JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 47, 8–22 (1997)
ARTICLE NO. PC971392

Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach[1]

Lee Wang,*,[2] Howard Jay Siegel,*,[2] Vwani P. Roychowdhury,†,[3] and Anthony A. Maciejewski*,[2]

*Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907-1285; and †Electrical Engineering Department, UCLA, Los Angeles, California 90095-1594

To exploit a heterogeneous computing (HC) environment, an application task may be decomposed into subtasks that have data dependencies. Subtask matching and scheduling consists of assigning subtasks to machines, ordering subtask execution for each machine, and ordering intermachine data transfers. The goal is to achieve the minimal completion time for the task. A heuristic approach based on a genetic algorithm is developed to do matching and scheduling in HC environments. It is assumed that the matcher/scheduler is in control of a dedicated HC suite of machines. The characteristics of this genetic-algorithm-based approach include: separation of the matching and the scheduling representations, independence of the chromosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. It is applicable to the static scheduling of production jobs and can be readily used to collectively schedule a set of tasks that are decomposed into subtasks. Some parameters and the selection scheme of the genetic algorithm were chosen experimentally to achieve the best performance. Extensive simulation tests were conducted. For small-sized problems (e.g., a small number of subtasks and a small number of machines), exhaustive searches were used to verify that this genetic-algorithm-based approach found the optimal solutions. Simulation results for larger-sized problems showed that this genetic-algorithm-based approach outperformed two nonevolutionary heuristics and a random search. © 1997 Academic Press

1. INTRODUCTION

Different portions of an application task often require different types of computation. In general, it is impossible for a single machine architecture with its associated compiler, operating system, and programming tools to satisfy all the computational requirements in such an application equally well. However, a *heterogeneous computing* (*HC*) environment that consists of a heterogeneous suite of machines, high-speed interconnections, interfaces, operating systems, communication protocols, and programming environments provides a variety of architectural capabilities, which can be orchestrated to perform an application that has diverse execution requirements [Fre89, FrS93, KhP93, SiA96, Sun92]. In the HC environment considered here, an application task can be decomposed into subtasks, where each subtask is computationally homogeneous (well suited to a single machine), and different subtasks may have different machine architectural requirements. These subtasks can have data dependences among them. Once the application task is decomposed into subtasks, the following decisions have to be made: *matching*, i.e., assigning subtasks to machines, and *scheduling*, i.e., ordering subtask execution for each machine and ordering intermachine data transfers. In this context, the goal of HC is to achieve the minimal *completion time*, i.e., the minimal overall execution time of the application task in the machine suite.

It is well known that such a matching and scheduling problem is in general NP-complete [Fer89]. A number of approaches to different aspects of this problem have been proposed (e.g., [EsS94, Fre89, IvO95, NaY94, TaA95, WaA94]). Different from the above approaches, this paper proposes a genetic-algorithm-based approach for solving the problem.

Genetic algorithms for subtask scheduling in a collection of homogeneous processors have been considered (e.g., [AhD96, BeS94, HoA94]). Performing matching and scheduling for a suite of heterogeneous machines, however, requires a very different genetic algorithm structure.

In [IvO95], a nonevolutionary heuristic based on level scheduling [ChL88, MuC69] is presented to find a suboptimal matching and concurrent scheduling decision. That approach is compared to the performance of the evolutionary genetic-algorithm-based approach proposed in this paper.

This paper proposes a genetic-algorithm-based approach for solving the matching and concurrent scheduling problem in HC systems. It decides the subtask to machine assignments, orders the execution of the subtasks assigned to each machine, and schedules the data transfers among subtasks. The characteristics of this approach include: separation of the matching and the scheduling representations, independence of the chro-

[1]This research was supported in part by NRaD under Subcontract 20-950001-70 and by the DARPA/ITO Quorum Program under NPS Subcontract N62271-97-M-0900.

[2]E-mail: {lwang,hj,maciejew}@ecn.purdue.edu.

[3]E-mail: vwani@ee.ucla.edu.

0743-7315/97 $25.00

mosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. The computation and communication overlap is limited only by intersubtask data dependencies and machine/network availability. This genetic-algorithm-based approach can be applied to performing the matching and scheduling in a variety of HC systems. It is applicable to the static scheduling of production jobs and can be readily used to collectively schedule a set of tasks that are decomposed into subtasks.

The organization of this paper is as follows. The matching and scheduling problem is defined in Section 2. Section 3 briefly describes genetic algorithms and gives the outline of the genetic-algorithm-based approach. In Section 4, the proposed representation of matching and scheduling decisions within the genetic framework is presented. Section 5 discusses how to generate the initial population of possible solutions used by the genetic algorithm. The selection mechanism is discussed in Section 6. Sections 7 and 8 define the crossover and mutation operators, respectively, used to construct new generations of populations. Section 9 gives the method for evaluating the quality of a solution and the experimental results are shown in Section 10. Some related work is viewed and compared with our approach in Section 11. Finally, Section 12 discusses some future research directions.

2. PROBLEM DEFINITION

There are many open research problems in the field of HC [SiA96]. To isolate and focus on the matching and scheduling problem, assumptions about other components of an overall HC system must be made. Assumptions such as those below are typically made by matching and scheduling researchers (e.g., [ShW96, SiY96]).

It is assumed that the application task is written in some machine-independent language (e.g., [WeW94]). It is also assumed that an application task is decomposed into multiple subtasks and the data dependencies among them are known and are represented by a directed acyclic graph. If intermachine data transfers are data dependent, then some set of expected data transfers must be assumed. The estimated expected execution time for each subtask on each machine is assumed to be known *a priori*. The assumption of the availability of expected subtask execution time for each type of machine is typically made for the current state-of-the-art in HC systems when studying the matching and scheduling problem (e.g., [Fre94, GhY93, ShW96, SiY96]). Finding the estimated expected execution times for subtasks is another research problem, which is outside the scope of this paper. Approaches for doing this estimation based on task profiling and analytical benchmarking are surveyed in [SiA96]. The HC system is assumed to have operating system support for executing each subtask on the machine it is assigned and for performing intermachine data transfers as scheduled by this genetic-algorithm-based approach.

In the type of HC system considered here, an application task is decomposed into a set of subtasks S. Define $|S|$ to be the number of subtasks in the set S and s_i to be the ith subtask. Then $S = \{s_i, 0 \le i < |S|\}$. An HC environment consists of a set of machines M. Define $|M|$ to be the number of machines in the set M and m_j to be the jth machine. Then $M = \{m_j, 0 \le j < |M|\}$. The estimated expected execution time of subtask s_i on machine m_j is T_{ij}, where $0 \le i < |S|$ and $0 \le j < |M|$. The *global data items* (*gdis*), i.e., data items that need to be transferred between subtasks, form a set G. Define $|G|$ to be the number of items in the set G and gdi_k to be the kth global data item. Then $G = \{\text{gdi}_k, 0 \le k < |G|\}$.

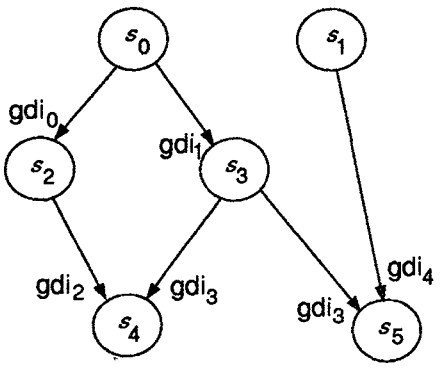


FIG. 1. An example DAG.

It is assumed that for each global data item, there is a single subtask that produces it (*producer*) and there are some subtasks that need this data item (*consumers*). The task is represented by a directed acyclic graph (*DAG*). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred. Figure 1 shows an example DAG.

The following further assumptions are made for the problem. One is the exclusive use of the HC environment for the application. The genetic-algorithm-based matcher/scheduler is in control of the HC machine suite. Another is nonpreemptive subtask execution. Also, all input data items of a subtask must be received before its execution can begin, and none of its output data items is available until the execution of this subtask is finished. If a data conditional is based on input data, it is assumed to be contained inside a subtask. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a subtask. These restrictions help make the matching and scheduling problem more manageable and solving this problem under these assumptions is a significant step forward for solving the general matching and scheduling problem.

3. GENETIC ALGORITHMS

Genetic algorithms (*GAs*) are a promising heuristic approach to finding near-optimal solutions in large search spaces [Dav91, Gol89, Hol75]. There are a great variety of approaches to GAs; many are surveyed in [SrP94, RiT94]. The following is a brief overview of GAs to provide background for the description of the proposed approach.

The first step necessary to employ a GA is to encode any possible solution to the optimization problem as a set of strings

(*chromosome*). Each chromosome represents one solution to the problem, and a set of chromosomes is referred to as a *population*. The next step is to derive an initial population. A random set of chromosomes is often used as the initial population. Some specified chromosomes can also be included. This initial population is the first generation from which the evolution starts.

The third step is to *evaluate* the quality of each chromosome. Each chromosome is associated with a *fitness value*, which is in this case the completion time of the solution (matching and scheduling) represented by this chromosome (i.e., the expected execution time of the application task if the matching and scheduling specified by this chromosome were used). Thus, in this research a smaller fitness value represents a better solution. The objective of the GA search is to find a chromosome that has the optimal (smallest) fitness value. The selection process is the next step. In this step, each chromosome is eliminated or duplicated (one or more times) based on its relative quality. The population size is typically kept constant.

Selection is followed by the *crossover* step. With some probability, some pairs of chromosomes are selected from the current population and some of their corresponding components are exchanged to form two valid chromosomes, which may or may not already be in the current population. After crossover, each string in the population may be *mutated* with some probability. The mutation process transforms a chromosome into another valid one that may or may not already be in the current population. The new population is then evaluated. If the stopping criteria have not been met, the new population goes through another cycle (iteration) of selection, crossover, mutation, and evaluation. These cycles continue until one of the stopping criteria is met.

In summary, the following are the steps that are taken to implement a GA for a given optimization problem: (1) an encoding, (2) an initial population, (3) an evaluation using a particular fitness function, (4) a selection mechanism, (5) a crossover mechanism, (6) a mutation mechanism, and (7) a set of stopping criteria. These steps of a typical GA are shown in Fig. 2.

Details of the steps for the implementation of the GA-based heuristic for HC will be discussed in the following sections. For some parameters of this GA, such as population size, values were selected based on information in the literature. For other parameters, such as the probability of performing a mutation operation, experiments were conducted (Section 10).

```
GA_matching_scheduling(){
        initial population generation;
        evaluation;
        while(stopping criteria not met){
                selection;
                crossover;
                mutation;
                evaluation;
        }
        output the best solution found;
}
```

FIG. 2. The steps in a typical GA.

4. CHROMOSOME REPRESENTATION

In this GA-based approach, each chromosome consists of two parts: the matching string and the scheduling string. Let *mat* be the *matching string*, which is a vector of length $|S|$, such that $\text{mat}(i) = j$, where $0 \le i < |S|$ and $0 \le j < |M|$; i.e., subtask s_i is assigned to machine m_j.

The *scheduling string* is a topological sort [CoL92] of the DAG, i.e., a total ordering of the nodes (subtasks) in the DAG that obeys the precedence constraints. Define *ss* to be the scheduling string, which is a vector of length $|S|$, such that $\text{ss}(k) = i$, where $0 \le i, k < |S|$, and each s_i appears only once in the vector, i.e., subtask s_i is the kth subtask in the scheduling string. Because it is a topological sort, if $\text{ss}(k)$ is a consumer of a global data item produced by $\text{ss}(j)$, then $j < k$. The scheduling string gives an ordering of the subtasks that is used by the evaluation step.

Then in this GA-based approach, a chromosome is represented by a two-tuple ⟨mat, ss⟩. Thus, a chromosome represents the subtask-to-machine assignments (matching) and the execution ordering of the subtasks assigned to the same machine. The scheduling of the global data item transfers and the relative ordering of subtasks assigned to different machines are determined by the evaluation step. Figure 3 illustrates two different chromosomes for the DAG in Fig. 1, for $|S| = 6$, $|M| = 3$, and $|G| = 5$.

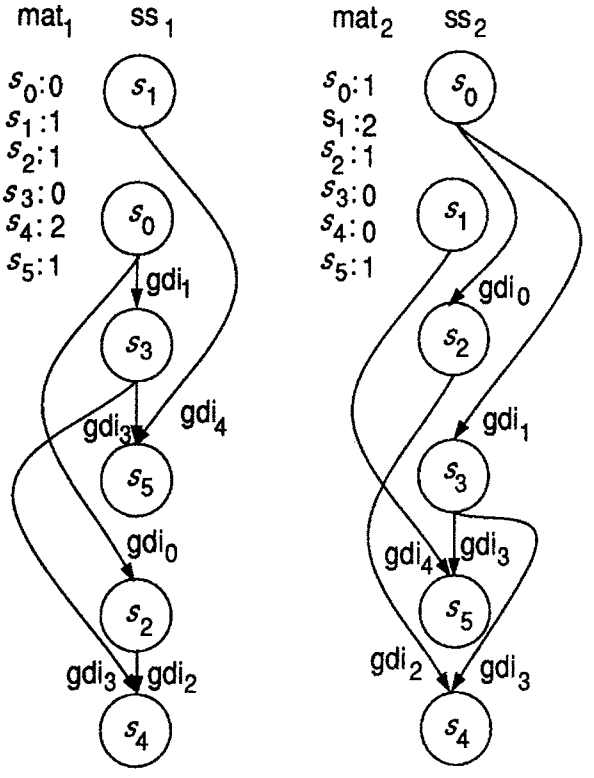


FIG. 3. Two chromosomes from the DAG in Fig. 1.

5. INITIAL POPULATION GENERATION

In the initial population generation step, a predefined number of chromosomes are generated, the collection of which form the initial population. When generating a chromosome, a new matching string is obtained by randomly assigning each subtask to a machine. To form a scheduling string, the DAG is first topologically sorted to form a basis scheduling string. Then, for each chromosome in the initial population, this basis string is mutated a random number of times (between one and the number of subtasks) using the scheduling string mutation operator (defined in Section 8) to generate the ss vector (which is a valid topological sort of the given DAG). Furthermore, it is common in GA applications to incorporate solutions from some nonevolutionary heuristics into the initial population, which may reduce the time needed for finding a satisfactory solution [Dav91]. In this GA-based approach, along with those chromosomes representing randomly generated solutions, the initial population also includes a chromosome that represents the solution from a nonevolutionary baseline heuristic. Details of this heuristic will be discussed in Section 10.

Each newly generated chromosome is checked against those previously generated. If a new chromosome is identical to any of the existing ones, it is discarded and the process of chromosome generation is repeated until a unique new chromosome is obtained. The reason why identical chromosomes are not allowed in the initial generation is that they could possibly drive the whole population to a premature *convergence*, i.e., the state where all chromosomes in a population have the same fitness value. It can be shown that for this GA-based approach, there is a nonzero probability that a chromosome can be generated to represent any possible solution to the matching and scheduling problem using the crossover and the mutation operators. The crossover and the mutation operators will be discussed later in Sections 7 and 8, respectively.

6. SELECTION

In this step, the chromosomes in the population are first ordered (ranked) by their fitness values from the best to the worst. Those having the same fitness value are ranked arbitrarily among themselves. Then a *rank-based roulette wheel selection scheme* can be used to implement the selection step [Hol75, SrP94]. In the rank-based selection scheme, each chromosome is allocated a sector on a roulette wheel. Let P denote the population size and A_i denote the angle of the sector allocated to the ith ranked chromosome. The 0th ranked chromosome is the fittest and has the sector with the largest angle A_0; whereas the $(P-1)$th ranked chromosome is the least fit and has the sector with the smallest angle A_{P-1}. The ratio of the sector angles between two adjacently ranked chromosomes is a constant $R = A_i/A_{i+1}$, where $0 \le i < P-1$. If the 360 degrees of the wheel are normalized to one, then

$$A_i = R^{P-i-1} \times (1-R)/(1-R^P),$$

where $R > 1$, $0 \le i < P$, and $0 < A_i < 1$.

The selection step generates P random numbers, ranging from zero to one. Each number falls in a sector on the roulette wheel and a copy of the owner chromosome of this sector is included in the next generation. Because a better solution has a larger sector angle than that of a worse solution, there is a higher probability that (one or more) copies of this better solution will be included in the next generation. In this way, the population for the next generation is determined. Thus, the population size is always P, and it is possible to have multiple copies of the same chromosome.

Alternatively, a *value-based roulette wheel selection scheme* can be used to implement a proportionate selection [SrP94]. Let f_i be the fitness value of the ith chromosome and f_{ave} be the average fitness value of the current population. In this selection scheme, the ith chromosome ($0 \le i < P$) is allocated a sector on the roulette wheel, the angle of which, A_i, is proportional to f_{ave}/f_i (assuming that the best chromosome has the smallest fitness value, which is the case for this research). The most appropriate selection scheme for this research was chosen experimentally. Details on the experiments can be found in Section 10 and [Wan97].

This GA-based approach also incorporates *elitism* [Rud94]. At the end of each iteration, the best chromosome is always compared with an elite chromosome, a copy of which is stored separately from the population. If the best chromosome is better than the elite chromosome, a copy of it becomes the elite chromosome. If the best chromosome is not as good as the elite chromosome, a copy of the elite chromosome replaces the worst chromosome in the population. Elitism is important because it guarantees that the quality of the best solutions found over generations is monotonically increasing.

7. CROSSOVER OPERATORS

Different crossover operators are developed for scheduling strings and matching strings. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings. For each pair, it randomly generates a cut-off point, which divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative positions of these subtasks in the other original scheduling string in the pair, thus guaranteeing that the newly generated scheduling strings are valid schedules. Figure 4 demonstrates such a scheduling string crossover process.

The crossover operator for the matching strings randomly chooses some pairs of the matching strings. For each pair, it randomly generates a cut-off point to divide both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The probability for performing crossovers was determined by experimentation. This is discussed in Section 10.

8. MUTATION OPERATORS

Different mutation operators are developed for scheduling strings and matching strings. The scheduling string mutation

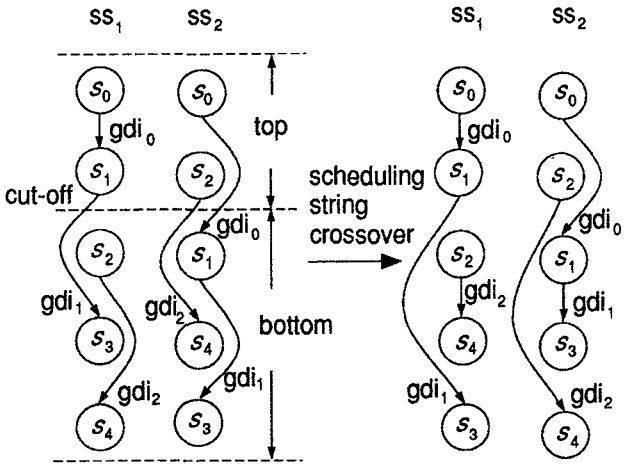


FIG. 4. A scheduling string crossover example.

operator randomly chooses some scheduling strings. Then for each chosen scheduling string, it randomly selects a victim subtask. The *valid range* of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed without violating any data dependency constraints. Specifically, the valid range is after all source subtasks of the victim subtask and before any destination subtask of the victim subtask. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its valid range. Figure 5 shows an example of this mutation process.

The matching string mutation operator randomly chooses some matching strings. On each chosen matching string, it randomly selects a subtask/machine pair. Then the machine assignment for the selected pair is changed randomly to another machine.

The probability for performing mutations was determined by experimentation. This is discussed in Section 10.

9. EVALUATION

The final step of a GA iteration is the evaluation of the fitness value of each chromosome. In this GA-based approach, the chromosome structure is independent of any particular communication subsystem. Only the evaluation step needs the communication characteristics of the given HC system to schedule the data transfers. To test the effectiveness of this GA-based approach, an example communication system was chosen. This GA-based approach can be used with any communication system that obeys the assumptions in Section 2.

To demonstrate the evaluation process, an example communication subsystem, which is modeled after a HiPPI LAN with a central crossbar switch [HoT89, ToR93], is assumed to connect a suite of machines. Each machine in the HC suite has one input data link and one output data link. All these links are connected to a central crossbar switch. Figure 6 shows an HC system consisting of four machines that are interconnected by such a crossbar switch. If a subtask needs a global data item that is produced or consumed earlier by a different subtask on the same machine, the communication time for this item is zero. Otherwise, the communication time is obtained by dividing the size of the global data item by the smaller bandwidth of the output link of the source machine and the input link of the destination machine. In this research, it is assumed that for a given machine, the bandwidths of the input link and the output link are equal to each other. It is also assumed that the crossbar switch has a higher bandwidth than that of each link. The communication latency between any pair of machines is assumed to be the same. Data transfers are neither preemptive nor multiplexed. Once a data transfer path is established, it cannot be relinquished until the data item (e.g., gdi_k) scheduled to be transferred over this path is received by the destination machine. Multiple data transfers over the same path have to be serialized.

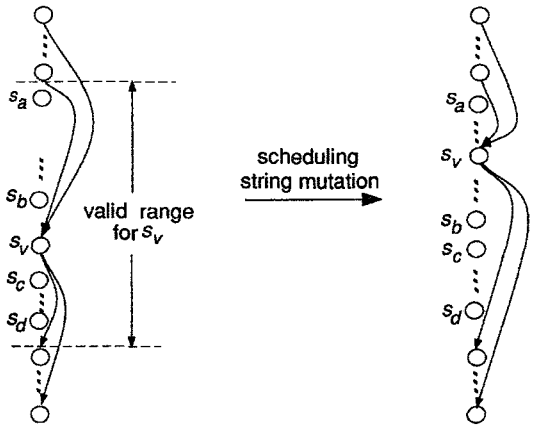


FIG. 5. A scheduling string mutation example. Only edges to and from the victim subtask s_v are shown. Before the mutation, s_v is between s_b and s_c. After the mutation, it is moved to between s_a and s_b.

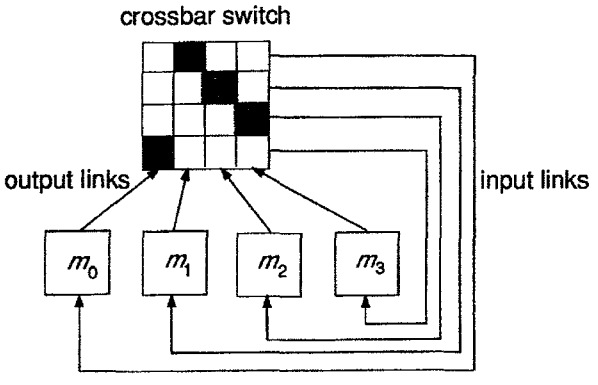


FIG. 6. An example HC system with four machines and a central crossbar switched network. Each machine has one output data link to and one input data link from the crossbar switch. Blackened squares in the switch correspond to active connections.

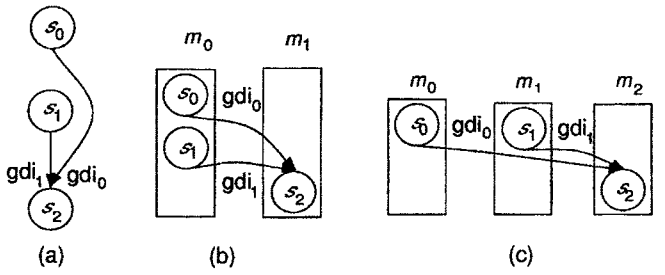


FIG. 7. An example showing the scheduling order for the input gdis of one subtask: (a) the example scheduling string; (b) the situation when the source subtasks of the input gdis are assigned to the same machine; (c) the situation when the source subtasks of the input gdis are assigned to different machines.

In this step, for each chromosome the final order of execution of the subtasks and the intermachine data transfers are determined. The evaluation procedure considers the subtasks in the order they appear on the scheduling string. Subtasks assigned to the same machine are executed exactly in the order specified by the scheduling string. For subtasks assigned to different machines, the actual execution order may deviate from that specified by the scheduling string due to factors such as input-data availability and machine availability. This is explained below.

Before a subtask can be scheduled, all of its input global data items must be received. For each subtask, its input data items are considered by the evaluation procedure in the order of their producers' relative positions in the scheduling string. The reason for this ordering is to better utilize the overlap of subtask executions and intermachine data communications. The following example illustrates this idea. Let ss(0) = 0, ss(1) = 1, and ss(2) = 2, as shown in Fig. 7a. Let s_2 need two gdis, gdi_0 and gdi_1, from s_0 and s_1, respectively. Depending on the subtask to machine assignments, the data transfers of gdi_0 and gdi_1 could be either local within a machine or across machines. If at least one data transfer is local, then the scheduling is trivial because it is assumed that local transfers within a machine take negligible time. However, there exist two situations where both data transfers are across machines so that they need to be ordered.

Situation 1. Let s_0 and s_1 be assigned to the same machine m_0 and s_2 be assigned to another machine m_1, as shown in Fig. 7b. In this situation, because s_0 is to be executed before s_1, gdi_0 is available before gdi_1 becomes available on machine m_0. Thus, it is better to schedule the gdi_0 transfer before the gdi_1 transfer.

Situation 2. Let the three subtasks s_0, s_1, and s_2 be assigned to three different machines m_0, m_1, and m_2, as shown in Fig. 7c. In this situation, if there is a data dependency from s_0 to s_1, then s_0 finishes its execution before s_1 could start its execution. Therefore, gdi_0 is available before gdi_1 becomes available. Hence, it is better to schedule the gdi_0 transfer before the gdi_1 transfer. If there are no data dependencies from s_0 to s_1, the gdi_0 transfer will still be scheduled before the gdi_1 transfer. While this may not be the best scheduling order for these gdis, the reverse order may be considered by other scheduling strings, i.e., there may be some other chromosome(s) that have ss(0) = 1 and ss(1) = 0. When such a chromosome is evaluated, the gdi_1 transfer will be scheduled before the gdi_0 transfer. Therefore, it is possible for all input gdi scheduling orderings for gdi_0 and gdi_1 to be examined.

In Fig. 8, a simple example is shown to illustrate the evaluation for a given chromosome. In this example (as well as some others given later), because there are only two machines, the source and destination machines for the gdi transfers are implicit. The ordering for the evaluation of subtasks and gdi transfers is: s_0, gdi_0, s_2, gdi_1, s_1, gdi_2, gdi_3, s_3. If a gdi consumer subtask is on the same machine as the producer or as a previous consumer of that gdi, no data transfer is required, as is the case for gdi_1 and gdi_3 in this example.

Data forwarding is another important feature of this evaluation process. For each input data item to be considered, the evaluation process chooses the source subtask from among the producer of this data item and all the consumers that have received this data item. These consumers are *forwarders*. The

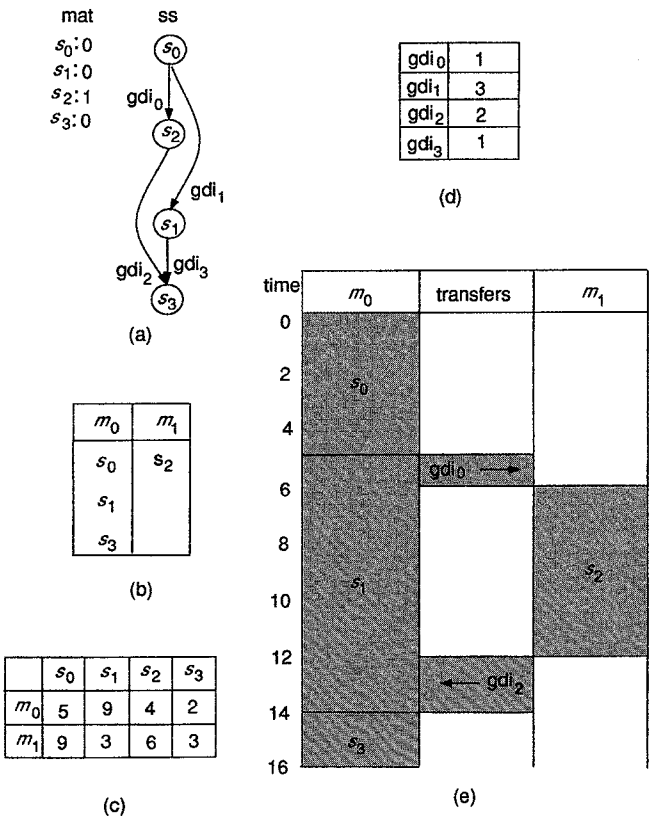


FIG. 8. An example showing the evaluation step: (a) the chromosome; (b) the subtask execution ordering on each machine given by (a); (c) the estimated subtask execution times; (d) the gdi intermachine transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution and data transfer timings, where the completion time for this chromosome is 16.

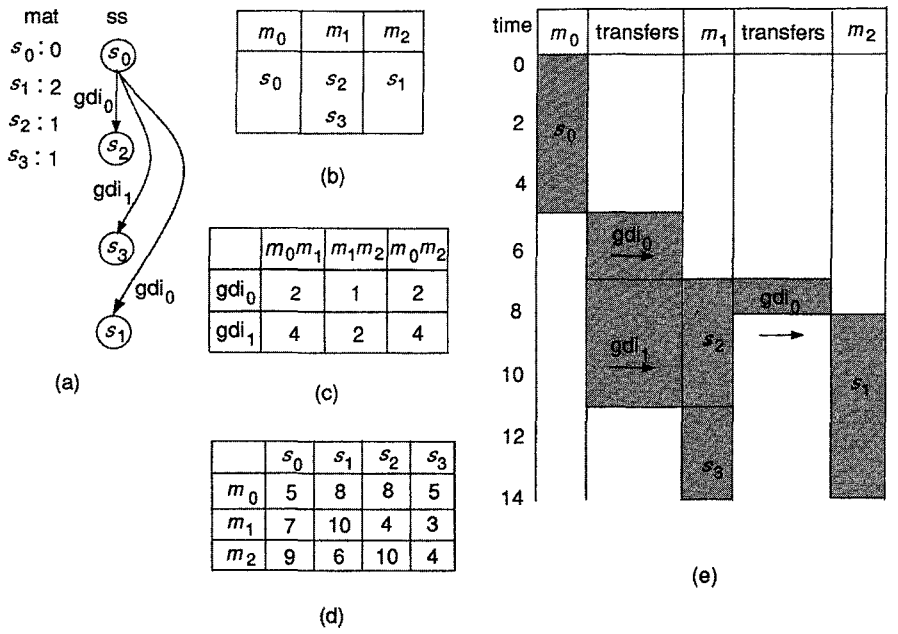


FIG. 9. A data forwarding example: (a) the chromosome; (b) the subtask execution ordering on each machine; (c) the gdi transfer times; (d) the estimated subtask execution times; and (e) the subtask execution and data transfer times using data forwarding.

one (either the producer or a forwarder) from which the destination subtask will receive the data item at the earliest possible time is chosen. Figure 9 shows an example of data forwarding. In this example, global data item gdi_0 is forwarded to subtask s_1 from a consumer subtask s_2 instead of from the producer subtask s_0. The resulting completion time is 14. If data forwarding is disabled for this example (i.e., global data item gdi_0 must be sent from subtask s_0 to subtask s_1), the completion time would be 16 (when subtask s_0 sends gdi_0 to subtask s_1 before sending gdi_1 to subtask s_3) or 19 (when subtask s_0 sends gdi_1 to subtask s_3 before sending gdi_0 to subtask s_1).

After the source subtask is chosen, the data transfer for the input data item is scheduled. A transfer starts at the earliest point in time from when the path from the source machine to the destination machine is free for a period at least equal to the needed transfer time. This (possibly) *out-of-order* scheduling of the input item data transfers utilizes previously idle bandwidths of the communication links and thus could make some input data items available to some subtasks earlier than otherwise from the in-order scheduling. As a result, some subtasks could start their execution earlier, which would in turn decrease the overall task completion time. This is referred to as out-of-order scheduling of data transfers because the data transfers do not occur in the order in which they are considered (i.e., the *in-order* schedule). Figures 10 and 11 show the in-order scheduling and the out-of-order scheduling for the same chromosome, respectively. In the in-order scheduling, the transfer of gdi_1 is scheduled before the transfer of gdi_2 because subtask s_2's input data transfers are considered before those of subtask s_3. In this example, the out-of-order schedule does decrease the total execution time of the given task.

When two chromosomes have different matching strings, they are different solutions because the subtask-to-machine assignments are different. However, two chromosomes that have the same matching string but different scheduling strings may or may not represent the same solution. This is because the scheduling string information is used in two cases: (1) for scheduling subtasks that have been assigned to the same machine and (2) for scheduling data transfers. Two different scheduling strings could result in the same ordering for (1) and (2).

After a chromosome is evaluated, it is associated with a fitness value, which is the time when the last subtask finishes its execution. That is, the fitness value of a chromosome is then the overall execution time of the task, given the matching and scheduling decision specified by this chromosome and by the evaluation process.

In summary, this evaluation mechanism considers subtasks in the order in which they appear in the scheduling string. For a subtask that requires some gdis from other machines, the gdi transfer whose producer subtask appears earliest in the scheduling string is scheduled first. When scheduling a gdi transfer, both the producing and the forwarding subtasks are considered. The source subtask that lets this consumer subtask receive this gdi at the earliest possible time is chosen to send the gdi. The out-of-order scheduling of the gdi transfers over a path could further reduce the completion time of the application.

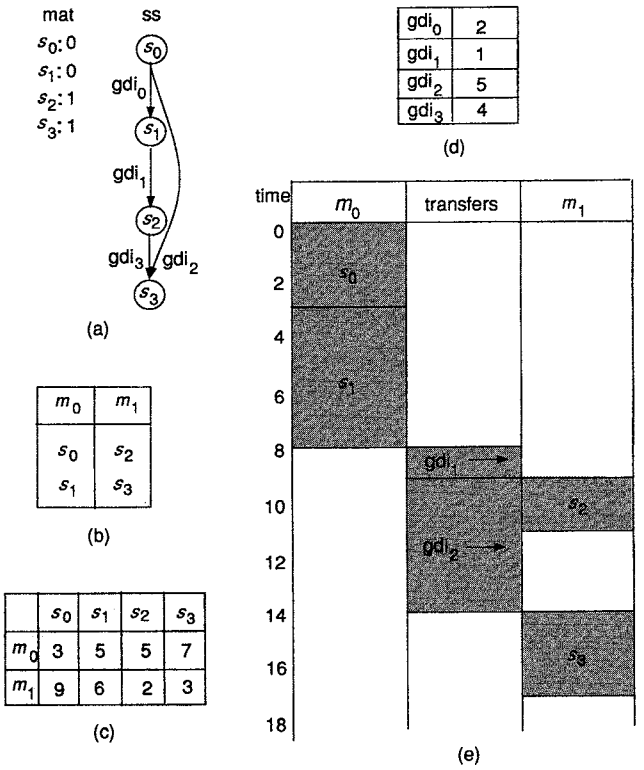


FIG. 10. An example showing the in-order scheduling of a chromosome: (a) the chromosome; (b) the subtask execution ordering on each machine; (c) the estimated subtask execution times; (d) the gdi transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution and data transfer timings using in-order transfers (the gdi_1 transfer occurs before the gdi_2 transfer), where the completion time is 17.

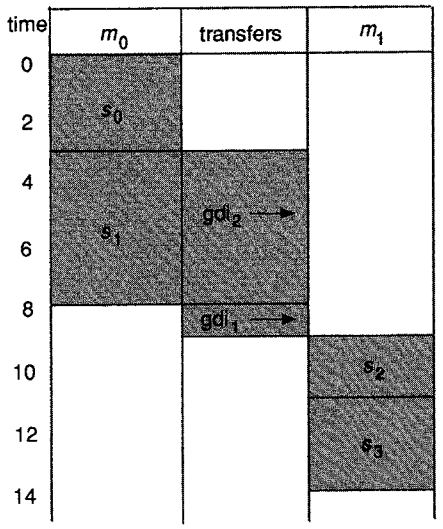


FIG. 11. An example showing the out-of-order scheduling, where the chromosome and other statistics are the same as in Fig. 10. The completion time is 14.

10. EXPERIMENTAL RESULTS

To measure the performance of this GA-based approach, randomly generated *scenarios* were used, where each scenario corresponded to a DAG, the associated subtask execution times, the sizes of the associated global data items, and the communication link bandwidths of the machines. The scenarios were generated for different numbers of subtasks and different numbers of machines, as specified below. The estimated expected execution time for each subtask on each machine, the number of global data items, the size of each global data item, and the bandwidth of each input link of each machine were randomly generated with uniform probability over some predefined ranges. For each machine, the bandwidth of the output link is made equal to that of the input link. The producer and consumers of each global data item were also generated randomly. The scenario generation used a $|G| \times |S|$ dependency matrix to guarantee that the precedence constraints from data dependencies were acyclic. Each row of this matrix specified the data dependencies of the corresponding global data item. In each row, the producer must appear to the left of all of its consumers.

These randomly generated scenarios were used for three reasons: (1) it is desirable to obtain data that demonstrate the effectiveness of the approach over a broad range of conditions, (2) a generally accepted set of HC benchmark tasks does not exist, and (3) it is not clear what characteristics a "typical" HC task would exhibit [WaA96]. Determining a representative set of HC task benchmarks remains a current and unresolved challenge for the scientific community in this research area.

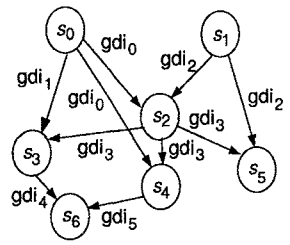
In this research, small-scale and larger scenarios were used to quantify the performance of this GA-based approach. The scenarios were grouped into three categories, namely tasks with light, moderate, and heavy communication loads. A lightly communicating task has its number of global data items in the range of $0 \leq |G| < (1/3)|S|$; a moderately communicating task has its number of global data items in the range of $(1/3)|S| \leq |G| < (2/3)|S|$; and a heavily communicating task has its number of global data items in the range of $(2/3)|S| \leq |G| < |S|$. The ranges of the global data item sizes and the estimated subtask execution times were both from 1 to 1000. For these scenarios, the bandwidths of the input and output links were randomly generated, ranging from 0.5 to 1.5. Hence, the communication times in these scenarios were source and destination machine dependent.

For each scenario, there were many *GA runs*, each of which was a GA search for the best solution to this scenario, starting from a different initial population. The probability of crossover was the same for the matching string and the scheduling string. The probability of mutation was also the same for the matching string and the scheduling string. The stopping criteria were (1) the number of iterations had reached 1000, (2) the population had converged (i.e., all the chromosomes had the same fitness value), or (3) the currently best solution had not improved over the last 150 iterations. All the GA runs discussed in this

section had stopped when the best solutions were not improved after 150 iterations.

The GA-based approach was first applied to 20 small-scale scenarios that involved up to ten subtasks, three machines, and seven global data items. The GA runs for small-scale scenarios had the following parameters. The probabilities for scheduling string crossovers, matching string crossovers, scheduling string mutations, and matching string mutations were chosen to be 0.4, 0.4, 0.1, 0.1, respectively. The GA population size, P, for small-scale scenarios was chosen to be 50. For these scenarios, the rank-based roulette wheel selection scheme was used. The angle ratio of the sectors on the roulette wheel for two adjacently ranked chromosomes, R, was chosen to be $1 + 1/P$. By using this simple formula, the angle ratio between the slots of the best and median chromosomes for $P = 50$ (and also for $P = 200$ for larger scenarios discussed later in this section) was very close to the optimal empirical ratio value of 1.5 in [Whi89].

The results from a small-scale scenario were used here to illustrate the search process. This scenario had $|S| = 7$, $|M| = 3$, and $|G| = 6$. The DAG, the estimated execution times, and the transfer times of the global data items are shown in Figs. 12a–12c, respectively. The total numbers of possible different matching strings and different valid scheduling strings (i.e., topological sorts of the DAG) were $3^7 = 2187$ and 16, respectively. Thus, the total search space had $2187 \times 16 = 34{,}992$ possible chromosomes.



(a)

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
m_0	872	251	542	40	742	970	457
m_1	898	624	786	737	247	749	451
m_2	708	778	23	258	535	776	15

(b)

	m_0m_1	m_0m_2	m_1m_2
gdi_0	489	321	489
gdi_1	1244	818	1244
gdi_2	62	41	62
gdi_3	830	545	830
gdi_4	387	255	387
gdi_5	999	656	999

(c)

FIG. 12. A small-scale simulation scenario: (a) the DAG, (b) the estimated execution times, and (c) the transfer times of the global data items.

Figure 13 depicts the evolution process of one GA run on this scenario. In each subfigure, the ss axis is the scheduling string axis and the mat axis is the matching string axis. The 16 different scheduling strings on the ss axis are numbered from 1 to 16. The 2187 different matchings on the mat axis are numbered from 1 to 2187. If there is a chromosome at a point (mat, ss), then there is a vertical pole at (mat, ss). The height of a pole represents the quality of the chromosome. The greater the height of the pole, the better a chromosome

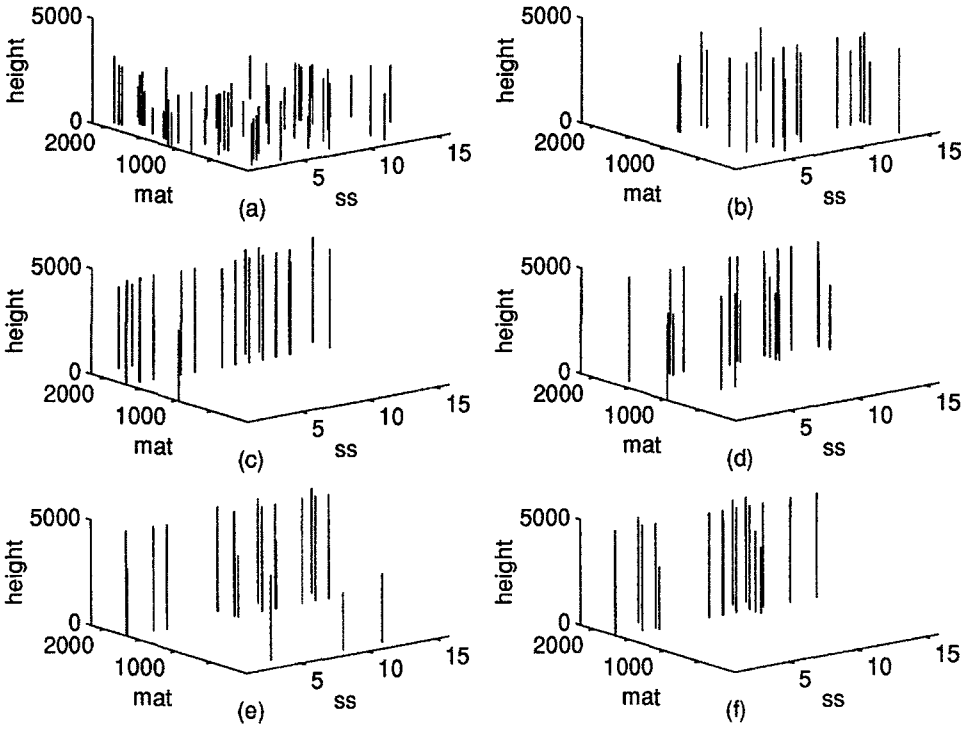


FIG. 13. Evolution of a GA run for the scenario in Fig. 12: (a) at iteration 0, (b) at iteration 40, (c) at iteration 80, (d) at iteration 120, (e) at iteration 160, and (f) at iteration 203 (when the search stopped). Height is a positive constant minus the task execution time associated with (*mat*, *ss*).

(solution) is. Multiple identical chromosomes at the same point are not differentiated. Figures 13a–13f show the distributions of the distinct chromosomes at iterations 0, 40, 80, 120, 160, and 203, respectively. This GA run stopped at iteration 203. This GA-based approach found multiple best solutions that have the same completion time, as shown in Fig. 13f.

Exhaustive searches were performed to find the optimal solutions for the small-scale scenarios. For each of the small-scale scenarios that were conducted, the GA-based approach found one or more optimal solutions that had the same completion time, verified by the best solution(s) found by the exhaustive search. The GA search for a small-scale scenario that had ten subtasks, three machines, and seven global data items took about 1 min to find multiple optimal solutions on a Sun Sparc5 workstation while the exhaustive search took about 8 h to find these optimal solutions.

The performance of this GA-based approach was also examined using larger scenarios with up to 100 subtasks and 20 machines. These larger scenarios were generated using the same procedure as for generating the small scenarios. The GA population size for larger scenarios was chosen to be 200.

Larger scenarios are intractable problems. It is currently impractical to directly compare the quality of the solutions found by the GA-based approach for these larger scenarios with those found by exhaustive searches. It is also difficult to compare the performance of different HC task matching and scheduling approaches due to the different HC system models used. Examples of such differences are given in the next section. However, the model used in [IvO95] is similar to the one being used in this research work. Hence, the performance of the GA-based approach on larger scenarios was compared with the nonevolutionary levelized min-time (*LMT*) heuristic proposed in [IvO95].

The LMT heuristic first levelizes the subtasks in the following way. The subtasks that have no input global data items are at the highest level. Each of the remaining subtasks is at one level below the lowest producer of its global data items. The subtasks at the highest level are to be considered first. The LMT heuristic averages the estimated execution times for each subtask across all machines. At each level, a level-average execution time, i.e., the average of the machine-average execution times of all subtasks at this level, is also computed. If there are some levels between a subtask and its closest child subtask, the level-average execution time of each middle level is subtracted from the machine-average execution time of this subtask. The adjusted machine-average execution times of the subtasks are used to determine the priorities of the subtasks within each level; i.e., a subtask with a larger average is to be considered earlier at its level. If the number of subtasks at a level is greater than the number of machines in the HC suite, the subtasks with smaller averages are merged so that as the result, the number of the combined subtasks at each level equals the number of machines available. When a subtask is being considered, it is assigned to the fastest machine available from those machines that have not yet been assigned any subtasks from the same level. Then, it is scheduled using the scheduling principles discussed in Section 9.

Another nonevolutionary heuristic, the *baseline* (*BL*), was developed as part of this GA research and the solution it found was incorporated into the initial population. Similar to the LMT heuristic, the baseline heuristic first levelizes the subtasks based upon their data dependencies. Then all subtasks are ordered such that a subtask at a higher level comes before one at a lower level. The subtasks in the same level are arranged in descending order of their numbers of output global data items (ties are broken arbitrarily). The subtasks are then scheduled in this order. Let the ith subtask in this order be σ_i, where $0 \leq i < |S|$. First, subtask σ_0 is assigned to a machine that gives the shortest completion time for σ_0. Then, the heuristic evaluates $|M|$ assignments for σ_1, each time assigning σ_1 to a different machine, with the previously decided machine assignment of σ_0 left unchanged. The subtask σ_1 is finally assigned to a machine that gives the shortest overall completion time for both σ_0 and σ_1. The baseline heuristic continues to evaluate the remaining subtasks in their order to be considered. When scheduling subtask σ_i, $|M|$ possible machine assignments are evaluated, each time with the previously decided machine assignments of subtasks σ_j $(0 \leq j < i)$ left unchanged. Subtask σ_i is finally assigned to a machine that gives the shortest overall completion time of subtasks σ_0 through σ_i. The total number of evaluations is thus $|S| \times |M|$, and only i subtasks (out of $|S|$) are considered when performing evaluations for the $|M|$ machine assignments for subtask σ_i.

Compared with the LMT and baseline nonevolutionary heuristics, the execution time of the GA-based approach was much greater, but it found much better solutions. This is appropriate for off-line matching and scheduling, rather than for real-time use (although in some applications, off-line precomputed GA mapping can be used on-line in real time [BuR97]).

To determine the best GA parameters for solving larger HC matching and scheduling problems, 50 larger scenarios were randomly generated in each communication category. Each of these scenarios contained 50 subtasks and five machines. For each scenario, 400 GA runs were conducted, half of which used the rank-based roulette selection scheme and the other half used the value-based roulette selection scheme. The 200 GA runs using the same selection scheme on each scenario had the following combinations of crossover probability and mutation probability. The crossover probability ranged from 0.1 to 1.0 in steps of 0.1, and the mutation probability ranged from 0.04 to 0.40 in steps of 0.04 and from 0.4 to 1.0 in steps of 0.1. Let the *relative solution quality* be the task completion time of the solution found by the LMT heuristic divided by that found by the approach being investigated. A greater value of the relative solution quality means that the approach being investigated finds a better solution to the HC matching and scheduling problem (i.e., with a shorter overall completion time for the application task represented by the

DAG). With each crossover and mutation probability pair and for each communication load, the average relative solution quality of the 50 GA runs, each on a different scenario, was computed. The following is a brief discussion and comparison of the rank-based and the value-based selection schemes, based on the experimental data obtained. Three-dimensional mesh and two-dimensional contour plots were used to analyze the experimental data. A detailed discussion and comparisons can be found in [Wan97].

Table I lists the best and worst average relative solution quality and the associated probabilities for each communication load with each selection scheme. The data in the table illustrates that the best solution found with the rank-based selection scheme was always better than that found with the value-based selection scheme in each communication load category. An analysis of the GA runs showed that the value-based selection scheme tended to improve the average fitness value of the population faster than the fitness value of the currently best chromosome. This caused the slot angle for the best chromosome in the population to decrease, thus reducing its possibility of selection in the search for better solutions.

For both selection schemes and each communication load category, a *region of good performance* could be identified for a range of crossover and mutation probabilities. The variation in the quality of solutions in each region of good performance was less than 33% of that over the entire range of crossover and mutation probabilities. In every case, this region of good performance also included the best relative solution quality. From Table I, it could be seen that the regions of good performance generally consisted of moderate to high crossover probability and low to moderate mutation probability. The values of the crossover and mutation probabilities in these regions are consistent with the results from the GA literature, which show that crossover is GA's major operator and mutation plays a secondary role in GA searches [Dav91, Gol89, SrP94].

With the rank-based selection scheme the regions of good performance were larger than those with the value-based selection scheme. Hence, the rank-based selection scheme was less sensitive to crossover and mutation probability selections to achieve good performance, whereas with the value-based selection scheme, one had to be careful in choosing crossover and mutation probabilities for the GA to find good solutions to the HC matching and scheduling problem.

Because the rank-based selection found better solutions and it was less sensitive to probability selections for good performance, it was chosen to be used for the larger scenarios. The crossover and mutation probabilities, as listed in Table I, with which the best relative solution quality had been achieved, were used in each corresponding communication load category. When matching and scheduling real tasks, the communication load can be determined by computing the ratio of the number of global data items to the number of subtasks. Once the communication load category is known, a probability pair from the corresponding region of good performance can be used.

TABLE I
Best and Worst Relative Solution Quality Found by the Rank-Based and Value-Based Selection Schemes with Associated Probabilities in Each Communication Load Category

Comm. load	Selection scheme	Best	Worst	Region of good performance
Light	Rank-based	Quality = 2.9138 P_{xover} = 0.4 P_{mut} = 0.40	Quality = 2.4692 P_{xover} = 0.5 P_{mut} = 1.00	Quality = 2.7876 to 2.9138 P_{xover} = 0.4 to 1.0 P_{mut}= 0.20 to 0.40
Light	Value-based	Quality = 2.7328 P_{xover} = 0.9 P_{mut} = 0.16	Quality = 2.2968 P_{xover} = 1.0 P_{mut} = 0.90	Quality = 2.6085 to 2.7328 P_{xover} = 0.6 to 0.9 P_{mut}= 0.12 to 0.24
Moderate	Rank-based	Quality = 2.7451 P_{xover} = 0.5 P_{mut} = 0.36	Quality = 2.1520 P_{xover} = 0.7 P_{mut} = 1.00	Quality = 2.5501 to 2.7451 P_{xover} = 0.3 to 1.0 P_{mut} = 0.20 to 0.50
Moderate	Value-based	Quality = 2.4424 P_{xover} = 0.9 P_{mut} = 0.12	Quality = 1.9615 P_{xover} = 1.0 P_{mut} = 1.00	Quality = 2.2958 to 2.4424 P_{xover} = 0.5 to 1.0 P_{mut} = 0.04 to 0.24
Heavy	Rank-based	Quality = 2.3245 P_{xover} = 1.0 P_{mut} = 0.20	Quality = 1.7644 P_{xover} = 0.1 P_{mut} = 1.00	Quality = 2.1568 to 2.3245 P_{xover} = 0.6 to 1.0 P_{mut}= 0.16 to 0.40
Heavy	Value-based	Quality = 2.0883 P_{xover} = 0.6 P_{mut} = 0.20	Quality = 1.6598 P_{xover} = 1.0 P_{mut} = 1.00	Quality = 1.9582 to 2.0883 P_{xover} = 0.5 to 1.0 P_{mut} = 0.16 to 0.24

Note. For each communication load category with each selection scheme, the rectangular region of good performance with the boundary crossover and mutation probabilities are listed. The best and worst relative solution quality within each region are also shown. In the table, P_{xover} is the crossover probability and P_{mut} is the mutation probability.

On Sun Sparc5 workstations, for these larger scenarios, both the LMT heuristic and the baseline heuristic took no more than 1 min of CPU time to execute. The average CPU execution time of the GA-based approach on these scenarios ranged from less than 1 min for the smallest scenarios (i.e., five subtasks, two machines, and light communication load) to about $3\frac{1}{2}$ h for the largest scenarios (i.e., 100 subtasks, 20 machines, and heavy communication load). Recall that it is assumed that this GA-based approach will be used for application tasks that are large production jobs such that the one time investment of this high execution time is justified.

The performance of the GA-based approach was also compared with that of a random search. For each iteration of the random search, a chromosome was randomly generated. This chromosome was evaluated and the fitness value was compared with the saved best fitness value. If the fitness value of the current chromosome was better than the saved best value, it became the saved best fitness value. For each scenario, the random search iterated for the same length of time as that taken by the GA-based approach on the same scenario.

Figure 14 shows the performance comparisons between the LMT heuristic and the GA-based approach for lightly communicating larger scenarios. In the figure, the horizontal axes are the number of subtasks in log scale. The vertical axes are the relative solution quality of the GA-based approach. The relative solution quality of the baseline (BL) heuristic and the random search is also shown in this figure. Each point in the figure is the average of 50 independent scenarios. The performance comparisons among the GA-based approach, the LMT heuristic, the baseline heuristic, and the random search for moderately communicating and heavily communicating larger scenarios are shown in Figs. 15 and 16, respectively.

In all cases, the GA-based approach presented here outperformed these other two heuristics and the random search. The improvement of the GA-based approach over the others showed an overall trend to increase as the number of subtasks increased. The exact shape of the GA-based-approach performance curves is not as significant as the overall trends because the curves are for a heuristic operating on randomly generated data, resulting in some varied performance even when averaged over 50 scenarios for each data point.

11. RELATED WORK

Different approaches to the HC matching and scheduling problem are difficult to compare. One of the reasons is that the HC models used vary from one approach to another. Furthermore, as discussed in Section 10, established test benchmarks do not exist at this time.

The most related research using GAs for HC includes [ShW96, SiY96, TiP96]. Our research significantly differs from the above approaches in terms of the HC models assumed. The following is a brief discussion of the related research work.

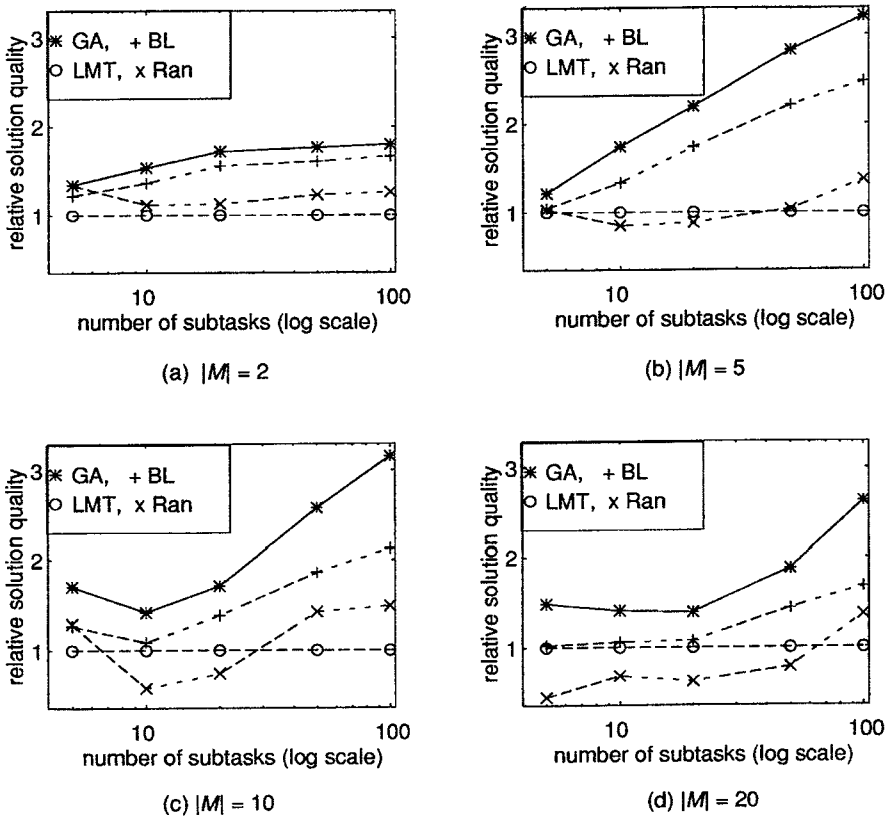


FIG. 14. Performance comparisons of the GA-based approach relative to the LMT heuristic for lightly communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic and the random search are also shown.

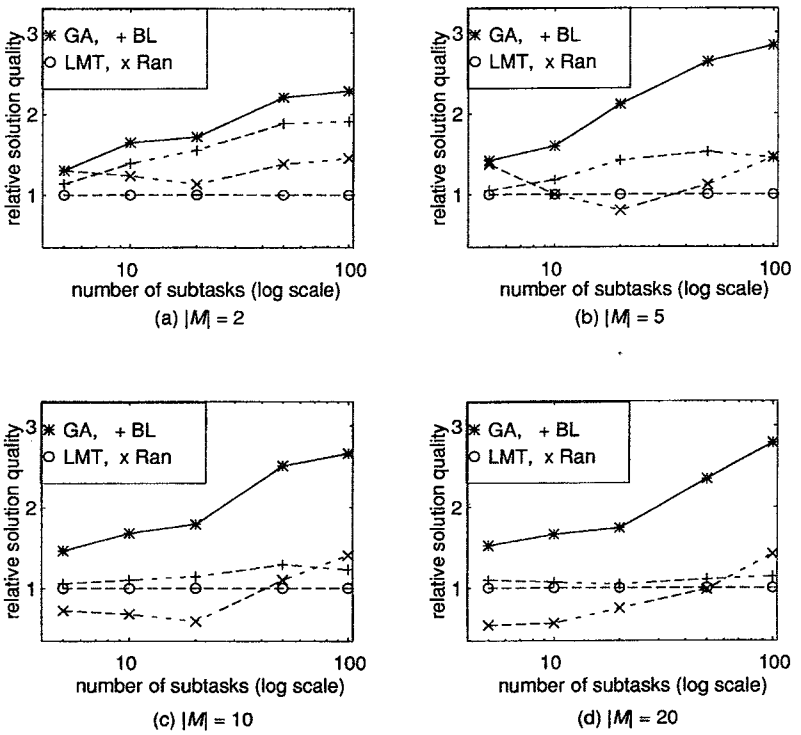


FIG. 15. Performance comparisons of the GA-based approach relative to the LMT heuristic for moderately communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite.The relative performance of the baseline heuristic and the random search are also shown.

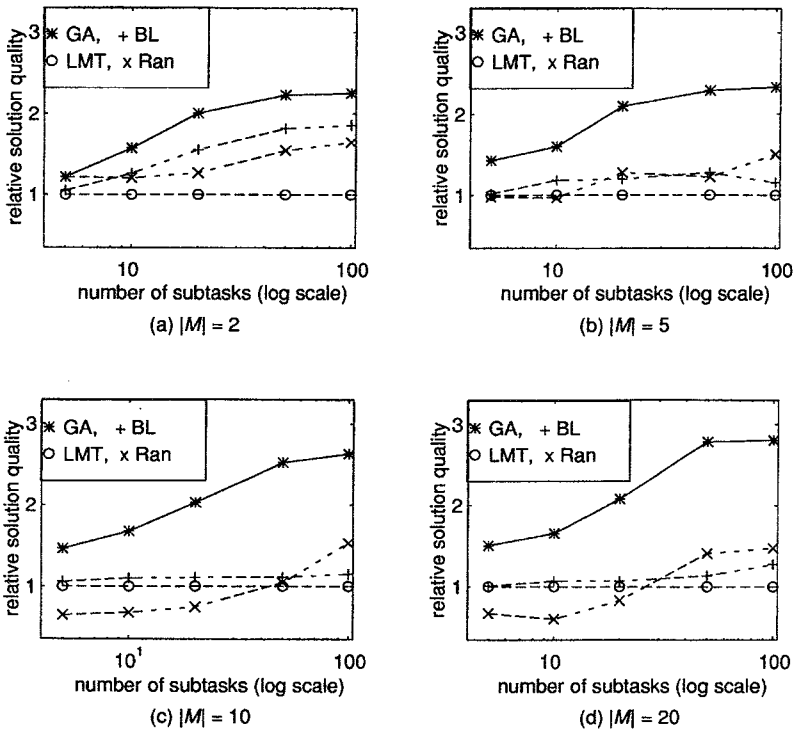


FIG. 16. Performance comparisons of the GA-based approach relative to the LMT heuristic for heavily communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic and the random search are also shown.

In [SiY96], a GA-based approach was proposed, in which the matcher/scheduler can utilize an unlimited number of machines as needed. In our proposed approach, however, an HC suite of a fixed number of machines is assumed. Another difference between these two approaches is that in [SiY96] a machine can send and receive data to and from an unlimited number of different machines concurrently. In our proposed approach, it is assumed that each machine has a single input link and a single output link such that all the input communications to one machine have to be serialized and all the output communications from one machine have to be serialized. A third difference between these two approaches is that in [SiY96] data can only be obtained from the original producer. In our proposed approach, however, data can be obtained either from the producer or from another subtask that has received the data. This is the data forwarding situation that was discussed in more detail in Section 9. Unlike the chromosome structure used in our proposed approach, which represents both matching and scheduling decisions, in [SiY96], a chromosome structure that only has the matching decision was used. Because of the assumptions made in [SiY96], for each matching decision an optimal scheduling can be computed.

Although a fully connected interconnection network is assumed in both [ShW96] and our proposed approach, in [ShW96] each machine can send to and receive from an unlimited number of different machines concurrently. Data forwarding is not utilized in [ShW96]. A simulated annealing technique was used in [ShW96] to do the chromosome selection. Similar to [SiY96], a chromosome structure that only has the matching decision was also used in [ShW96]. A nonrecursive algorithm was used in [ShW96] to determine a scheduling for each matching decision.

A GA-based approach in [TiP96] was used to design application-specific multiprocessor systems. Different from the goal set for this research, which is to minimize the total execution time, [TiP96] considered both the execution time and the system cost for a given application. In our approach, however, it is assumed that a machine suite is given, and the only goal is to minimize the completion time of the application.

12. CONCLUSION

A novel genetic-algorithm-based approach for task matching and scheduling in HC environments was presented. This GA-based approach can be used in a variety of HC environments because it does not rely on any specific communication subsystem models. It is applicable to the static scheduling of production jobs and can be readily used for scheduling multiple independent tasks (and their subtasks) collectively. For small-scale scenarios, the proposed approach found optimal solutions. For larger scenarios, it outperformed two nonevolutionary heuristics and a random search.

There are a number of ways this GA-based approach for HC task matching and scheduling may be built upon for future research. These include extending this approach to allow multiple producers for each of the global data items, parallelizing the GA-based approach, developing evaluation procedures for other communication subsystems, and considering loop and data-conditional constructs that involve multiple subtasks.

In summary, a novel GA design was developed for use in HC. This GA design has been shown to be a viable approach to the important problems of matching and scheduling in an HC environment.

ACKNOWLEDGMENTS

The authors thank M. Maheswaran, J. M. Siegel, M. D. Theys, and S. Wang for their valuable comments. A preliminary version of portions of this work was presented at the 5th Heterogeneous Computing Workshop (HCW'96).

REFERENCES

[AhD96] Ahmad, I., and Dhodhi, M. K. Multiprocessor scheduling in a genetic paradigm. *Parallel Comput.* **22,** 3 (Mar. 1996), 395–406.

[BeS94] Benten, M. S. T., and Sait, S. M. Genetic scheduling of task graphs. *Internat. J. Electron.* **77,** 4 (Apr. 1994), 401–405.

[BuR97] Budenske, J. R., Ramanujan, R. S., and Siegel, H. J. On-line use of off-line derived mappings for iterative automatic target recognition tasks and a particular class of hardware platforms. *Proc. 1997 Heterogeneous Computing Workshop (HCW'97).* IEEE Computer Society, Geneva, Switzerland, 1997, pp. 96–110.

[ChL88] Chen, C. L., Lee, C. S. G., and Hou, E. S. H. Efficient scheduling algorithms for robot inverse dynamic computation on a multiprocessor system. *IEEE Trans. Systems Man Cybernet.* **18,** 5 (Sept.–Oct. 1988), 729–743.

[CoL92] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1992.

[Dav91] Davis, L. (Ed.). *Handbook of Genetic Algorithms.* Van Nostrand–Reinhold, New York, 1991.

[EsS94] Eshaghian, M. M., and Shaaban, M. E. Cluster-M programming paradigm. *Internat. J. High Speed Comput.* **6,** 2 (Feb. 1994), 287–309.

[Fer89] Fernandez-Baca, D. Allocating modules to processors in a distributed system. *IEEE Trans. Softare Engrg.* **SE-15,** 11 (Nov. 1989), 1427–1436.

[Fre89] Freund, R. F. Optimal selection theory for superconcurrency. *Proc. Supercomputing '89.* IEEE Computer Society, Reno, NV, 1989, pp. 699–703.

[Fre94] Freund, R. F. The challenges of heterogeneous computing. *Parallel Systems Fair at the 8th International Parallel Processing Symp.* IEEE Computer Society, Cancun, Mexico, 1994, pp. 84–91.

[FrS93] Freund, R. F., and Siegel, H. J. Heterogeneous processing. *IEEE Comput.* **26,** 6 (June 1993), 13–17.

[GhY93] Ghafoor, A., and Yang, J. Distributed heterogeneous supercomputing management system. *IEEE Comput.* **26,** 6 (June 1993), 78–86.

[Gol89] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley, Reading, MA, 1989.

[HoA94] Hou, E. S. H., Ansari, N., and Ren, H. Genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Systems* **5,** 2(Feb. 1994), 113–120.

[Hol75] Holland, J. H. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, 1975.

[HoT89] Hoebelheinrich, R., and Thomsen, R. Multiple crossbar network integrated supercomputing framework. *Proc. Supercomputing '89.* IEEE Computer Society and ACM SIGARCH, Reno, NV, 1989, pp. 713–720.

[IvO95] Iverson, M. A., Ozguner, F., and Follen, G. J. Parallelizing existing applications in a distributed heterogeneous environment.

Proc. 1995 Heterogeneous Computing Workshop (HCW'95). IEEE Computer Society, Santa Barbara, CA, 1995, pp. 93–100.

[KhP93] Khokhar, A., Prasanna, V. K., Shaaban, M., and Wang, C. L. Heterogeneous computing: Challenges and opportunities. *IEEE Comput.* **26,** 6 (June 1993), 18–27.

[MuC69] Muntz, R. R., and Coffman, E. G. Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comput.* **C-18,** 11 (Nov. 1969), 1014–1020.

[NaY94] Narahari, B., Youssef, A., and Choi, H. A. Matching and scheduling in a generalized optimal selection theory. *Proc. 1994 Heterogeneous Computing Workshop (HCW'94).* IEEE Computer Society, Cancun, Mexico, 1994, pp. 3–8.

[RiT94] Ribeiro Filho, J. L., and Treleaven, P. C. Genetic-algorithm programming environments. *IEEE Comput.* **27,** 6 (June 1994), 28–43.

[Rud94] Rudolph, G. Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Networks* **5,** 1 (Jan. 1994), 96–101.

[ShW96] Shroff, P., Watson, D. W., Flann, N. S., and Freund, R. F. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 98–104.

[SiA96] Siegel, H. J., Antonio, J. K., Metzger, R. C., Tan, M., and Li, Y. A. Heterogeneous computing. In Zomaya, A. Y. (Ed.). *Parallel and Distributed Computing Handbook.* McGraw-Hill, New York, 1996, pp. 725–761.

[SiY96] Singh, H., and Youssef, A. Mapping and scheduling heterogeneous task graphs using genetic algorithms. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 86–97.

[SrP94] Srinivas, M., and Patnaik, L. M. Genetic algorithms: A survey. *IEEE Comput.* **27,** 6 (June 1994), 17–26.

[Sun92] Sunderam, V. S. Design issues in heterogeneous network computing. *Proc. Workshop on Heterogeneous Processing.* IEEE Computer Society, Beverly Hills, CA, March 1992, pp. 101–112.

[TaA95] Tan, M., Antonio, J. K., Siegel, H. J., and Li, Y. A. Scheduling and data relocation for sequentially executed subtasks in a heterogeneous computing system. *Proc. 1995 Heterogeneous Computing Workshop (HCW'95).* IEEE Computer Society, Santa Barbara, CA, April 1995, pp. 109–120.

[TiP96] Tirat-Gefen, Y. G., and Parker, A. C. MEGA: An approach to system-level design of application-specific heterogeneous multiprocessors. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 105–117.

[ToR93] Tolmie, D., and Renwick, J. HiPPI: Simplicity yields success. *IEEE Network* **7,** 1 (Jan. 1993), 28–32.

[WaA94] Watson, D. W., Antonio, J. K., Siegel, H. J., and Atallah, M. J. Static program decomposition among machines in an SIMD/SPMD heterogeneous environment with nonconstant mode switching costs. *Proc. 1994 Heterogeneous Computing Workshop (HCW'94).* IEEE Computer Society, Cancun, Mexico, April 1994, pp. 58–65.

[WaA96] Watson, D. W., Antonio, J. K., Siegel, H. J., Gupta, R., and Atallah, M. J. Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 24–37.

[Wan97] Wang, L. A genetic-algorithm-based approach for subtask matching and scheduling in heterogeneous computing environments and a comparative study on parallel genetic algorithms. Ph.D. thesis, School of Electrical and Computer Engineering, Purdue University, 1997.

[WeW94] Weems, C. C., Weaver, G. E., and Dropsho, S. G. Linguistic support for heterogeneous parallel processing: A survey and an approach. *Proc. 1994 Heterogeneous Computing Workshop (HCW'94).* IEEE Computer Society, Cancun, Mexico, April 1994, pp. 81–88.

[Whi89] Whitley, D. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Proc. 1989 International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA, June 1989, pp. 116–121.

Received February 1, 1997; received July 15, 1997; revised August 25, 1997

LEE WANG received a B.S.E.E. degree from Tsinghua University, Beijing, China; an M.S. degree in physics from Bowling Green State University, Bowling Green, Ohio, USA; and a Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, Indiana, USA, in 1990, 1992, and 1997, respectively. He worked as a research engineer for Architecture Technology Corporation during the summer of 1996. In June 1997, Dr. Wang joined Microsoft Corporation in Redmond, Washington, USA. His research interests include task matching and scheduling in heterogeneous computing environments, parallel languages and compilers, reconfigurable parallel computing systems, data parallel algorithms, distributed operating systems, and multimedia technology development. Dr. Wang has authored or coauthored one journal paper, six conference papers, two book chapters, and one language user's manual.

HOWARD JAY SIEGEL received B.S.E.E. and B.S. Management degrees from MIT (1972), and the M.A. (1974), M.S.E. (1974), and Ph.D. (1977) degrees in computer science from Princeton University. He is a Professor in the School of Electrical and Computer Engineering at Purdue University. His research interests include heterogeneous computing, parallel processing, interconnection networks, and the design and use of the PASM reconfigurable parallel machine. He is an IEEE Fellow (1990) and an ACM Fellow (1998), has coauthored more than 240 technical papers, was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*, and is a member of the Steering Committee of the annual Heterogeneous Computing Workshop sponsored by the IEEE Computer Society.

VWANI P. ROYCHOWDHURY received the B.Tech. degree from the Indian Institute of Technology, Kanpur, India, and the Ph.D. degree from Stanford University, Stanford, CA, in 1982 and 1989, respectively, all in Electrical Engineering. He is currently a professor in the Department of Electrical Engineering at the University of California, Los Angeles. From August 1991 to June 1996, he was a faculty member at the School of Electrical and Computer Engineering at Purdue University. His research interests include parallel algorithms and architectures, design and analysis of neural networks, application of computational principles to nanoelectronics, special purpose computing arrays, VLSI design, and fault-tolerant computation. He has coauthored several books including "Discrete Neural Computation: A Theoretical Foundation" (Prentice-Hall, Englewood Cliffs, NJ, 1995) and "Theoretical Advances in Neural Computation and Learning" (Kluwer Academic, Dordrecht, 1994).

ANTHONY A. MACIEJEWSKI received the B.S.E.E., M.S., and Ph.D. degrees in electrical engineering from The Ohio State University, Columbus, OH, in 1982, 1984, and 1987, respectively. In 1985–1986 he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan. Since 1988 he has been with the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana, where he is currently an Associate Professor. His primary research interests center on the simulation and control of kinematically redundant robotic systems.