A GENETIC-ALGORITHM-BASED APPROACH FOR TASK MATCHING AND SCHEDULING IN HETEROGENEOUS COMPUTING ENVIRONMENTS

Lee Wang, Howard Jay Siegel, and Vwani P. Roychowdhury
Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285 USA
e-mail: {lwang, hj, vwani}@ecn.purdue.edu

KEY WORDS: genetic algorithms; heterogeneous computing; matching; scheduling.

ABSTRACT

To exploit a heterogeneous computing (HC) environment, an application task may be decomposed into subtasks that have data dependencies. Task matching and scheduling consists of assigning subtasks to machines, ordering subtask execution for each machine, and ordering inter-machine data transfers. The goal is to achieve the minimal completion time for the task. The corresponding matching and scheduling problem is known to be NP-complete. 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. Various 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 a non-evolutionary heuristic.

This research was supported in part by NRaD under subcontract number 20-950001-70.

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 [1, 2, 3, 4]. The goal of HC is to achieve the minimal completion time, i.e., the overall execution time of the application task in the machine suite. In an HC environment, 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 inter-machine data transfers.

It is well known that such a matching and scheduling problem is in general NP-complete [5]. A number of approaches to different aspects of this problem have been proposed (e.g., [1, 6, 7, 8, 9, 10]). Different from the above approaches, this paper proposes a genetic-algorithm-based approach for solving the problem.

A genetic algorithm for subtask scheduling in a collection of identical processors is discussed in [11]. The resulting scheduling satisfies the precedence constraints among the subtasks. Data transfers and the use of heterogeneous machines, however, are not considered there, and they are in this paper.

In [7], a non-evolutionary heuristic based on level scheduling [12, 13] 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 chromosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. This genetic-algorithm-based approach can be applied to performing the matching and scheduling in a variety of HC systems.

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 goodness of a solution and the experimental results are shown in Section 10. Finally, Section 11 discusses some future research directions.

2. PROBLEM DEFINITION

To focus on the matching and scheduling problem, it is assumed that the application task is written in some machine-independent language (e.g., [14]). 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 inter-machine 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*. Finding the estimated expected execution times for subtasks is another research problem, which is beyond the scope of this paper. Approaches based on task profiling and analytical benchmarking are surveyed in [4]. The HC system is assumed to have an operating system support for executing each subtask on the machine it is assigned and for performing inter-machine data transfers as scheduled by this genetic-algorithm-based approach.

In an HC system, an application task is decomposed into a set of subtasks Γ. Define $|\Gamma|$ to be the number of subtasks in the set Γ and γ_i to be the i-th subtask. Then $\Gamma = \{ \gamma_i, 0 \le i < |\Gamma| \}$. An HC environment consists of a set of machines Π. Define $|\Pi|$ to be the number of machines in the set Π and π_j to be the j-th machine. Then $\Pi = \{\pi_j, 0 \le j < |\Pi| \}$. The estimated expected execution time of subtask γ_i on machine π_j is ET_{ij}, where $0 \le i < |\Gamma|$ and $0 \le j < |\Pi|$. The global data items (*gdi*s), 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 k-th global data item. Then $G = \{gdi_k, 0 \le k < |G| \}$.

Initially, 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). Hence, the task is represented by a single–producer directed acyclic graph (SPDAG). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred over it. Figure 1 shows an example SPDAG. This approach has been extended to allow the global data items to have multiple producers [15].

The following further assumptions are made for the problem. One is the exclusive use of the HC environment for the application. The GA-based matcher/scheduler is in control of the HC machine suite. Another is non-preemptive 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. Any loops and data conditionals are assumed to be contained inside subtasks. 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.

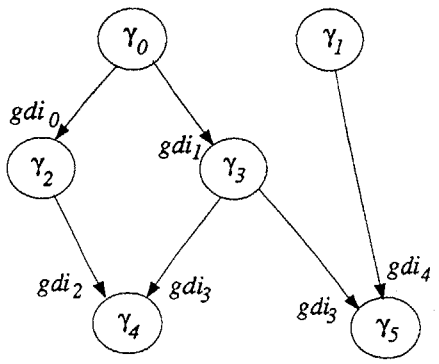


Figure 1: An example SPDAG.

3. GENETIC ALGORITHMS

Genetic algorithms (GAs) are a promising heuristic approach to finding near-optimal solutions in large search spaces [16, 17, 18]. There are a great variety of approaches to GAs; many are surveyed in [19, 20]. 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 some of the possible solutions to the optimization problem as a set of strings (chromosomes). 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. The objective of the GA search is to find a chromosome that has the optimal 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. If the stopping criteria have not been met, the new generation goes through another cycle (iteration) of selection, crossover, and mutation. 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) a fitness function, (4) a selection mechanism, (5) a crossover mechanism, (6) a mutation mechanism, and (7) stopping criteria. The outline of the proposed GA-based approach is shown in Figure 2. Details of the steps in this approach will be discussed in the following sections.

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

Figure 2: Outline of the GA-based approach.

4. CHROMOSOME REPRESENTATION

In this GA-based approach, each chromosome is presented by a two-tuple. Let *mat* be the matching string, which is a vector of length $|\Gamma|$, such that $mat(i) = \pi_j$, where $0 \le i < |\Gamma|$ and $0 \le j < |\Pi|$, i.e., subtask γ_i is assigned to machine π_j.

The scheduling string is a topological sort [21] of the SPDAG, i.e., a total ordering of the nodes (subtasks) in the SPDAG that obeys the precedence constraints. Define *ss* to be the scheduling string, which is a vector of length $|\Gamma|$, such that $ss(k) = \gamma_i$, where $0 \le i,k < |\Gamma|$, and each γ_i appears only once in the vector, i.e., subtask γ_i is the k-th subtask in the scheduling string. Because it is a topological sort, if $ss(k)$ is a consumer of a global

data item produced by $ss(j)$, then $j < k$. The scheduling string gives the execution ordering of the subtasks assigned to the same machine: for two such subtasks, the one closer to the top of the *ss* will be executed first. For subtasks assigned to different machines, execution scheduling is determined by the data dependencies among them. Thus, different scheduling strings may represent the same overall schedule.

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 left to the evaluation step. Figure 3 illustrates two chromosomes from the SPDAG in Figure 1, for $|\Gamma| = 6$, $|\Pi| = 3$, and $|G| = 5$.

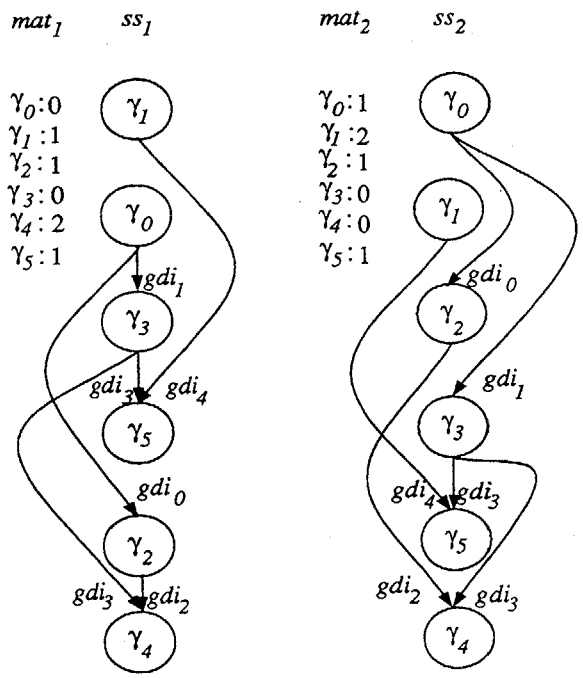


Figure 3: Two chromosomes from the SPDAG in Figure 1.

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 assigning each subtask to a machine randomly. To form a scheduling string, the SPDAG 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 SPDAG). Furthermore, it is common in GA applications to incorporate solutions from some non-evolutionary heuristics into the initial population, which may reduce the time needed for finding a satisfactory solution [16]. 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 level-scheduling heuristic. Details of this heuristic will be discussed in Section 9.

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 non-zero 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 [15]. 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 is used to implement proportionate selection [18, 19]. In this scheme, each chromosome is allocated a sector on the roulette wheel. Let C denote the population size and S_i denote the angle of the sector allocated to the i-th ranked chromosome. The 0-th ranked chromosome is the fittest and has the sector with the largest angle S_0; whereas the $(C-1)$-th ranked chromosome is the least fit and has the sector with the smallest angle S_{C-1}. The ratio of the sector angles between two adjacently ranked chromosomes is a constant $R = S_i / S_{i+1}$, where $0 \le i < C-1$. If the 360 degrees of the wheel is normalized to one, then

$$S_i = R^{C-i-1} \times (1-R) / (1-R^C)$$

where $R > 1, 0 \le i < C$, and $0 < S_i < 1$.

The selection step generates C 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 C, and it is possible to have multiple copies of the same chromosome.

This GA-based approach also incorporates elitism, i.e., the best solution found so far is always maintained in the population [22]. 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 cuts 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 cut both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

8. MUTATION OPERATORS

Different mutation operators are developed for scheduling strings and matching strings. The scheduling string mutation operator randomly chooses some scheduling strings. Then for each chosen scheduling string, it randomly selects a victim subtask. The free 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 free range is below the source

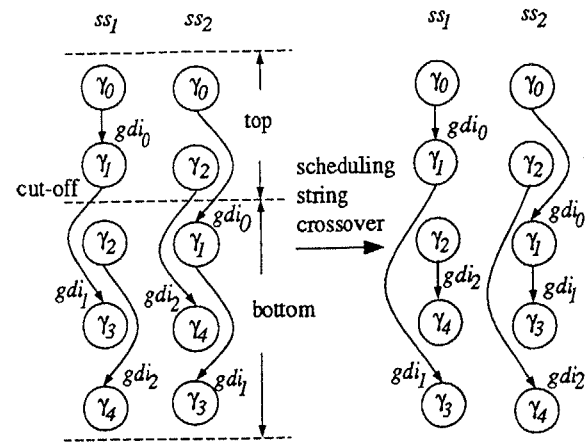


Figure 4: A scheduling string crossover example

subtask of the victim's shortest input edge and above the destination subtask of the victim's shortest output edge in the scheduling string. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its free 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.

9. EVALUATION

The final step of an evaluation 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 demonstrate the evaluation process, a communication subsystem, which is modeled after a HiPPI LAN with a central crossbar switch [23, 24], 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. (Currently, communication times are source and destination machine independent; machine dependence is being added.) Data transfers are neither preemptive nor

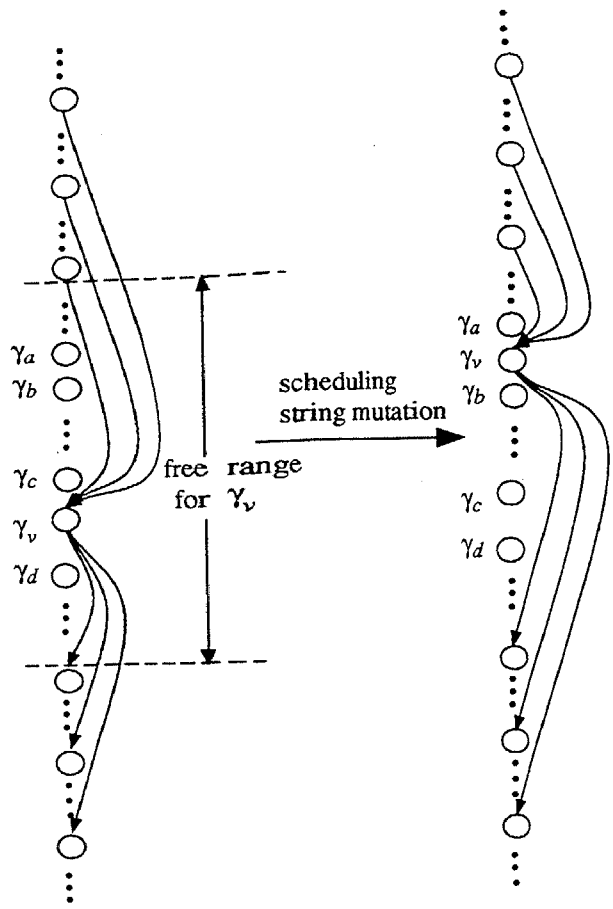


Figure 5: A scheduling string mutation example. Only edges to and from the victim subtask γ_v are shown. Before the mutation, γ_v is between γ_c and γ_d. After the mutation, it is moved up to between γ_a and γ_b.

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.

In this step, each chromosome is evaluated, i.e., both the execution of all of the subtasks and all of the data transfers are scheduled. The evaluation procedure evaluates the subtasks in the order they appear on the scheduling string. Thus, the subtasks assigned to the same machine are executed exactly in this order. For subtasks assigned to different machines, execution scheduling is determined by the data dependencies among them. Before a subtask can be scheduled, all of its input global data items have to be received. For each subtask, its input data items are considered by the evaluation procedure in the order of their producers' relative

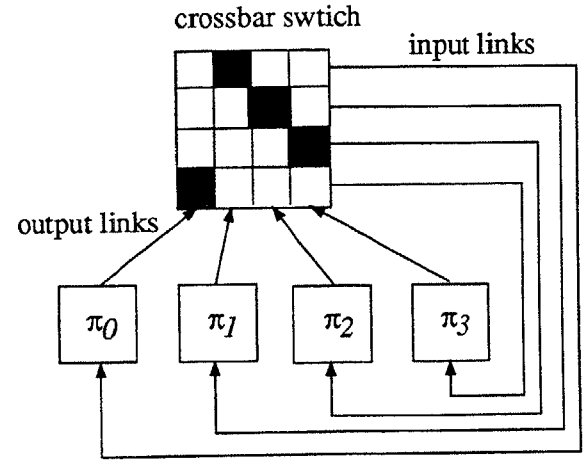


Figure 6: An example HC system with four machines and a central crossbar switched network. Each machine has one output data link to and one input data link from the crossbar switch. Blackened squares in the switch correspond to active connections.

positions in the scheduling string. Figure 7 demonstrates the scheduling order. In Figure 8, a simple example is shown to illustrate the evaluation for a given chromosome.

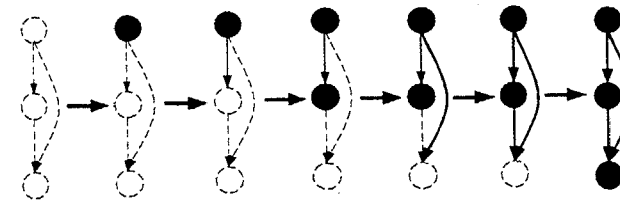
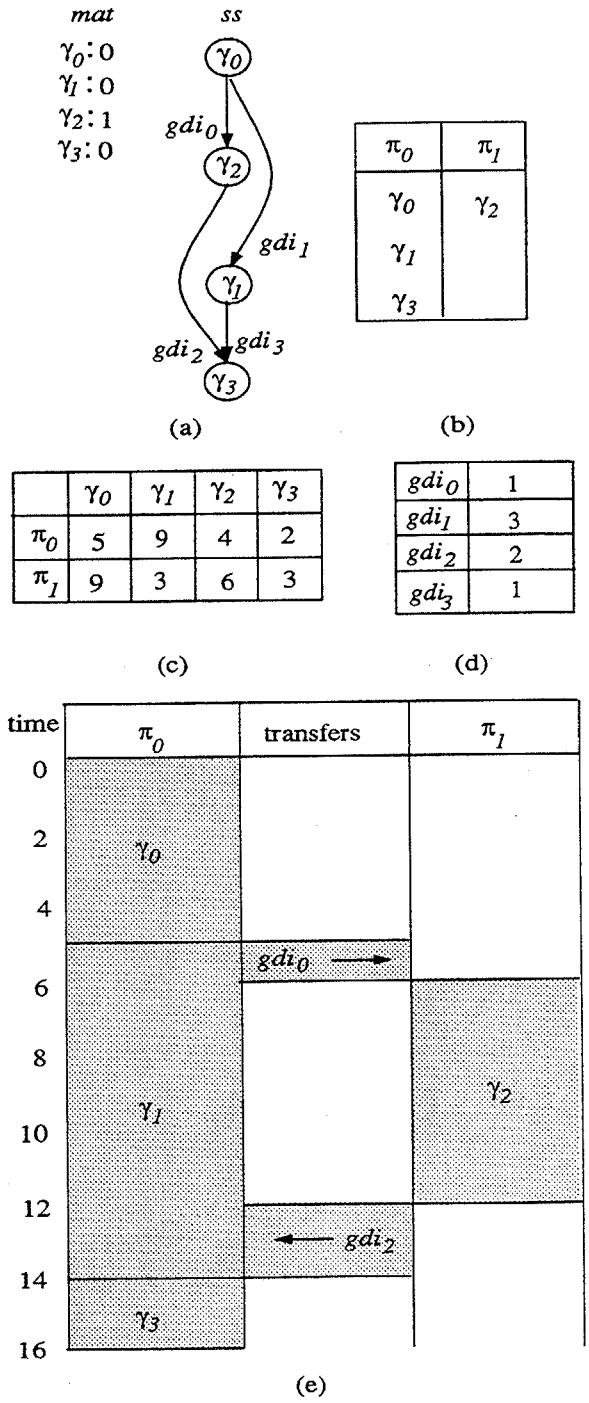


Figure 7: Scheduling order in the evaluation step. The example scheduling string has three subtasks and three global data item transfers, represented by circles and arrows, respectively. Dotted circles and arrows indicate that those subtasks and *gdi* transfers are not scheduled yet. Blackened circles and solid arrows indicate that they have been scheduled.

When a subtask to be scheduled has multiple input *gdi*s that have not been received, the global data item that has the farthest source subtask is scheduled first. The reason for this ordering is to better utilize the overlap of subtask executions and inter-machine data communications. The following example illustrates this idea. Let a subtask γ_0 be above another subtask γ_1 and γ_1 be above yet another subtask γ_2 in the scheduling string, as shown in Figure 9(a). Let γ_2 need two *gdi*s, gdi_0 and gdi_1, from γ_0 and γ_1, respectively. Depending on the subtask to machine assignments, the data transfers of gdi_0 and gdi_1



π_0	π_1
γ_0	γ_2
γ_1	
γ_3	

	γ_0	γ_1	γ_2	γ_3
π_0	5	9	4	2
π_1	9	3	6	3

gdi_0	1
gdi_1	3
gdi_2	2
gdi_3	1

Figure 8: An example showing the evaluation: (a) the chromosome; (b) the subtask execution orderings on each machine given by (a); (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, where the completion time for this chromosome is 16.

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 γ_0 and γ_1 be assigned to the same machine π_0 and γ_2 be assigned to another machine π_1, as shown in Figure 9(b). In this situation, because γ_0 is to be executed before γ_1, gdi_0 is available before gdi_1 becomes available on machine π_0. Thus, it is better to schedule the gdi_0-transfer before the gdi_1-transfer.

Situation 2: Let the three subtasks γ_0, γ_1, and γ_2 be assigned to three different machines π_0, π_1, and π_2, as shown in Figure 9(c). In this situation, if there is a data dependence from γ_0 to γ_1, then γ_0 finishes its execution before γ_1 could start its. 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 is no data dependencies from γ_0 to γ_1, as long as γ_0 is above γ_1 in the scheduling string, the gdi_0-transfer can still be scheduled before the gdi_1-transfer. The reason behind this is that if there is no data dependency from γ_0 to γ_1, there may be some other chromosome(s) that have γ_1 above γ_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 exist.

Data forwarding is another important feature of this evaluation process. For each input data item to be considered, the procedure 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 one (either the producer or a forwarder) from which the destination subtask will receive the data item at the earliest possible time is chosen.

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 free 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 executions earlier, which would in turn decrease the overall task completion time. Figure 10 and Figure 11 show the in-order scheduling and the out-of-order

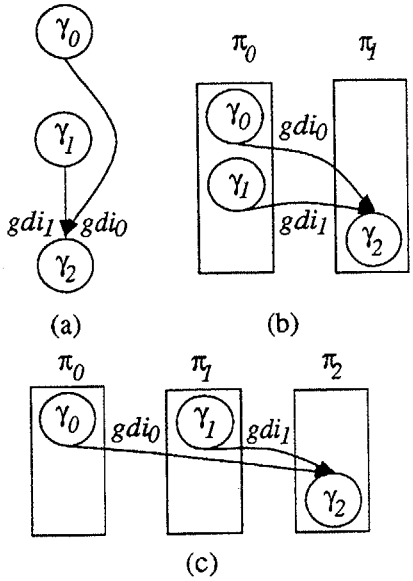


Figure 9: An example showing the scheduling order for the input *gdi*s of one subtask: (a) the example scheduling string; (b) the situation when the source subtasks of the input *gdi*s are assigned to the same machine; (c) the situation when the source subtasks of the input *gdi*s are assigned to different machines.

scheduling for the same chromosome, respectively. In this example, the out-of-order schedule does decrease the total execution time of the given task.

Because the particular communication subsystem that is used to demonstrate the evaluation procedure in this paper has a central crossbar switch network, an edge on a scheduling string, i.e., a logical path from the source subtask to the destination subtask, has an equivalent physical path from the source machine to the destination machine. There are no intermediate machines in the physical path.

However, if a communication subsystem is not fully connected, then a logical path for a *gdi* transfer across machines (corresponding to an edge in the scheduling string) might have multiple physical paths from the source machine to the destination machine. In this case, the polynomial single-pair-shortest-path algorithm [21] could be used to choose a shortest physical path that takes the least time for each *gdi* transfer. Furthermore, when the *gdi* is transferred over the chosen physical path, copies of this *gdi* can be dropped to some intermediate machines when it is en route to the destination machine, provided some subtasks on these machines will need this *gdi* later. Thus, at the time when these subtasks are to be

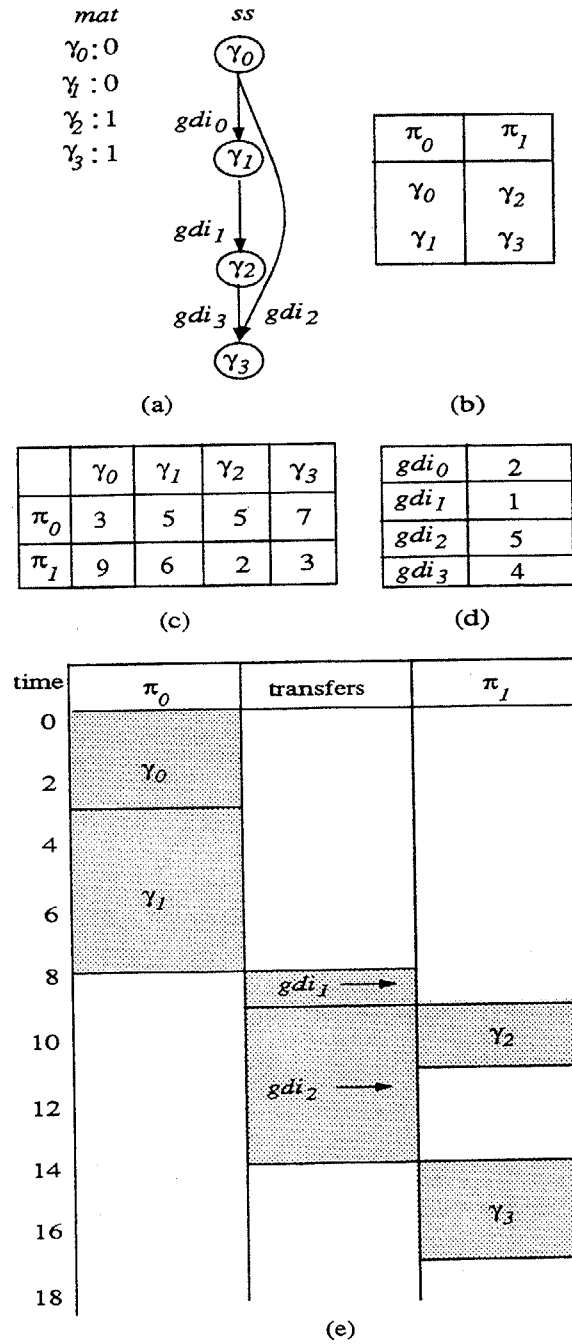


Figure 10: An example showing the in-order scheduling: (a) the chromosome; (b) the subtask execution ordering on each machine; (c) the estimated subtask execution times; (d) the *gdi* transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution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

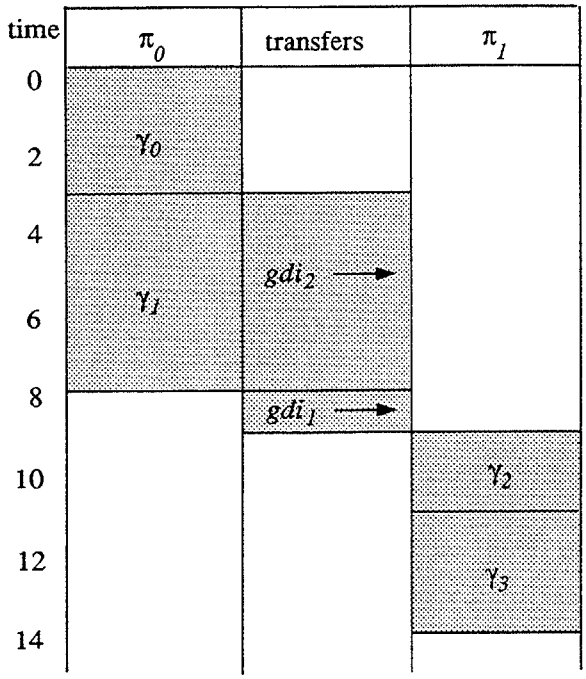


Figure 11: An example showing the the out-of-order scheduling, where the chromosome is the same as in Figure 10. The completion time is 14.

scheduled, they have had this *gdi* already and it is unnecessary to schedule the transfers for this *gdi* for these subtasks. This is the multiple data copy situation. This situation could further decrease the completion time as the result of the reduced number of *gdi* transfers and, in turn, possible earlier start times for some subtasks.

After a chromosome is evaluated, it is associated with a fitness value, which is the time when the last subtask finishes its execution. 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 schedules subtasks in their order in the scheduling string and schedules the input *gdi*s for a subtask before it can be scheduled. For a subtask that requires some *gdi*s from other machines, the *gdi*-transfer with the farthest source subtask 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 the needing 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 running on an HC suite of machines. In the case that the machines are not fully connected, when considering the *gdi*-transfer from each producer or forwarder, the physical path that takes the least transfer time is chosen. Also, copies of this *gdi* are dropped at some intermediate machines if these machines have some subtasks that will need this *gdi* later.

10. EXPERIMENTAL RESULTS WITH SIMULATED PROGRAM BEHAVIOR

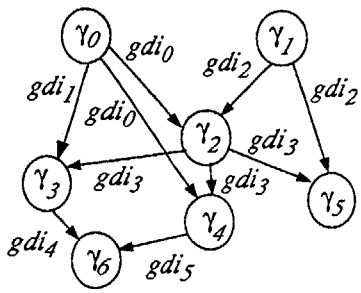
Randomly generated task behaviors (i.e., SPDAGs and the associated subtask execution times and global data item transfer times) were used to measure the performance of this GA-based approach. The tests were generated for different numbers of subtasks and different numbers of machines, as specified below. The estimated expected execution time for each subtask on each machine, the number of global data items, and the time needed for each data item transfer were uniformly randomly generated within some predefined ranges. The producer and consumers of each global data item were also generated randomly. The test generation guaranteed that the precedence constraints from data dependencies were acyclic.

These randomly generated task behaviors 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 [25]. Determining a representative set of HC task benchmarks remains a current and unresolved challenge for the scientific community in this research area.

The GA runs had the following parameters. The probabilities for scheduling string crossovers, matching string crossovers, scheduling string mutations, and matching string mutations were 0.4, 0.4, 0.1, 0.1, respectively. This set of numbers was selected by experimentation. With them, the GA-based approach achieved the best performance on the small-scale tests described below, compared with other sets of numbers tested. The angle ratio of the sectors on the roulette selection wheel for two adjacently ranked chromosomes, R, is chosen to be $1 + 1 /$ (number of chromosomes). A value that decreased as the number of chromosomes increased was selected to ensure each chromosome had a reasonable size sector. Further tests will be conducted to evaluate other possible R values. The GA runs stopped if either the number of iterations reached 1000, the population converged (all the chromosomes had the same fitness value), or the best solution found was not improved after 150 iterations. Small-scale and larger tests were con-

ducted to quantify the performance of the GA-based approach.

The results from a small-scale test is used here to illustrate the search process. This test had $|\Gamma| = 7$, $|\Pi| = 3$, and $|G| = 6$. The SPDAG, the estimated execution times, and the transfer times of the global data items are shown in Figures 12(a), 12(b), and 12(c), respectively. The total numbers of possible different matching strings and different valid scheduling strings (i.e., topological sorts of the SPDAG) were $3^7 = 2187$ and 16 (the number of possible valid topological sorts), respectively. Thus, the total search space had $2187 \times 16 = 34992$ possible chromosomes. The population size for this test was chosen to be 50. In this example, the transfer times for the global data items are source and destination machine independent (this GA-based approach is being extended to include the situations when the transfer times are source and destination machine dependent).



(a)

	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6
π_0	872	251	542	40	742	970	457
π_1	898	624	786	737	247	749	451
π_2	708	778	23	258	535	776	15

(b)

gdi_0	347
gdi_1	883
gdi_2	44
gdi_3	589
gdi_4	275
gdi_5	709

(c)

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

Figure 13 depicts the evolution process of this test. 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 (solution) is. Multiple identical chromosomes at the same point are not differentiated. Figures 13(a)-13(d) show the distributions of the distinct chromosomes at iterations 0, 20, 40, and 43, respectively. The GA run converged at iteration 43, as shown in Figure 13(d), where all the chromosomes had the same height. Convergence does not necessarily mean that all chromosomes become identical; this GA-based approach could find multiple best solutions that have the same completion time.

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. In both of these cases, two different scheduling strings could result in the same ordering for (1) and (2).

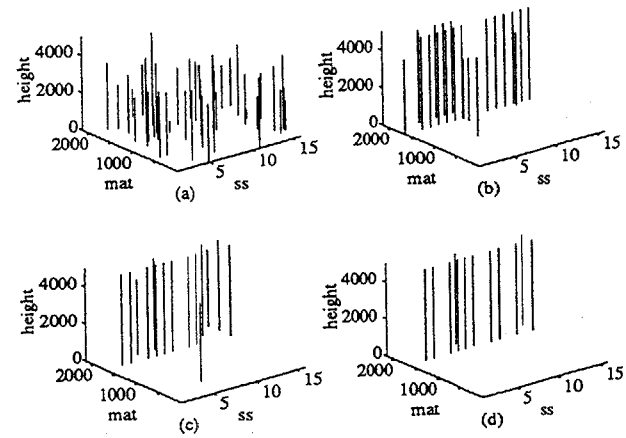


Figure 13: Evolution of a GA run for the test in Figure 12. (a) at iteration 0, (b) at iteration 20, (c) at iteration 40, and (d) at iteration 43, when the search converges.

Exhaustive searches were performed to find the optimal solutions for 20 small-scale tests that involved up to ten subtasks, three machines, and seven global data items. For each of the small-scale tests that were conducted, the GA-based approach found one or more optimal solutions that had the same completion time as that of the optimal solution found by exhaustive searches. The GA search for a small-scale test that had ten subtasks, three machines, and seven global data items took about one minute to find multiple optimal solutions on a Sun Sparc5 workstation while the exhaustive search took about eight hours on a Sun Sparc10 workstation to find a single optimal solution.

The larger tests with up to 100 subtasks and 20 machines were conducted. They fall in three categories: lightly, moderately, and heavily communicating tasks. A lightly communicating task has its number of global data items in the range of $0 \leq |G| < (1/3)|\Gamma|$; a moderately communicating task has its number of global data items in the range of $(1/3)|\Gamma| \leq |G| < (2/3)|\Gamma|$; and a heavily communicating task has its number of global data items in the range of $(2/3)|\Gamma| \leq |G| < |\Gamma|$. The ranges for the global data item transfer times and for the estimated subtask execution times were both from 1 to 1,000. Thus, data transfers were comparable in time with subtask execution. The population size for these larger tests was chosen to be 200.

Larger tests are intractable problems. It is currently impractical to directly compare the quality of the solutions found by the GA-based approach for these larger tests 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. However, the model used in [7] is similar to the one being used in this research work. Hence, the performance of the GA-based approach on larger tests was compared with the levelized min-time (LMT) heuristic proposed in [7].

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 rest of the subtasks is at one level below the lowest producer of its global data items. The subtasks at the highest level are to be scheduled first. The LMT heuristic then averages the estimated execution times for each subtask across all machines and uses these averages as the criteria to decide the priorities of the subtasks within each level, i.e., a subtask with a larger average is to be scheduled 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 to the number of machines available. Along with the averaged execution time of each subtask, the priority of a subtask is also affected by the number of levels between its level and the level of the closest child subtask that needs some data from this subtask. Farther the closest child subtask, the lower the priority of the producer subtask has. When a subtask is to be scheduled, it is assigned to the fastest machine available from those machines that have not yet been assigned any subtasks from the same level.

Another non-evolutionary heuristic, baseline (BL), was developed 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 the descending order of their numbers of output global data items (ties are broken arbitrarily). The subtasks are then scheduled in this order. When a subtask is to be scheduled, the heuristic evaluates $|\Pi|$ assignments, each time assigning the subtask to a different machine. This subtask is finally assigned to the machine that gives the shortest partial completion time for the subtasks that have been scheduled (including this subtask).

Figure 14 shows the performance comparisons between the LMT heuristic and the GA-based approach for lightly communicating larger tests. In the figure, the horizontal axes are the number of subtasks in log scale. The vertical axes are the relative solution quality, which is defined as the task completion time of the solution found by the LMT heuristic divided by that found by the approach being plotted. The relative solution quality of the baseline (BL) heuristic is also shown in this figure. Each point in the figure is the average of 50 independent tests. The performance comparisons among the GA-based approach, the LMT heuristic, and the baseline heuristic for moderately communicating and heavily communicating larger tests are shown in Figure 15 and Figure 16, respectively.

In all cases, the GA-based approach presented here outperformed these other two heuristics. The improvement of the GA-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 tests for each data point.

11. CONCLUSION

A novel genetic-algorithm-based approach for task matching and scheduling in HC environments was presented. This GA-based approach can be used to schedule application tasks 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. This GA-based approach has also been extended to do the matching and scheduling for ap-

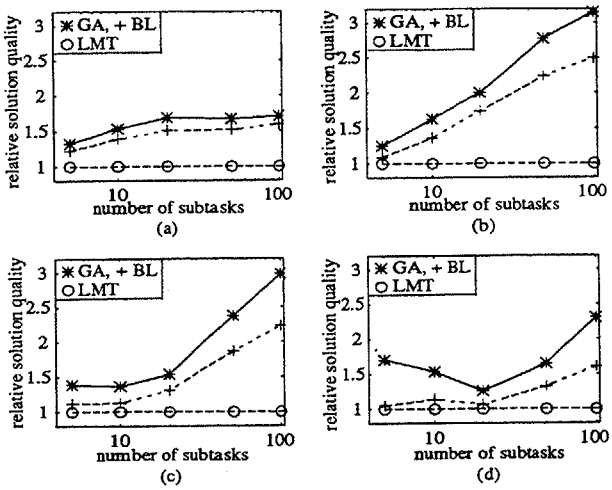


Figure 14: Performance comparisons of the GA-based approach relative to the LMT heuristic for lightly communicating larger tests in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic is also shown.

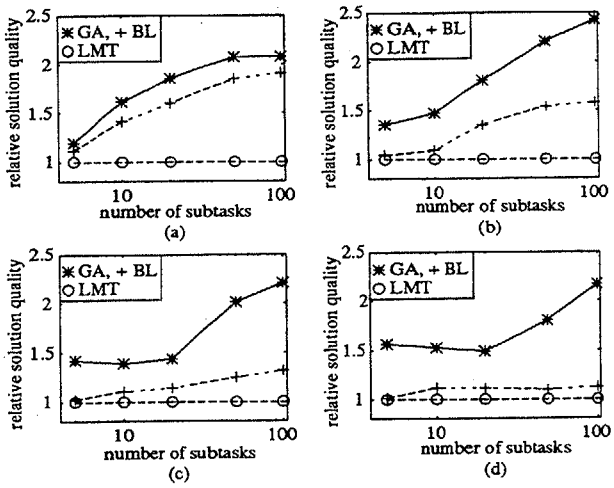


Figure 15: Performance comparisons of the GA-based approach relative to the LMT heuristic for moderately communicating larger tests in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic is also shown.

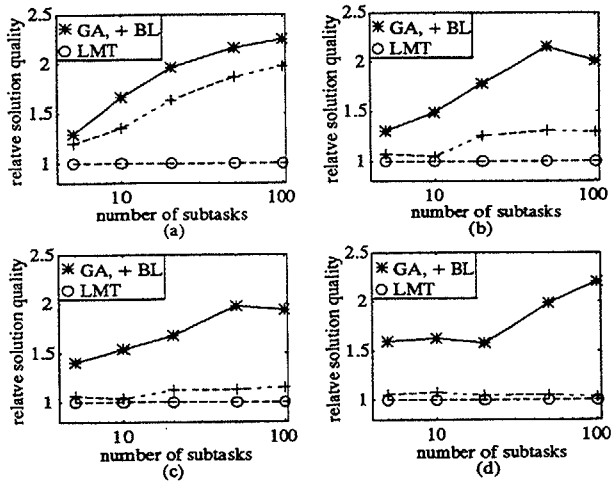


Figure 16: Performance comparisons of the GA-based approach relative to the LMT heuristic for heavily communicating larger tests in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic is also shown.

plication tasks that have multiple producers for some global data items [15]. For small-scale tests, the proposed approach found optimal solutions. For larger tests, it outperformed a non-evolutionary heuristic.

There are a number of parameters used in genetic algorithms, e.g., the probabilities of the genetic operators, the population size, and the criteria to stop the evolution. Extensive tests are being conducted to find the best parameters for this GA-based approach. Other ongoing and future research includes 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, GAs have been shown to be a viable approach to the important problems of matching and scheduling in an HC environment. Future research will focus on additional testing and refinement of this approach.

Acknowledgments: We thank M. Maheswaran and M. D. Theys for their proofreading and valuable comments.

REFERENCES

[1] R. F. Freund, "Optimal Selection Theory for Superconcurrency," in *Proceedings of Supercomputing '89*, November 1989, pp. 699-703.

[2] R. F. Freund and H. J. Siegel, "Heterogeneous Processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[3] 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.

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

[5] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Software Engineering*, Vol. SE-15, No. 11, November 1989, pp. 1427-1436.

[6] M. M. Eshaghian and M. E. Shaaban, 1994, "Cluster-M Programming Paradigm," *International Journal of High Speed Computing*, Vol. 6, No. 2, June 1994, pp. 287-309.

[7] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," in *Proceedings of 1995 Heterogeneous Computing Workshop (HCW '95)*, April 1995, pp. 93-100.

[8] B. Narahari, A. Youssef, and H. A. Choi, "Matching and Scheduling in a Generalized Optimal Selection Theory," in *Proceedings of 1994 Heterogeneous Computing Workshop (HCW '94)*, April 1994, pp. 3-8.

[9] M. Tan, J. K. Antonio, H. J. Siegel, and Y. A. Li, "Scheduling and Data Relocation for Sequentially Executed Subtasks in a Heterogeneous Computing System," in *Proceedings of 1995 Heterogeneous Computing Workshop (HCW '95)*, April 1995, pp. 109-120.

[10] 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," in *Proceedings of 1994 Heterogeneous Computing Workshop (HCW '94)*, April 1994, pp. 58-65.

[11] E. S. H. Hou, N. Ansari, and H. Ren, "Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, No. 2, February 1994, pp. 113-120.

[12] C. L. Chen, C. S. G. Lee, and E. S. H. Hou, "Efficient Scheduling Algorithms for Robot Inverse Dynamic Computation on a Multiprocessor System," *IEEE Trans. Systems, Man, Cybernetics*, Vol. 18, No. 5, September-October 1988, pp. 729-743.

[13] R. R. Muntz and E. G. Coffman, "Optimal Preemptive Scheduling on Two-Processor Systems," *IEEE Trans. Computers*, Vol. C-18, No. 11, November 1969, pp. 1014-1020.

[14] C. C. Weems, G. E. Weaver, and S. G. Dropsho, "Linguistic Support for Heterogeneous Parallel Processing: A Survey and an Approach," in *Proceedings of 1994 Heterogeneous Computing Workshop (HCW '94)*, April 1994, pp. 81-88.

[15] L. Wang, H. J. Siegel, and V. P. Roychowdhury, "A Genetic-Algorithm-Based Heuristic for the Matching and Scheduling Problems in a Heterogeneous Computing System," *Technical Report*, ECE School, Purdue, in preparation.

[16] L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, 1991.

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

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

[19] M. Srinivas and L. M. Patnaik, "Genetic Algorithms: A Survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17-26.

[20] J. L. Ribeiro Filho and P. C. Treleaven, "Genetic-Algorithm Programming Environments," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 28-43.

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

[22] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms," *IEEE Trans. Neural Networks*, Vol. 5, No. 1, January 1994, pp. 96-101.

[23] R. Hoebelheinrich and R. Thomsen, "Multiple Crossbar Network Integrated Supercomputing Framework," in *Proceedings of Supercomputing '89*, November 1989, pp. 713-720.

[24] D. Tolmie and J. Renwick, "HiPPI: Simplicity Yields Success," *IEEE Network*, Vol. 7, No. 1, January 1993, pp. 28-32.

[25] D. W. Watson, J. K. Antonio, H. J. Siegel, R. Gupta, and M. J. Atallah, "Static Matching of Ordered Program Segments to Dedicated Machines in a Heterogeneous Computing Environment," in *Proceedings of 1996 Heterogeneous Computing Workshop (HCW '96)*, April 1996, in this proceedings.

AUTHOR BIOGRAPHIES

Lee Wang is a Ph.D. candidate in School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana. His research interests include task matching and scheduling in heterogeneous computing environments, parallel languages, reconfigurable parallel computing systems, data parallel algorithms, distributed computing, automatic parallelization techniques, distributed operating systems, and parallel simulations. He is the student coordinator of a parallel language research project and a codeveloper of a graduate-level course on programming parallel machines. He has authored or coauthored four conference papers, two book chapters, and one language user's manual.

Mr. Wang received a BSEE degree from Tsinghua University, Beijing, China, in 1990 and an MS degree in physics from Bowling Green State University, Bowling Green, Ohio, in 1992. From July 1992 to December 1992, he was a research associate in Department of Electrical Engineering at the Ohio State University, Columbus, Ohio. Mr. Wang is a member of the IEEE and the IEEE Computer Society.

Howard Jay Siegel is a Professor and Coordinator of the Parallel Processing Laboratory in the School of Electrical and Computer Engineering at Purdue University. He received B.S. degrees in both electrical engineering and management from MIT, and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has coauthored over 200 technical papers, has coedited six volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He is a Fellow of the IEEE, was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and is currently on the Editorial Boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He is an international keynote speaker and tutorial lecturer, as well as a consultant for government and industry.

Prof. Siegel's research interests include heterogeneous computing, parallel algorithms, interconnection networks, and the PASM reconfigurable parallel computer system. In the area of heterogeneous computing, he is examining ways to match segments of a task to different machines in a heterogeneous suite to exploit the varied computational capabilities available. His algorithm work explores the factors involved in mapping a problem onto a parallel processing system to minimize execution time. Topological properties and fault tolerance are the focus of his research on interconnection networks for large-scale parallel machines. He is analytically and experimentally investigating the utility of the three dimensions of dynamic reconfigurability supported by the PASM design ideas and the small-scale proof-of-concept prototype: mixed-mode parallelism, switchable inter-processor communications, and system partitionability. Agencies that have supported his research include the Air Force Office of Scientific Research, Army Research Office, Ballistic Missile Defense Agency, Defense Mapping Agency, IBM, NASA, Naval Ocean Systems Center, Naval Research Laboratory, National Science Foundation, NRaD, Office of Naval Research, and Rome Laboratory.

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 an associate professor in the School of Electrical and Computer Engineering at Purdue University. From September 1989 to August 1991 he held a Research Associateship position in the Department of Electrical Engineering at Stanford University. His research interests include parallel algorithms and architectures, computational paradigms for nanoelectronics, design and analysis of neural networks, special purpose computing arrays, VLSI design, and fault-tolerant computation.