item e from machine s to machine r
$FT(v)$ finishing time of atomic input operation (associated with an input-data vertex v) and computation step (associated with an output-data vertex v)
$G[i]$ generated output-data set of subtask i
$Gd[i, j]$ jth generated output-data item of subtask i
$Gr[Af, DS]$ generated graph corresponding to a particular DS and Af
$H[k, j]$ minimum time for machine j to obtain the initial data element d_k from one of the devices where d_k is stored before the execution of program P
$I[i]$ input-data set of subtask i
$Id[i, j]$ jth input-data item of subtask i
$M[j]$ jth machine in the HC system, $0 \le j < m$
m number of machines in the HC system
$MST[Af]$ minimum spanning tree of Dg
n number of subtasks in the application program P
$NI[i]$ number of input-data items required by subtask i
$NG[i]$ number of output-data items generated by subtask i
$NS[j]$ number of subtasks assigned to be executed on machine j
$Order[i](j)$ relative order of the jth atomic input operation of subtask i with respect to all the atomic input operations to be performed during the execution of program P
Q number of initial data elements for the application program P
Sf scheduling function associated with the application program P
$S[i]$ ith subtask of an application program P, $0 \le i < n$
$ST(v)$ starting time of atomic input operation (associated with an input-data vertex v) and computation step (associated with an output-data vertex v)
$Tree[Af, DS]$ spanning tree associated with Af and DS
$V_g[i]$ output-data vertex of subtask i
$V[i, j]$ jth input-data vertex of subtask i
W maximum communication time necessary to transfer any data item from an initial source or machine

in the heterogeneous suite to any other machine

ACKNOWLEDGMENTS

The authors thank R. Gupta, G. Krishnamurthy, M. Maheswaran, and J. M. Siegel for their valuable comments. We also thank the anonymous reviewers for their suggestions. A preliminary version of portions of this material was presented at the Fourth Heterogeneous Computing Workshop (HCW '95). This research was supported by Rome Laboratory under contract number F30602-94-C-0022 and NRaD under contract number N66001-96-M-2277.

REFERENCES

[1] I. Ahmad and Y.K. Kwok, "A Parallel Approach for Multiprocessor Scheduling," *Proc. Int'l Parallel Processing Symp.*, pp. 289-293, Apr. 1995.

[2] S.H. Bokhari, "A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System," *IEEE Trans. Software Eng.*, vol. 7, no. 6, pp. 583-589, Nov. 1981.

[3] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*. New York: Elsevier Science, 1976.

[4] S. Chen, M.M. Eshaghian, A. Khokhar, and M.E. Shaaban, "A Selection Theory and Methodology for Heterogeneous Supercomputing," *Proc. Workshop Heterogeneous Processing*, pp. 15-22, Apr. 1993.

[5] M. Cierniak, W. Li, and M.J. Zaki, "Loop Scheduling for Heterogeneity," *Proc. Fourth IEEE Int'l Symp. High-Performance Distributed Computing*, pp. 78-85, Aug. 1995.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. Cambridge, Mass.: MIT Press, 1990.

[7] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Software Eng.*, vol. 15, no. 11, pp. 1,427-1,436, Nov. 1989.

[8] R.F. Freund, "Optimal Selection Theory for Superconcurrency," *Proc. Supercomputing '89*, pp. 699-703, 1989.

[9] R.F. Freund and H.J. Siegel, "Heterogeneous Processing," *Computer*, vol. 26, no. 6, pp. 13-17, June 1993.

[10] A. Ghafoor and J. Yang, "Distributed Heterogeneous Supercomputing Management System," *Computer*, vol. 26, no. 6, pp. 78-86, June 1993.

[11] A. Khokhar, V.K. Prasanna, M. Shaaban, and C.L. Wang, "Heterogeneous Supercomputing: Problems and Issues," *Proc. Workshop Heterogeneous Processing*, pp. 3-12, Mar. 1992.

[12] A. Khokhar, V.K. Prasanna, M. Shaaban, and C.L. Wang, "Heterogeneous Computing: Challenges and Opportunities," *Computer*, vol. 26, no. 6, pp. 18-27, June 1993.

[13] A.E. Klietz, A.V. Malevsky, and K. Chin-Purcell, "A Case Study in Metacomputing: Distributed Simulations of Mixing in Turbulent Convection," *Proc. Workshop Heterogeneous Processing*, pp. 101-106, Apr. 1993.

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

[15] B. Narahari, A. Youssef, and H.A. Choi, "Matching and Scheduling in a Generalized Optimal Selection Theory," *Proc. Heterogeneous Computing Workshop*, pp. 3-8, Apr. 1994.

[16] H.J. Siegel, J.K. Antonio, R.C. Metzger, M. Tan, and Y.A. Li, "Heterogeneous Computing," *Parallel and Distributed Computing Handbook*, A.Y. Zomaya, ed., pp. 725-761. New York: McGraw-Hill, 1996.

[17] H.J. Siegel, H.G. Dietz, and J.K. Antonio, "Software Support for Heterogeneous Computing," *ACM Computing Surveys*, 1996.

[18] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 75-87, Feb. 1993.

[19] H.S. Stone, "Multiprocessor Scheduling with the Aid of the Network Flow Algorithms," *IEEE Trans. Software Eng.*, vol. 3, no. 1, pp. 85-93, Jan. 1977.

[20] M. Tan, J.K. Antonio, H.J. Siegel, and Y.A. Li, "Impact of Data-Reuse and Multiple Data-Copies in a Heterogeneous Computing System with Sequentially Executed Subtasks," TR-EE 95-2, Electrical Eng. School, Purdue Univ., 34 pp., Jan. 1995.

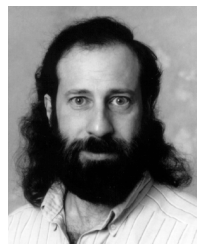
[21] D. Towsley, "Allocating Programs Containing Branches and Loops within a Multiple Processor System," *IEEE Trans. Software Eng.*, vol. 12, no. 10, pp. 1,018-1,024, Oct. 1986.

[22] M. Wang, S.-D. Kim, M.A. Nichols, R.F. Freund, H.J. Siegel, and W.G. Nation, "Augmenting the Optimal Selection Theory for Superconcurrency," *Proc. Workshop Heterogeneous Processing*, pp. 13-22, Mar. 1992.

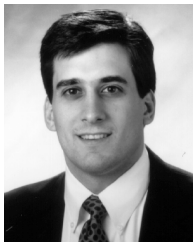
[23] J. Yang, A. Khokhar, S. Sheikh, and A. Ghafoor, "Estimating Execution Time for Parallel Tasks in Heterogeneous Processing (HP) Environment," *Proc. Heterogeneous Computing Workshop*, pp. 23-28, Apr. 1994.

[24] X. Zhang and Y. Yan, "Modeling and Characterizing Parallel Computing Performance on Heterogeneous Networks of Workstations," *Proc. Seventh IEEE Symp. Parallel and Distributed Processing*, pp. 25-34, Oct. 1995.

Min Tan is a PhD candidate in the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana. He attended Shanghai Jiao Tong University, Shanghai, People's Republic of China, in 1988. He received a BA degree in mathematics and physics from Western Maryland College in 1993. In 1994, he received an MS degree in electrical engineering from Purdue University. While at Purdue, he received the "Estus H. and Vashti L. Magoon Outstanding Teaching Award" in 1996. His research interests include data source management in heterogeneous computing, data staging issues for network communication, video compression and financial applications on parallel and distributed systems, and dynamic partitionability for reconfigurable parallel processing machines. He has authored or coauthored 12 technical papers in these and related areas. He is a student member of the IEEE, the IEEE Computer Society, and the Eta Kappa Nu honorary society.



Howard Jay Siegel received two BS degrees from the Massachusetts Institute of Technology (MIT), and the MA, MSE, and PhD degrees from Princeton University. He is a professor and coordinator of the Parallel Processing Laboratory in the School of Electrical and Computer Engineering at Purdue University. He has coauthored more than 230 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing* (second edition, 1990). He is a fellow of the IEEE, was a coeditor-in-chief of the *Journal of Parallel and Distributed Computing*, and has served on the editorial boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He is a member of the steering committee of the annual IEEE Computer Society Heterogeneous Computing Workshop and was general chair in 1997. Prof. Siegel's research and consulting interests include heterogeneous computing, parallel algorithms, interconnection networks, and reconfigurable parallel computer systems. In the area of heterogeneous computing, he is examining ways to match segments of a task to different machines in a heterogeneous suite to exploit the varied computational capabilities available. His algorithm work explores the factors involved in mapping a problem onto a parallel processing system to minimize execution time. Topological properties and fault tolerance are the focus of his research on interconnection networks for large-scale parallel machines. He is analytically and experimentally investigating the utility of the three dimensions of dynamic reconfigurability supported by the PASM design ideas and the small-scale proof-of-concept prototype: mixed-mode parallelism, switchable interprocessor communications, and system partitionability.



John K. Antonio received the BS, MS, and PhD degrees in electrical engineering from Texas A&M University, College Station, Texas. He currently holds the position of associate professor of computer science within the College of Engineering at Texas Tech University. Prior to joining Texas Tech, he was with the School of Electrical and Computer Engineering at Purdue University. During the summers of 1991-1994, he participated in a faculty research program at Rome Laboratory, Rome, New York, where he conducted research in the area of high performance computing. His current research interests include heterogeneous systems, configuration techniques for embedded parallel systems, and computational aspects of control and optimization. He has coauthored more than 50 publications in these and related areas. For the past four years, he has organized the Industrial Track and Commercial Exhibits portions of the IEEE International Parallel Processing Symposium. He is a member of the IEEE Computer Society, and is also a member of the Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi honorary societies. Organizations that have supported his research include the U.S. Air Force Office of Scientific Research, the U.S. National Science Foundation, NRaD, Orincon, Inc., and Rome Laboratory.

Yan Alexander Li received his BE degree from Tsinghua University, Beijing, China, in 1991, and his MSEE and PhD degrees from Purdue University, West Lafayette, Indiana, in 1993 and 1996, respectively. He is currently a senior system architect at Intel Corporation. He is a member of the IEEE and Eta Kappa Nu. His major research interests include parallel processing, high-performance heterogeneous computing, computer architecture, and computer systems simulation.