

1-1-1995

Impact of Data-Reuse and Multiple Data-Copies in a Heterogeneous Computing System with Sequentially Executed Subtasks

Min Tan

Purdue University School of Electrical Engineering

John K. Antonio

Purdue University School of Electrical Engineering

Howard Jay Siegel

Purdue University School of Electrical Engineering

Yan Alexander Li

Purdue University School of Electrical Engineering

Follow this and additional works at: <http://docs.lib.purdue.edu/ecetr>

Tan, Min; Antonio, John K.; Siegel, Howard Jay; and Li, Yan Alexander, "Impact of Data-Reuse and Multiple Data-Copies in a Heterogeneous Computing System with Sequentially Executed Subtasks " (1995). *ECE Technical Reports*. Paper 49.
<http://docs.lib.purdue.edu/ecetr/49>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

IMPACT OF DATA-REUSE AND
MULTIPLE DATA-COPIES IN A
HETEROGENEOUS COMPUTING
SYSTEM WITH SEQUENTIALLY
EXECUTED SUBTASKS

MIN TAN
JOHN K. ANTONIO
HOWARD JAY SIEGEL
YAN ALEXANDER LI

TR-EE 95-2
JANUARY 1995



SCHOOL OF ELECTRICAL ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Impact of Data-Reuse and Multiple Data-Copies in a Heterogeneous Computing System with Sequentially Executed Subtasks

Min Tan
John K. Antonio
Howard Jay Siegel
Yan Alexander Li

Parallel Processing Laboratory
School of Electrical Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285, USA
(mtan, jantonio, hj, yali)@ecn.purdue.edu

January 1995

This research **was** supported by Rome Laboratory under contract number F30602-94-C-0022.

Impact of Data-Reuse and Multiple Data-Copies in a Heterogeneous Computing System with Sequentially Executed Subtasks

Min Tan, John K. Antonio, Howard Jay Siegel, and Yan Alexander Li

	Page
List of Figures.....	iii
Abstract.....	iv
1. Introduction.....	1
2. A Mathematical Model for Matching, Scheduling, and Data Relocation in HC	6
3. Impact of Data-Reuse Without Considering Multiple Data-Copies.....	10
4. Impact of Multiple Data-Copies and Temporally Interleaved Execution of Atomic Input Operations for Different Subtasks (TIE).....	12
5. A Minimum Spanning Tree Based Algorithm for Finding the Optimal Set of Data-Source Functions and the Corresponding Set of Ordering Functions.....	23
5.1 Description of the Algorithm	23
5.2 Proof of Correctness of the Algorithm.....	28
6. Two-Stage Approach for Matching, Scheduling, and Data Relocation in HC	29
7. Summary	30
References.....	32

List of Figures:

- Figure 1:** Data-distribution situations in HC. (a) Subtask-flow graph. (b) Data-reuse. (c) The multiple data-copies situation.
- Figure 2:** Linear array network of four machines with the initial data element d_0 on $M[0]$
- Figure 3:** Linear array network of four machines with the initial data on $M[0]$ and $M[3]$.
- Figure 4:** The generation of the vertices for the atomic operations of $S[i]$.
- Figure 5:** Subtask-flow graph for the example application program.
- Figure 6:** Generating a spanning tree with respect to the set of data-source 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.
- Figure 7:** Generating a minimum spanning tree for the example application program and its corresponding valid data-source functions. (a) The minimum spanning tree (solid lines). (b) The set of data-source functions. (c) The linear array network and the matching scheme.

Abstract

In a heterogeneous computing (HC) environment, an application program is decomposed into subtasks, then each computationally homogeneous subtask is assigned to the machine where it is best suited for execution. 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. A mathematical model is presented for three of the factors that affect the execution time of an application program in an HC system: matching, scheduling, and data relocation schemes. Two data relocation situations are identified, namely data-reuse and multiple data-copies. It is proved that without considering multiple data-copies, but allowing data-reuse, the execution time of given application program depends only on the matching scheme. A polynomial algorithm, which is minimum spanning tree based, is introduced to find the optimal scheduling scheme and the optimal data relocation scheme with respect to an arbitrary matching scheme when data-reuse and multiple data-copies are considered. Finally, a two-stage approach for matching, scheduling, and data relocation in HC is presented.

1. Introduction

A single application program often requires a variety of 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 distinct and varied computational requirements of a given application [FrS93, KhP93, SiA95].

The goal of HC is to decompose an application program into subtasks, and then assign each computationally homogeneous subtask 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 such that the total execution time (computation time and inter-machine communication time) of the application program is minimized. This 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., [ChE93, Fre89, NaY94, WaK92]). A collection of algorithms, called graph-based algorithms in this paper (e.g. [Bok81, NaY94, Sto77, Tow86]), 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 Figure 1(a), each vertex of the subtask-flow graph represents a subtask. Let $S[k]$ denotes the k -th, subtask. There is an edge from $S[k]$ to $S[j]$ labeled with the variable name of the data 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 has already been done.

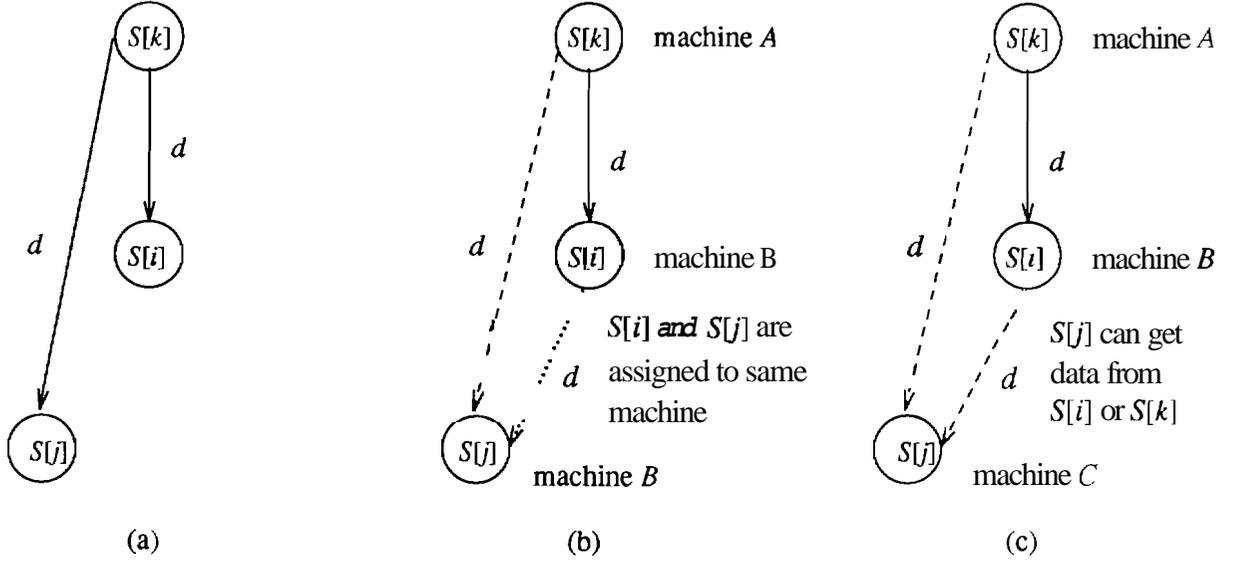


Figure 1: Data-distribution situations in HC. (a) Subtask-flow graph. (b) Data-reuse. (c) The multiple data-copies situation.

Let a data item be a block of information that can be designated to be transferred between subtasks. For example, 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 two-tuple (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 (s, d) , or $s = -1$ 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).

Sequential execution of **subtasks** assumes that at any instant in time during the execution of a specific application program \underline{P} , only one **subtask** of \underline{P} is being executed on any of the machines in the heterogeneous suite. In practice, concurrent execution of **subtasks** is possible, however, the simplifying assumption of **sequentiality** is made here as a step toward solving the more general problem. This simplifying assumption is used by many other researchers as well (e.g., [Bok81, Sto77, Tow86]).

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 a **subtask-flow graph** (e.g., [Bok81, Lo88, NaY94, Sto77, Tow86]). Thus, no matter which machine is used for executing each **subtask** of a specific application program, the decision (derived from the **subtask-flow graph**) of which **subtask(s)** each **subtask** should obtain its corresponding input-data items from is unchanged and independent of any particular matching scheme between machines and **subtasks**.

The above assumption generally needs refinement in the case of HC. Two data-distribution situations arise, namely data-reuse and multiple data-copies. It is assumed that each **subtask** $S[i]$ keeps a copy of each of its individual input-data items and output-data items on the machine to which $S[i]$ is assigned by the matching scheme. Data-reuse arises when two **subtasks**, $S[i]$ and $S[j]$, need the same data item $e = (k, \mathfrak{O})$ from $S[k]$ (as in the example **subtask-flow graph** in Figure 1(a)). For any data item $e = (k, d)$, e represents the value of the associated data and $|e|$ (as well as $|d|$) represents the size of the associated data. As shown in Figure 1(b), suppose the particular matching scheme is the one that assigns $S[k]$ to machine **A**, $S[i]$ to machine **B**, and $S[j]$ to machine **B**. Furthermore, assume for this example that the **subtasks** are executed in the order k, i , and 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 Figure 1(b), 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 Figure 1(b)). If a **subtask-flow graph** is used to compute inter-**subtask** communication cost, then without considering machine assignments, the

impact of data-reuse is ignored.

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 different machines in the HC system. In the example in Figure 1(c), the matching scheme assigns $S[k]$ to machine A, $S[i]$ to machine B, and $S[j]$ to machine C. Therefore, $S[j]$ can get 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, using only information from the subtask-flow graph, the possibility of multiple sources of a needed data item due to a specific matching scheme is not considered.

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 sequence of execution for 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 [SiA95].

The inter-machine communication time between subtasks can be substantial in an HC system. Thus, this inter-machine 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 with a known matching scheme. In particular, the impact of scheduling and data relocation schemes on the communication time of the subtasks executed in sequence are examined.

In Section 2, a mathematical model for matching, scheduling, and data relocation in HC is introduced. Section 3 presents a theorem, which states that, without considering multiple data-copies and with the consideration of data-reuse, the execution time of a given application

program depends only on the specific matching scheme (i.e., it is independent of scheduling). In Section 4, an extension to the usual scheduling methodology is introduced. Specifically, the temporally interleaved execution of the atomic input operations of different **subtasks** (TIE) is considered. When considering multiple data-copies, this extension to scheduling can decrease the execution time of an application program. A minimum spanning tree based algorithm (referred to as the TIE algorithm) is described in Section 5 that finds, for a given matching, 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 and an example is given. Based on this TIE algorithm, a two-stage approach for matching, scheduling, and data relocation in HC is proposed in Section 6.

2. A Mathematical Model for Matching, Scheduling, and Data Relocation in HC

A mathematical model for matching, scheduling, and data relocation in HC is formalized in this section. The model serves as the mathematical basis for Theorem 1 presented in Section 3. The TIE algorithm in Section 5 is given in unambiguous terms based on this mathematical model.

- (1) An application program P is composed of a set of subtasks $\underline{S} = \{ S[0], S[1], \dots, S[n - 1] \}$, 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 $\underline{I}[i] = (Id[i, 0], Id[i, 1], \dots, Id[i, NI[i] - 1])$, the other is the generated output-data set $\underline{G}[i] = \{ Gd[i, 0], Gd[i, 1], \dots, Gd[i, NG[i] - 1] \}$. 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. To satisfy this assumption, a combination of following two approaches is used: (i) a cycle is unrolled (conceptually) and the number of times a cycle repeats is known or estimated (as is typically done by optimizing compilers) or (ii) the whole looping construct is viewed as part of a single subtask and the boundaries for decomposing an application program into subtasks are not allowed to be in the middle of a loop.
- (3) An HC system consists of a heterogeneous suite of machines $\underline{M} = \{ M[0], M[1], \dots, M[m - 1] \}$, 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 $M[j]$ [GhY93, YaK94]. 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 [SiA95] for a survey of these techniques).

- (5) Suppose that a set of initial data elements $\{d_0, d_1, \dots, d_{Q-1}\}$ are required for executing the application program P , where Q is the number of initial data element; for P . A set of initial-data functions $\underline{H} = \{H[0], H[1], \dots, H[Q-1]\}$ is defined, where $H[k](j)$ ($0 \leq k < Q$ and $0 \leq j < m$) represents the least amount of 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 $0 \leq s, r < m$, where $D[s, r](|e|)$ denotes the communication time for transferring data item e (of size $|e|$) from machine $M[s]$ to machine $M[r]$ [GhY93, KhP92]. It is assumed that $D[s, s](|e|) < D[s, r](|e|)$ for $r \neq s$, i.e., the communication time for machine $M[s]$ to fetch any data item from its local storage (denoted as $D[s, s](|e|)$) is smaller than the communication time required to fetch the same data item from any other machine in the heterogeneous suite (denoted as $D[s, r](|e|)$ and $r \neq s$). Having $s = -1$ indicates that $e = (-1, d)$ and d is one of the initial data elements of P and there exists k ($0 \leq k < Q$) such that $d = d_k$ and $D[s, r](|e|) = H[k](r)$.
- (7) An assignment function \underline{A} is associated with the application program P , 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.
- (8) Given that sequential execution of the subtasks for the application program P is assumed, a scheduling function \underline{S} is associated with the application program P . $Sf(i) = k$ means that $S[i]$ is the k -th subtask to be executed. The scheduling function \underline{S} corresponds to the scheduling problem discussed in Section 1. \underline{S} is a bijection from the set S onto itself (i.e., a permutation).

Sf is defined as a valid scheduling function if and only if for all $S[i_1]$ and $S[i_2]$ such that $G[i_1] \cap I[i_2] \neq \emptyset$, $Sf[i_1] < Sf[i_2]$. For the rest of the paper, all scheduling functions considered will be valid. Therefore, if one of the input-data items required by $S[i_2]$ is one of the output-data items generated by $S[i_1]$, then, with respect to a valid scheduling function, $S[i_2]$ must be executed after $S[i_1]$ is executed. If two subtasks have no data dependency between them, then either can be executed before the other.

(9) The set of data-source functions is $\underline{DS} = \{ DS[0], DS[1], \dots, DS[n-1] \}$, where $DS[i](j) = k$ ($0 \leq i, k < n$) 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. When data-reuse is considered, a restriction on DS is made. For any two subtasks $S[i_1]$ and $S[i_2]$, if $Af(i_1) = Af(i_2) = k$, $Sf(i_2) < Sf(i_1)$, and there exists j_1 and j_2 such that $Id[i_1, j_1] = Id[i_2, j_2]$, then $DS[i_1](j_1) = i_2$. That is, if $S[i_1]$ and $S[i_2]$ are assigned to the same machine $M[k]$ by Af , and $S[i_2]$ is executed before $S[i_1]$ according to Sf , and $S[i_2]$ and $S[i_1]$ have a common input-data item (possibly generated from third different subtask), then $S[i_1]$ should take advantage of data-reuse and obtain the common data item from $S[i_2]$ that was executed previously on the same machine $M[k]$. Because each machine in the HC system can fetch any data item from its local storage faster than fetching it from other machines in the HC suite (see definition (6)), this restriction on DS by considering data-reuse is justified.

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 .

(10) 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:

$$\text{Execution_time}_P(Af, Sf, DS) = \\ \text{Computation_time}_P(Af, Sf, DS) + \text{Communication_time}_P(Af, Sf, DS),$$

such that,

$$\text{Computation_time}_P(Af, Sf, DS) = \sum_{i=0}^{n-1} C[i, Af(i)] = \text{Computation_time}_P(Af)$$

and

$$\text{Communication_time}_P(Af, Sf, DS) = \sum_{i=0}^{n-1} \sum_{j=0}^{MI[i]-1} D[Af(DS[i](j)), Af(i)](|Id[i, j]|).$$

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)). Thus, $\text{Communication_time}_P$ does indeed depend on Sf . The objective of matching, scheduling, and data relocation for HC is to find an assignment function Af^* , a valid scheduling function Sf^* , and a set of data-source functions DS^* such that

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

3. Impact of Data-Reuse Without Considering Multiple Data-Copies

The following lemma and theorem use the mathematical model described in the previous section for the case where there are data-reuses. The multiple data-copies situation is not considered in this section.

Lemma 1: When data-reuse is considered and the multiple data-copies situation is not, $DS = f'(Af, Sf)$, where f' is a function of Af and Sf .

Proof: Without considering both data-reuse and multiple data copies, DS is uniquely determined by the underlying given subtask-flow graph and Af . When the data-reuse (only) is considered, then by definition, DS is uniquely determined by the conditions imposed by Af and Sf (see definition (9) in Section 2). Thus, without considering multiple data-copies, DS is a function of Af and Sf . \square

Theorem 1: If data-reuse is considered but the multiple data-copies situation is not, then the execution time of an application program P is a function of Af only, i.e.,

$$\text{Execution-time, } (Af, Sf, DS) = f(Af).$$

Proof: From Lemma 1, DS is a function of Af and Sf . Thus, Execution-time_P is only a function of Af and Sf . It needs to be shown that Execution-time_P is independent of Sf .

As shown in definition (10), because $\text{Computation-time}_P$ is independent of Sf and DS , it needs to be shown that $\text{Communication-time}_P$ is independent of Sf and DS . Recall that the formula for $\text{Communication-time}_P$ is

$$\text{Communication-time}_P(Af, Sf, DS) = \sum_{i=0}^{n-1} \sum_{j=0}^{M[i]-1} D[Af(DS[i](j)), Af(i)](|Id[i, j]|).$$

Case 1: For subtask $S[i]$ and data item $Id[i, j]$, such that $S[i]$ cannot receive $Id[i, j]$ from a subtask on machine $M[Af(i)]$ according to any valid scheduling function Sf (i.e., there is no opportunity for data-reuse for the particular data item $Id[i, j]$ of $S[i]$): As stated in the proof of Lemma 1, with no data-reuse, the value of $DS[i](j)$ (and hence the value of $D[Af(DS[i](j)), Af(i)](|Id[i, j]|)$)

is independent of Sf .

Case 2: For subtask $S[i]$ and data item $Id[i, j]$, such that $S[i]$ can receive $Id[i, j]$ from a subtask on the same machine $M[Af(i)]$ according to an arbitrary scheduling function Sf (i.e., there is opportunity for data-reuse for the particular data item $Id[i, j]$ of $S[i]$): Let $Id[i, j] = (x, d)$, where d is the corresponding variable name and $S[x]$ is the subtask that generates d . Af determines which subtasks are executed on machine $M[Af(i)]$. Let $S' \subseteq S$ be the subset of subtasks that are executed on $M[Af(i)]$ and need the unique input-data item (x, d) . The data item (x, d) must be moved from $M[Af(x)]$ to $M[Af(i)]$ just once, and then can be used by all $S[k] \in S'$. Thus, the communication time for all $S[k] \in S'$ to receive (x, d) from $M[Af(x)]$ is equal to the time for any one of the subtasks in S' to receive (x, d) from $M[Af(x)]$ and the time for other subtasks in S' to fetch (x, d) from local storage with the consideration of data-reuse. Mathematically, if $|S'|$ is the size of the subset S' , and $S[q]$ is an arbitrary element of S' , then the time for all $S[k] \in S'$ to receive (x, d) is:

$$D[Af(x), Af(q)](|(x, d)|) + (|S'| - 1)D[Af(q), Af(q)](|(x, d)|).$$

Therefore, the communication time for all $S[k] \in S'$ to obtain data item (x, d) is independent of the order of execution of the subtasks in S' , and hence is independent of Sf . Thus, as with Case 1, it is shown that $\text{Communication_time}_P$ is independent of Sf .

Therefore, both $\text{Computation_time}_P$ and $\text{Communication_time}_P$ are independent of Sf . Thus, Execution_time_P depends on Af only. □

4. Impact of Multiple Data-Copies and Temporally Interleaved Execution of Atomic Input Operations for Different Subtasks (TIE)

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

It was shown in Theorem 1 that, with the consideration of data-reuse and without considering multiple data-copies, the execution time of any application program P depends only on the assignment function Af . But when one considers the multiple data-copies situation, the execution time of an application program P 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 a linear array network (illustrated by the dashed lines in Figure 2). Here, $D[s, r](|d|) = |s - r| |d|L$, where $0 \leq s, r < 4$ and L is the length of the physical link between the neighboring machines in the linear array network. This equation for D is just an over simplified example: any appropriate equation that represents the communication costs of the network in the HC system can be used. An initial data element d_0 is stored on machine $M[0]$. $(-1, d_0)$ is the only required input-data item for both $S[0]$ and $S[1]$ (thus, there is no data dependency between $S[0]$ and $S[1]$). Assume that $Af(0) = 1$ and $Af(1) = 2$ (thus, Computation-time $_P$ is determined). If $S[0]$ is scheduled for execution before $S[1]$, by the data transfers illustrated by the solid lines in Case 1 of Figure 2, $Communication_time_P = 2|d_0|L$. If $S[0]$ is executed after $S[1]$, by the data transfers illustrated by the solid lines in Case 2 of Figure 2, $Communication_time_P = 3|d_0|L$. Hence, depending on which scheduling function (and, in general, which set of data-source functions) is chosen, the execution 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

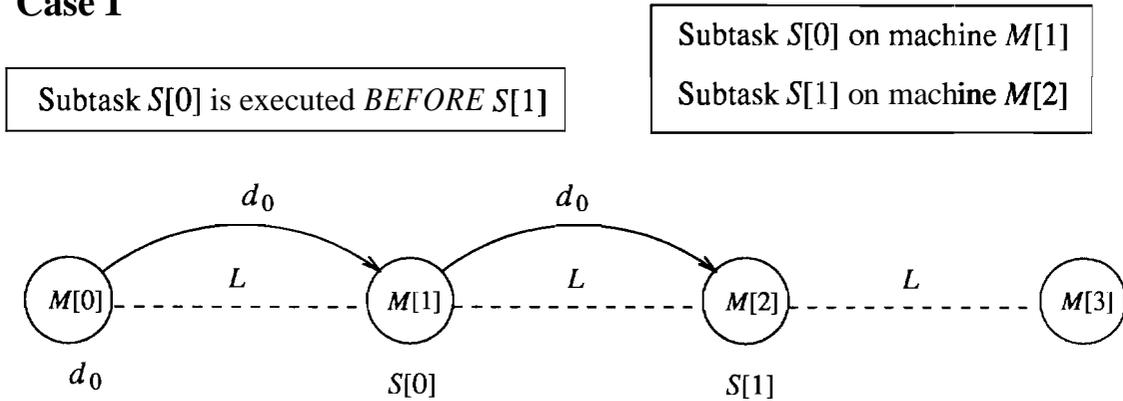
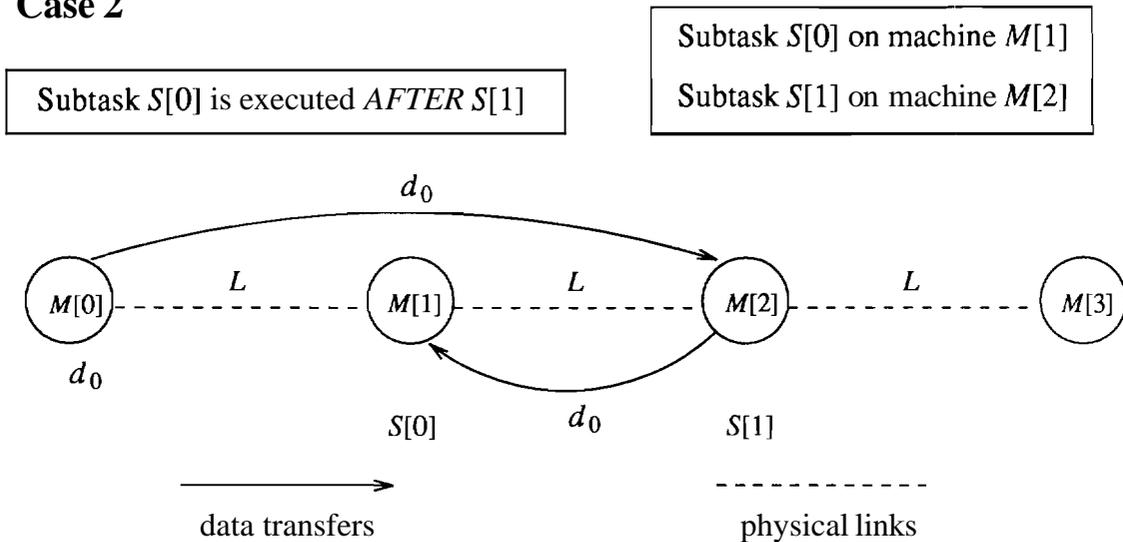
Case 1**Case 2**

Figure 2: Linear array network of four machines with the initial data element d_0 on $M[0]$.

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 executing the subtasks, *not* the order for executing the atomic input operations. Most of the existing algorithms for matching, scheduling, and data relocation in HC only allow consecutive execution of the atomic input operations of each subtask. This means that if $Sf(i_1) < Sf(i_2)$, then all atomic input operations of $S[i_1]$ must be executed *before* the atomic input operations of $S[i_2]$ are executed. The temporally interleaved execution of atomic input operations for different

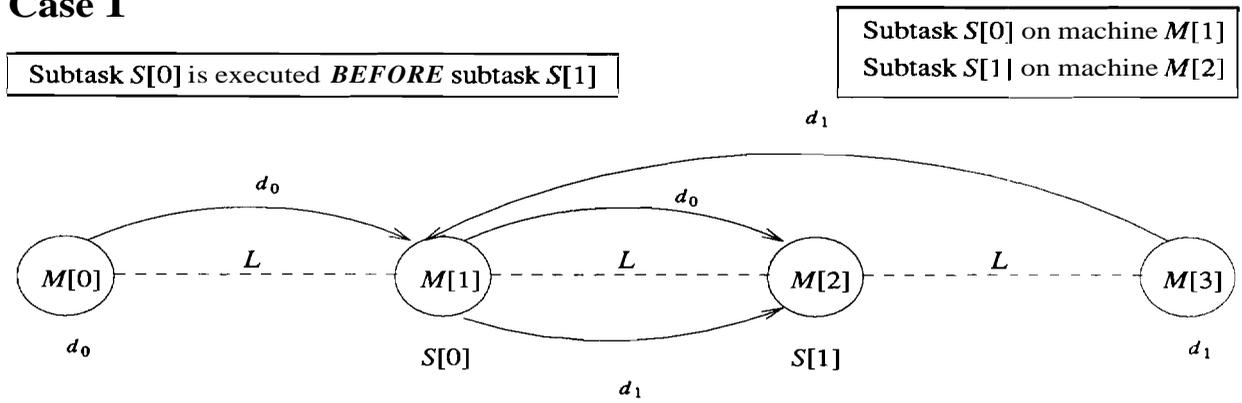
subtasks (TIE) allows some of the atomic input operations of $S[i_1]$ to be executed *after* some atomic input operations of $S[i_2]$ are executed even if $Sf(i_1) < Sf(i_2)$. The effective use of TIE can result in a smaller execution time than that associated with considering the sequence of $NI[i]$ 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 as described by Figure 2, assume that $(-1, d_0)$ and $(-1, d_1)$ are the only required input-data items of both $S[0]$ and $S[1]$ (initially stored on $M[0]$ and $M[3]$ respectively and $|d_0| = |d_1|$). If $S[0]$ is executed before $S[1]$, by the data transfers illustrated by the solid lines in Case 1 of Figure 3, $Communication_time_P = 5|d_0|L$. If $S[0]$ is executed after $S[1]$, by the data transfers illustrated by the solid lines in Case 2 of Figure 3, $Communication_time_P = 5|d_0|L$. But if TIE is allowed, suppose the atomic input operation for $S[0]$ to obtain $(-1, d_0)$ is executed first, then the atomic input operation for $S[1]$ to obtain $(-1, d_1)$ is executed second, followed by the atomic input operation for $S[0]$ to obtain $(-1, d_1)$ and the atomic input operation for $S[1]$ to obtain $(-1, d_0)$ (the order for executing the atomic input operations of $S[0]$ and $S[1]$ is indicated by the numbers in the circles in Case 3 of Figure 3), then $Communication_time_P = 4|d_0|L$. For all three cases, the same Af is used, and hence $Computation_time_P$ is the same.

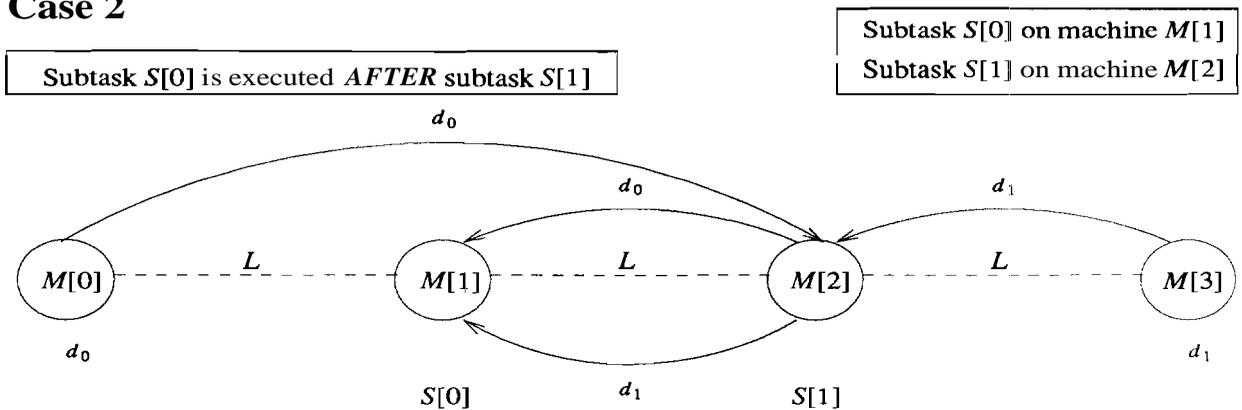
A set of ordering functions $\underline{Order} = \{ Order[i] \mid 0 \leq i < n \}$ is associated with P . If $Order[i](j) = k$, where $0 \leq j < NI[i]$ and $0 \leq k < \sum_{i=0}^{n-1} NI[i]$, then the j -th atomic input operation of $S[i]$ (to obtain the input-data item $Id[i, j]$) is the k -th atomic input operation to be executed during the execution of P .

The usual definition of scheduling implicitly assumes that the atomic input operations (corresponding to communication) and computation of $S[i]$ are executed indivisibly. Suppose the execution steps of two or more subtasks are interleaved and the concept of sequential execution of subtasks (i.e., no concurrent execution of different subtasks across different machines in the HC suite) is still enforced. Given the mathematical model presented in Section

Case 1



Case 2



Case 3

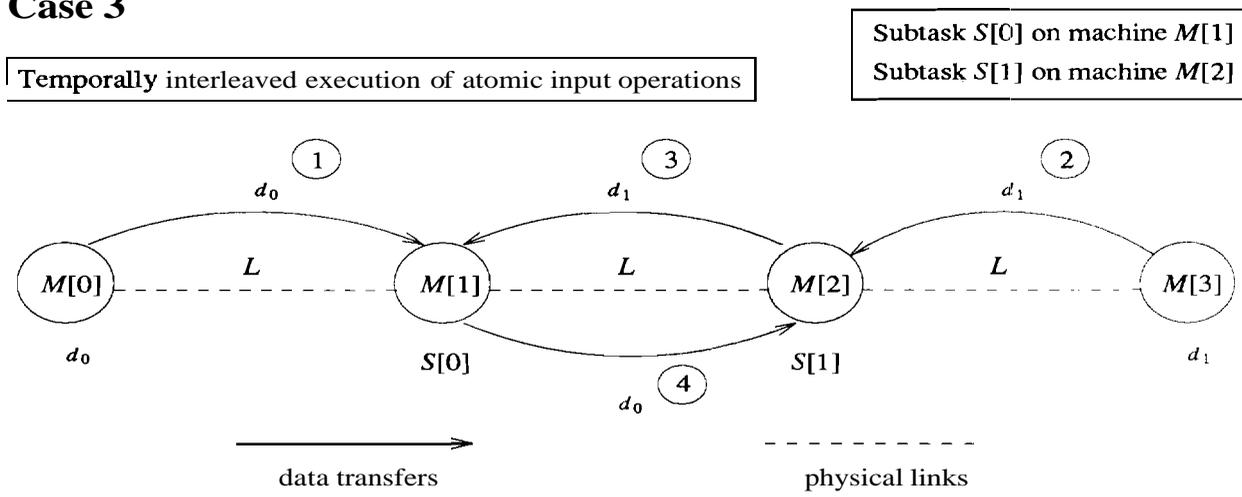


Figure 3: Linear array network of four machines with the initial data on $M[0]$ and $M[3]$.

2, the interleaved computation of subtasks cannot change the total computation time (which is determined by Af). However, interleaved communication (i.e., the atomic input operations of subtasks) may result in smaller total communication time. This is the effect that TIE is trying to exploit. Thus, extending the definition of scheduling function Sf to allow TIE can potentially enhance the performance of the corresponding 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 an extension to the regular scheduling function Sf .

In the following Steps 1 to 4, a graph (denoted as $\underline{Gr [Af , DS]}$) corresponding to a particular DS (with respect to an arbitrary assignment function Af) is generated. With the consideration of 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 property of $Gr[Af, DS]$. There may be many such valid sets, each corresponding to a unique graph, and each resulting in a Communication-timep 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 Figure 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).

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 $V[i, j]$ ($0 \leq j < NI[i]$) and the output-data vertex $V_g[i]$ (as shown in Figure 4). \underline{V} is a set that contains all the above vertices associated with the application program P in Steps 1 and 2.

Step 3: Let \underline{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).

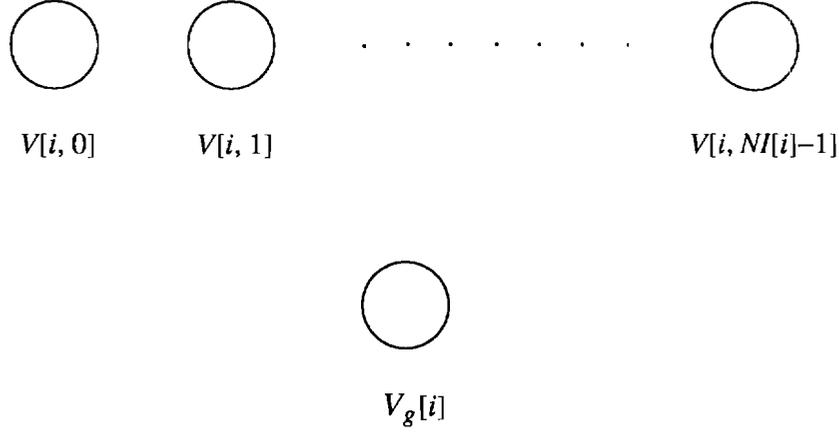


Figure 4: The generation of the vertices for the atomic operations of $S[i]$.

Step 4: For any input-data vertex $V[i_1, j_1]$, suppose that $DS[i_1](j_1) = i_2$, where $-1 \leq i_2 < n$.

Case A: If $0 \leq i_2 < n$, then for all j_2 such that $Id[i_1, j_1] = Id[i_2, j_2]$, a directed edge with weight $D[Af(i_2), Af(i_1)](|Id[i_1, j_1]|)$ is added from $V[i_2, j_2]$ to $V[i_1, j_1]$.

Case B: If $0 \leq i_2 < n$, then for all j_2 such that $Id[i_1, j_1] = Gd[i_2, j_2]$, a directed edge with weight $D[Af(i_2), Af(i_1)](|Id[i_1, j_1]|)$ is added from $V_g[i_2]$ to $V[i_1, j_1]$.

Case C: If $i_2 = -1$, then there exists k ($0 \leq k < Q$), such that $Id[i_1, j_1] = (-1, d_k)$, and a directed edge with weight $H[k](Af(i_1))$ is added from the Source vertex to $V[i_1, j_1]$.

For any input-data vertex $V[i_1, j_1]$ ($0 \leq i_1 < n$ and $0 \leq j_1 < NI[i_1]$), one and only one case of A, B, or C can occur. That is, $S[i_1]$ can obtain its required input-data item $Id[i_1, j_1]$ either from copying $S[i_2]$'s input-data item (Case A), or from the subtask that generates $Id[i_1, j_1]$ (Case B), or from the Source vertex if $Id[i_1, j_1] = (-1, d_k)$ and d_k is one of the initial data elements (Case C). Thus, any vertex $V[i_1, j_1]$ has one and only one parent vertex. Also, the weight of the edge between $V[i_1, j_1]$ and 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 Figure 5 and the sizes of the data items are shown as follows (d_0 and d_1 are

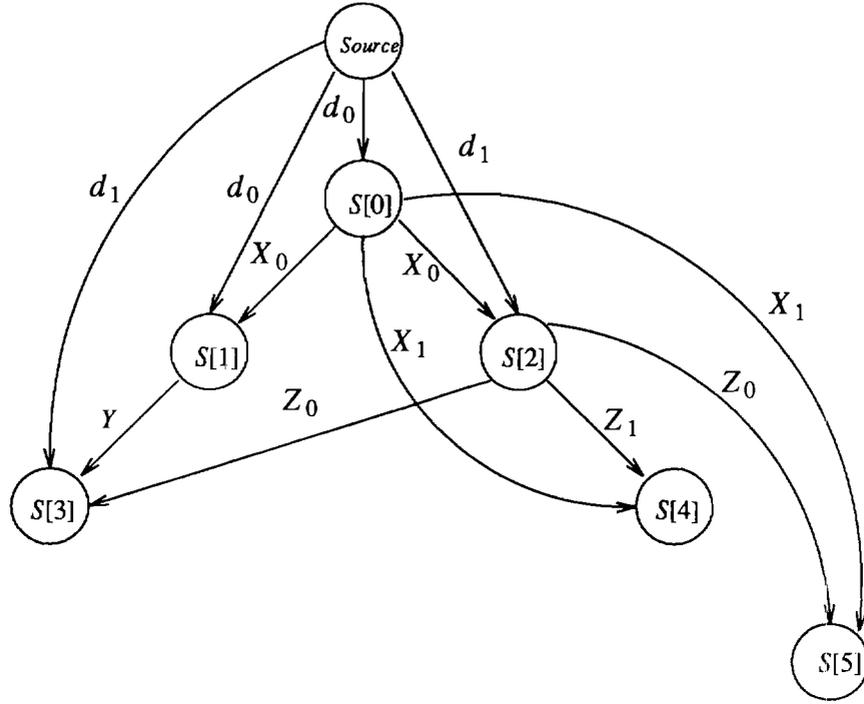


Figure 5: Subtask-flow graph for the example application program.

the variable names of initial data elements of \mathbf{P} ; $X_0, X_1, Y, Z_0,$ and Z_1 are the variable names of generated data items of \mathbf{P} ; a is an arbitrary constant).

$S[0]$: $NI[0] = 1, Id[0, 0] = (-1, d_0), |d_0| = 2a$;

$NG[0] = 2, Gd[0, 0] = (0, X_0), Gd[0, 1] = (0, X_1), |X_0| = 8a, |X_1| = 3a$.

$S[1]$: $NI[1] = 2, Id[1, 0] = (-1, d_0), Id[1, 1] = (0, X_0)$;

$NG[1] = 1, Gd[1, 0] = (1, Y), |Y| = 5a$.

$S[2]$: $NI[2] = 2, Id[2, 0] = (0, X_0), Id[2, 1] = (-1, d_1), |d_1| = 6a$;

$NG[2] = 2, Gd[2, 0] = (2, Z_0), Gd[2, 1] = (2, Z_1), |Z_0| = 4a, |Z_1| = a$.

$S[3]$: $NI[3] = 3, Id[3, 0] = (-1, d_1), Id[3, 1] = (1, Y), Id[3, 2] = (2, Z_0)$; and $NG[3] = 0$.

$S[4]$: $NI[4] = 2$, $Id[4, 0] = (0, X_1)$, $Id[4, 1] = (2, Z_1)$; and $NG[4] = 0$.

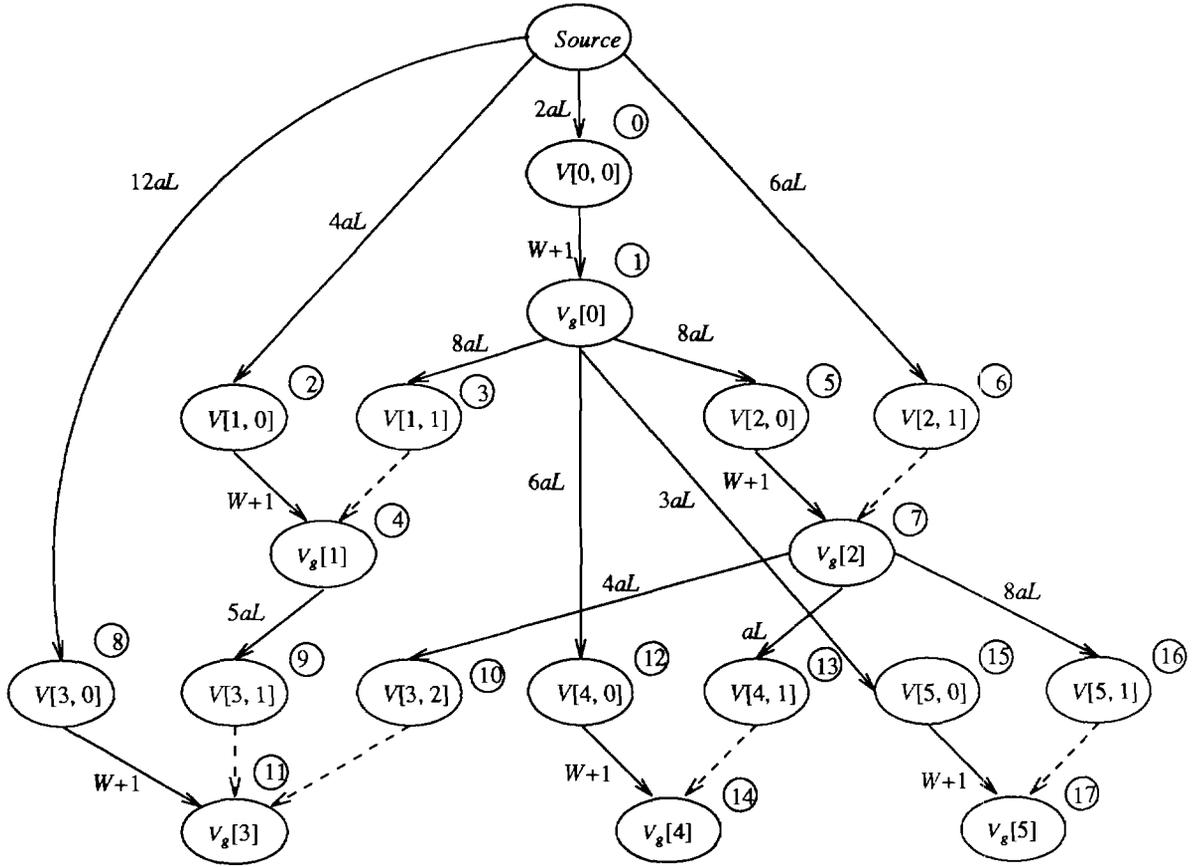
$S[5]$: $NI[5] = 2$, $Id[5, 0] = (0, X_1)$, $Id[5, 1] = (2, Z_0)$; and $NG[5] = 0$.

Based on the same linear array of machines as shown in Figure 2, the result of applying the set of data-source functions defined by the subtask-flow graph in Figure 5 is illustrated in Figure 6(a) and given in Figure 6(b). The solid lines in Figure 6(a), except the lines with weight $W + 1$, show the direct edges added by applying Step 4. The assignment function Af for this current example is shown in Figure 6(c): $Af(0) = 1$, $Af(1) = 2$, $Af(2) = 2$, $Af(3) = 1$, $Af(4) = 3$, and $Af(5) = 0$. W is $24aL$ according to Step 3.

Step 5: For every $0 \leq i < n$, a directed edge with weight $W + 1$ (i.e., a weight greater than any possible communication time) is added from $V[i, 0]$ to $V_g[i]$. $V_g[i]$ also has one and only one parent vertex, i.e., $V[i, 0]$. These directed edges are shown by the solid lines with weight $W + 1$ in Figure 6(a).

If $Gr[Af, DS]$ generated above 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. The DS defined in Figure 6(b) is a valid set of data-source functions.

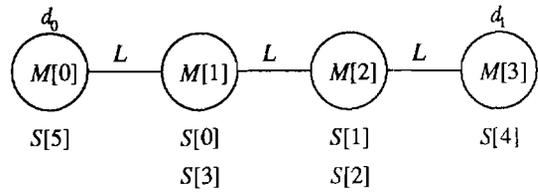
The reason for this definition is that, for a valid set of data-source functions DS, $Gr[Af, DS]$ must be an acyclic graph. Otherwise a deadlock arises in the application program P, which makes P unschedulable (recall the earlier example of an invalid DS). Because a $Gr[Af, DS]$ generated with respect to a valid DS is acyclic and each vertex (except the Source vertex) of $Gr[Af, DS]$ has one and only one parent vertex, from basic graph theory [BoM76], $Gr[Af, DS]$ is a tree with 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 tree or not. Furthermore, with an arbitrary assignment function Af and a valid set of data-source functions DS, the weight of the edge between $V[i_1, j_1]$ ($0 \leq i_1 < n$ and $0 \leq j_1 < NI[i_1]$) and its



(a)

$DS0 = -1;$
 $DS[1](0) = -1, \quad DS1 = 0;$
 $DS[2](0) = 0, \quad DS[2](1) = -1;$
 $DS[3](0) = -1, \quad DS[3](1) = 1, \quad DS[3](2) = 2;$
 $DS[4](0) = 0, \quad DS[4](1) = 2;$
 $DS[5](0) = 0, \quad DS[5](1) = 2.$

(b)



(c)

Figure 6: Generating a spanning tree with respect to the set of data-source 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.



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

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

where $\text{Weight}(x)$ is the sum of the weights on all edges of tree x . For the application program P specified by Figure 5, with respect to the given assignment function Af and the given valid data-source functions DS as defined in Figure 6(b), $\text{Communication_time}_P(Af, DS) = 67aL$.

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 $\text{Tree}[Af, DS]$ for every i_1 and j_1 except $j_1 = 0$ (i.e., $0 \leq i_1 < n$ and $1 \leq j_1 < NI[i_1]$). These directed edges are illustrated by the dashed lines shown in Figure 6(a) for the example application program P . 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 [CoL92] to this generated DAG. For the example application program P , the numbers in the circles in Figure: 6(a) indicate one ordering for the execution of the corresponding atomic input operations and subtask computation of P as determined by one particular topological sort.

It is stated in part (10) of the mathematical model presented in Section 2 that $\text{Communication_time}_P$ is a function of Af , Sf , and DS. If TIE is allowed, because Order is an extended version of Sf , $\text{Communication_time}_P$ is a function of 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 the respective valid DS. If not, the scheduling scheme and the data relocation scheme are incompatible with each other (i.e., Order and DS collectively cannot result in an operational program). Suppose Order_1 and Order_2 are two sets of ordering functions, because $\text{Communication_time}_P(Af, DS) = \text{Weight}(\text{Tree}[Af, DS]) - n(W + 1)$, $\text{Communication_time}_P(Af, \text{Order}_1, DS) = \text{Communication_time}_P(Af, \text{Order}_2, DS)$. 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 , $\text{Communication_time}_P$ is a function of Af and DS only. 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 is to find an assignment function Af^* and a valid set of data-source functions DS^* , such that

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

5. A Minimum Spanning Tree Based Algorithm for Finding the Optimal Set of Data-Source Functions and the Corresponding Set of Ordering Functions

5.1 Description of the Algorithm

For an arbitrary assignment function Af , a minimum spanning tree based algorithm is presented for finding a corresponding optimal valid set of data-source functions (denoted by DS^*), such that for any other valid set of data-source functions DS ,

$$\text{Execution-time, } (Af, DS^*) \leq \text{Execution-time, } (Af, DS).$$

A directed graph Dg (see Figure 7(a)) 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 Section 4). Figure 7 is based on the example program shown in Figure 5, uses the same machine and assignment function as in Figure 6(c), and has all the same vertices as in Figure 6(a).

- (a) For every i_1, j_1, i_2 , and j_2 , where $0 \leq i_1, i_2 < n$, $0 \leq j_1 < NI[i_1]$, $0 \leq j_2 < NI[i_2]$, and $i_1 \neq i_2$, 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.
- (b) For every i_1, j_1, i_2 , and j_2 , where $0 \leq i_1, i_2 < n$, $0 \leq j_1 < NG[i_1]$, and $0 \leq j_2 < NI[i_2]$, 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.
- (c) For every i, j , and k , such that $Id[i, j] = (-1, d_k)$, where $0 \leq i < n$, $0 \leq j < NI[i]$, and $0 \leq k < Q$, a directed edge from Source vertex to $V[i, j]$ with weight $H[k](Af(i))$ is added.

All the edges generated in (a), (b), and (c) are called fetch edges. For the example application program P illustrated by the subtask-flow graph in Figure 5, with the linear network of four machines as the heterogeneous suite and the assignment function defined in Figure 7(c) (and Figure 6(c)), the edges (both solid lines and dashed lines) of Dg in Figure 7(a) (except the ones with weight $W + 1$) are fetch edges.

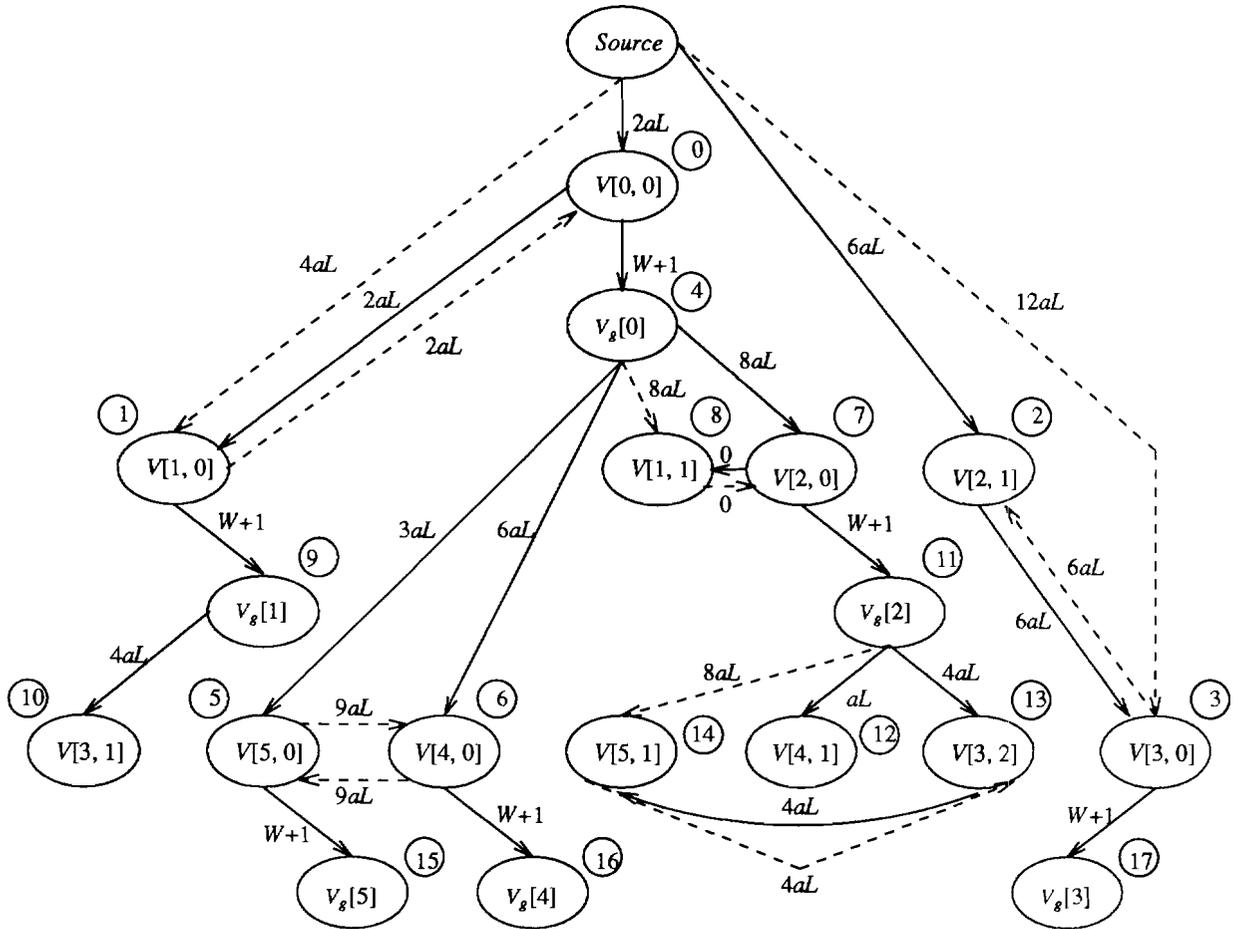
(d) For every $0 \leq i < n$, a directed edge from $V[i, 0]$ to $V_g[i]$ with weight $W + 1$ is added.

All these edges generated in (d) 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 Figure 7(a) 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 [CoL92], referred to as the TIE algorithm in this paper, for finding a minimum spanning tree (denoted as $MST[Af]$) of Dg is applied. 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 tree partially generated during the execution of this TIE algorithm. The order of execution for the atomic input operation that corresponds to any vertex $V[i, j]$ ($0 \leq i < n$ and $0 \leq j < NI[i]$) in V is $Order^*[i](j)$. The TIE algorithm is described as follows.

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

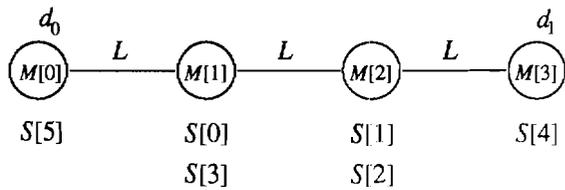
Step 2: Case A: If the set of cut edge(s) between A and $V - A$ (a cut edge is an edge that connects a vertex in A and a vertex in $V - A$) contains fetch edge(s) then find a cut edge that has the smallest weight (there might be several, in this case just choose an arbitrary minimum weight edge). Include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



- $DS^*0 = -1;$
- $DS^*[1](0) = 0, \quad DS^*1 = 2;$
- $DS^*[2](0) = 0, \quad DS^*[2](1) = -1;$
- $DS^*[3](0) = 2, \quad DS^*[3](1) = 1, \quad DS^*[3](2) = 2;$
- $DS^*[4](0) = 0, \quad DS^*[4](1) = 2;$
- $DS^*[5](0) = 0, \quad DS^*[5](1) = 3.$

(b)

(a)



(c)

Figure 7: Generating a minimum spanning tree for the example application program and its corresponding valid data-source functions. (a) The minimum spanning tree (solid lines). (b) The set of data-source functions. (c) The linear array network and the matching scheme.

is greater than the weight of any fetch edge.

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 $\{ V_g[i_0], V_g[i_1], \dots, V_g[i_j], \dots, V_g[i_{u-1}] \}$, where $1 \leq u \leq n$, $0 \leq j < u$, $0 \leq i_j < n$, and u is the number of activate edges in that set. It can be shown that there always exists at least one j ($0 \leq j < u$) such that all $V[i_j, k]$ is contained in A already ($0 \leq k < NI[i_j]$) 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 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 contain a deadlock. Include a ready-to-execute vertex (if there are several, choose any one of the ready-to-execute vertices) $V_g[i_j]$ in A and T , and its corresponding activate edge (i.e., the edge from $V[i_j, 0]$ to $V_g[i_j]$) in T . Because all $V_g[i]$ ($0 \leq i < n$) are included in the $MST[A_f]$ 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. The above procedure that uses two classes of edges and places a ready-to-execute vertex into T and A is the only difference between Prim's algorithm and the TIE algorithm. 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 Figure 5, with the linear network of four machines as the heterogeneous suite and the same assignment functions defined in Figure 7(c), the solid lines in Figure 7(a) show the $MST[A_f]$ corresponding to A_f after applying the TIE algorithm to D_g . This $MST[A_f]$ was generated by knowing only A_f , $I[i]$, 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:

- (a) If, in $MST[Af]$, the parent vertex of $V[i_1, j_1]$ is $V[i_2, j_2]$, then $DS^*[i_1](j_1) = i_2$.
- (b) If, in $MST[Af]$, the parent vertex of $V[i_1, j_1]$ is $V_g[i_2]$, then $DS^*[i_1](j_1) = i_2$.
- (c) 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 one and only one parent vertex, and the value of $DS^*[i_1](j_1)$ for any $0 \leq i_1 < n$ and $0 \leq j_1 < NI[i_1]$ is unique. The optimal set of data-source functions DS for the application program P illustrated by the subtask-flow graph in Figure 5 is derived and given in Figure 7(b) according to the procedures described above. The numbers in the circles in Figure 7(a) indicate the order in which vertices were added to the minimum spanning tree, and is the order for executing their corresponding atomic input operations and subtask computation during the execution of the application program P . The set of ordering functions, $Order^*[i](j)$, generated by the TIE algorithm corresponds to this order except it does not include the computation vertices (i.e., V_g 's).

For the complexity analysis of the TIE algorithm, suppose that $|E|$ is the number of edges in Dg and $|V|$ 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 [CoL92], the worst case asymptotic complexity of the algorithm for finding DS^* is $O(|E| + |V| \lg |V|)$. For Dg , $|V| = \sum_{i=0}^{n-1} (NI[i] + 1) + 1$

$= \sum_{i=0}^{n-1} NI[i] + n + 1$. 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 the source where the initial data elements are stored. Each vertex $V_g(i)$ is

connected to $V[i, 0]$ only. Thus, $|E| \leq n \sum_{i=0}^{n-1} NI[i] + n$. If $A = \sum_{i=0}^{n-1} NI[i]$, then $|V| = A + n + 1$ and $|E| \leq$

$nA + n$. The worst case asymptotic complexity of the TIE algorithm in terms of A and n is $O[nA + (n + A)\lg(n + A)]$, where n is the number of subtasks in P .

5.2 Proof of Correctness of the Algorithm

It is shown in Section 4 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 $Weight(Tree_P[Af, DS])$) is $Communication_time_P(Af, DS) + n(W + 1)$.

Thus,

$$\begin{aligned} Execution_time_P(Af, DS) &= Computation_time_P(Af) + Communication_time_P(Af, DS) \\ &= Computation_time_P(Af) + Weight(Tree_P[Af, DS]) - n(W+1). \end{aligned}$$

Because

$$\begin{aligned} Execution_time_P(Af, DS^*) &= Computation_time_P(Af) + Weight(Tree_P[Af, DS^*]) - n(W+1) \\ &= Computation_time_P(Af) + Weight(MST[Af]) - n(W+1) \end{aligned}$$

and

$$Weight(MST[Af]) \leq Weight(Tree_P[Af, DS]),$$

it is true that

$$Execution_time_P(Af, DS^*) \leq Execution_time_P(Af, DS).$$

For the application program P illustrated by the subtask-flow graph in Figure 5, if the set of data-source functions DS is determined directly from the subtask-flow graph provided (as shown in Figure 6(b)), the $Execution_time_P$ 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 5.1 and using DS^* , the $Execution_time_P$ is $C[0, 1] + C[1, 2] + C[2, 2] + C[3, 1] + C[4, 3] + C[5, 0] + 47aL$.

6. Two-Stage Approach for Matching, Scheduling, and Data Relocation in HC

In Sections 4 and 5, it was shown how to calculate an $Order^*$ and a DS^* given a fixed Af (allowing both data-reuse and multiple data-copies). The algorithm presented in Section 5 can be used to do this in polynomial time. However, as was stated in Section 4, the objective of matching, scheduling, and data relocation (with the assumption that TIE is allowed) is to find Af^* and DS^* (with one of its corresponding $Order^*$) for a specific application program P , such that, for any assignment function Af and any valid set of data-source functions DS , $Execution_time_P(Af^*, DS^*) \leq Execution_time_P(Af, DS)$. The problem of Finding Af^* is, in general, NP-complete with an arbitrary heterogeneous suite of m machines and an arbitrary application program P with n subtasks [Fer89].

One approach to matching, scheduling, and data relocation in HC is to find a suboptimal assignment function Af and a suboptimal valid set of data-source functions DS using a heuristic algorithm. Another approach, called the two-stage approach for matching, scheduling, and data relocation in HC, is introduced as follows:

Stage 1: Any existing heuristic (e.g., [Lo88, WaA94, WaS94]) for finding a suboptimal assignment function, Af_{sub} , can be applied in the first stage.

Stage 2: Once a specific assignment function Af_{sub} is found, the TIE algorithm can be applied to find the optimal set of data-source functions DS^* and the corresponding set of ordering functions $Order^*$ with respect to Af_{sub} . The tuple $(Af_{sub}, Order^*, DS^*)$ is a suboptimal solution for matching, scheduling, and data relocation problems in HC.

One of the advantages of the two-stage approach is that efforts for deriving the heuristic can be concentrated solely on finding a "good" suboptimal assignment function Af_{sub} . After Stage 1, the separate provably optimal TIE algorithm for finding $Order^*$ and DS^* with respect to Af_{sub} can be applied.

7. Summary

In an HC system, the subtasks of an application program P must be assigned to a suite of heterogeneous machines to utilize the computational resources effectively (the matching problem). The execution time of P is impacted by the order of execution of subtasks (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 inter-machine communication time in an HC system can have a significant impact on overall system performance, so any techniques that can be used to reduce this time are important. This paper focuses on scheduling schemes and data relocation schemes to minimize inter-machine communication time for a given matching scheme.

In this paper, a mathematical model for matching, scheduling, and data relocation in HC was presented. The assignment function Af , the scheduling function Sf (including the set of ordering functions $Order$), and the set of data-source functions DS were used to quantify the matching, scheduling, and data relocation problems respectively. Two data-distribution situations were identified, namely data-reuse and the multiple data-copies. A theorem was presented, which states that if only data-reuse is considered (and not the multiple data-copies situation), then the execution time of P is independent of Sf and DS . Section 4 introduced an extension to scheduling, called temporally interleaved execution of the atomic input operations for different subtasks (TIE). Examples were provided to show that both multiple data-copies and TIE have impact on the execution time of the application program P . A minimum spanning tree based algorithm with polynomial complexity was described for finding an optimal set of ordering functions $Order^*$ and an optimal set of data-source functions DS^* for an arbitrary assignment function Af . Based on this algorithm, a two-stage approach for matching, scheduling, and data relocation in HC was proposed.

To limit the scope of this paper, the sequential execution of subtasks for a specific application program P was assumed. But data-reuse and multiple data-copies will also occur when con-

current execution of subtasks across different machines in the HC is allowed. This sequential work is a necessary step in solving the more general situation involving concurrency. Future research includes applying the concepts developed here to the more general problem.

Acknowledgments: The authors thank R. Gupta, M. Maheswaran, and J. M. Siegel for their valuable comments.

References

- [Bok81] S. H. Bokhari, "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system," IEEE Trans. on *Software Engineering*, Vol. SE-7, No. 6, Nov. 1981, pp. 583-589.
- [BoM76] J. A. Bondy and U. S. R. Murty, Graph theory with applications, Elsevier Science Publishing Co., Inc., New York, NY, 1976.
- [ChE93] S. Chen, M. M. Eshaghian, A. Khokhar, and M. E. Shaaban, "A selection theory and methodology for heterogeneous supercomputing," Workshop on *Heterogeneous Processing*, Apr. 1993, pp. 15-22.
- [CoL92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, Mass., 1992.
- [Fer89] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," IEEE Trans. on Software Engineering, Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.
- [Fre89] R. F. Freund, "Optimal selection theory for superconcurrency," *Supercomputing '89*, Nov. 1989, pp. 699-703.
- [FrS93] R. F. Freund and H. J. Siegel, "Guest editors' introduction: heterogeneous processing," IEEE Computer, Vol. 26, No. 6, June 1993, pp. 13-17.
- [GhY93] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," IEEE Computer, Vol. 26, No. 6, June 1993, pp. 78-86.
- [KhP92] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous supercomputing: problems and issues," Workshop on Heterogeneous Processing, Mar. 1992, pp. 3-12.
- [KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: challenges and opportunities," IEEE Computer, Vol. 26, No. 6, June 1993, pp.

18-27.

- [Lo88] V. M. Lo, "Heuristic algorithm for task assignment in distributed systems," *IEEE Trans. on Computers*, Vol. 37, No. 11, Nov. 1988, pp. 1384-1397.
- [NaY94] B. Narahari, A. Youssef, and H. A. Choi, "Matching and scheduling in a generalized optimal selection theory," *Heterogeneous Computing Workshop*, Apr. 1994, pp. 3-8.
- [SiA95] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Handbook of Parallel and Distributed Computing*, edited by A. Y. Zomaya, McGraw-Hill, 1995, to appear (also Technical Report EE-37, School of Electrical Engineering, Purdue University, Dec. 1994).
- [Sto77] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. on Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.
- [Tow86] D. Towsley, "Allocating programs containing branches and loops within a multiple processor system," *IEEE Trans. on Software Engineering*, Vol. SE-12, No. 10, Oct. 1986, pp. 1018-1024.
- [WaA94] D. W. Watson, J. K. Antonio, H. J. Siegel, and M. J. Atallah, "Static program decomposition among machines in an SIMD/SPMD heterogeneous environment with non-constant mode switching costs," *Heterogeneous Computing Workshop*, Apr. 1994, pp. 58-65.
- [WaK92] 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," *Workshop on Heterogeneous Processing*, Mar. 1992, pp. 13-22.
- [WaS94] D. W. Watson, H. J. Siegel, J. K. Antonio, M. A. Nichols, and M. J. Atallah, "A block-based mode selection model for SIMD/SPMD parallel environments," *J. Parallel and Distributed Computing*, Vol. 21, No. 3, June 1994, pp. 271-288.

- [YaK94] J. Yang, A. Khokhar, S. Sheikh, and A. Ghafoor, "Estimating execution time for parallel tasks in heterogeneous processing (HP) environment," *Heterogeneous Computing Workshop*, Apr. 1994, pp. 23-28.