Journal of Parallel and Distributed Computing **62**, 1338–1361 (2002)
doi:10.1006/jpdc.2002.1850

An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing Systems[1]

Muhammad K. Dhodhi[2]

Lucent Technologies, InterNetworking Systems, 1 Robbins Road, Westford, Massachusetts 01886
E-mail: dhohdi@lucent.com

Imtiaz Ahmad and Anwar Yatama
Department of Computer Engineering, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

and

Ishfaq Ahmad

Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, Texas

Received March 29, 2000; accepted January 29, 2002

This paper presents a problem-space genetic algorithm (PSGA)-based technique for efficient matching and scheduling of an application program that can be represented by a directed acyclic graph, onto a mixed-machine distributed heterogeneous computing (DHC) system. PSGA is an evolutionary technique that combines the search capability of genetic algorithms with a known fast problem-specific heuristic to provide the best-possible solution to a problem in an efficient manner as compared to other probabilistic techniques. The goal of the algorithm is to reduce the overall completion time through proper task matching, task scheduling, and inter-machine data transfer scheduling in an integrated fashion. The algorithm is based on a new evolutionary technique that embeds a known problem-specific fast heuristic into genetic algorithms (GAs). The algorithm is robust in the sense that it explores a large and complex solution space in smaller CPU time and uses less memory space as compared to traditional GAs. Consequently, the proposed technique schedules an application program with a comparable schedule length in a very short CPU time, as compared to GA-based heuristics. The paper includes a performance comparison showing the viability and effectiveness of the proposed technique through comparison with existing GA-based techniques. © 2002 Elsevier Science (USA)

[1]This research is funded by Kuwait University Grant EE-073.
[2]To whom correspondence should be addressed.

Key Words: distributed heterogeneous computing; heterogeneous processing; matching and scheduling; evolutionary computation; genetic algorithms; heuristics.

1. INTRODUCTION

A mixed-machine distributed heterogeneous computing (DHC) system generally consists of a heterogeneous suite of machines, high-speed networks, communication protocols, operating systems, and programming environments [8, 16, 24, 28, 30, 32]. DHC is emerging as a cost-effective solution to high-performance computing as opposed to expensive parallel machines. For effective utilization of a suite of diverse machines in a DHC system, an application program can be partitioned into a set of tasks (program segments) represented by an edge-weighted directed acyclic graph (DAG), such that each task is computationally homogeneous and can be assigned to the best-suited machine in the DHC system [24, 28, 30]. The matching and scheduling problem is the assignment of the tasks of a DAG to a suite of heterogeneous machines, sequencing the order of task execution for each machine such that precedence relationships between the tasks are not violated, and orchestrating inter-machine data transfers with the objective to minimize the total completion time [6, 7, 9, 15, 19, 22–25, 28–32, 35]. In general, the problem of task assignment and scheduling is known to be NP-complete [10]. Because of the intractable nature of the matching and scheduling problem, new efficient techniques are always desirable to obtain the best-possible solution within an acceptable CPU time.

Genetic algorithms (GAs) [11, 13, 17, 21], well known for their robustness, are probabilistic techniques that start from an initial population of randomly generated potential solutions to a problem, and gradually evolve towards better solutions through a repetitive application of genetic operators such as selection, crossover and mutation. The evolution process proceeds through generations by allowing selected members of the current population, chosen on the basis of some fitness criteria, to combine through a crossover operator to produce offspring thus forming a new population. The evolution process is repeated until certain criteria are met. GAs have been applied successfully to solve scheduling problems in a variety of fields, such as, job shop scheduling [4], sequence scheduling [33, 34], time-table problems [21], schedule optimization [27], task scheduling and allocation onto homogeneous multiprocessor systems [13, 17, 20], and task scheduling and matching in heterogeneous computing environments [23, 25, 30].

This paper proposes a technique based on a problem-space genetic algorithm (PSGA) [5, 26], which performs task matching, task scheduling, and inter-machine message scheduling in an integrated fashion. PSGA is an evolutionary technique that combines the search capability of genetic algorithms with a known fast problem-specific heuristic to provide the best-possible solution to a problem in an efficient manner as compared to other probabilistic techniques.

In a PSGA [5, 26], the chromosome is based on the problem data and all the genetic operators are applied in the problem space, so there is no need to modify genetic operators for each application. The solution is obtained by applying a simple and fast known heuristic to map from problem-space to solution-space, where each chromosome guides the heuristic to generate a different solution. Therefore, the search can be conducted much more efficiently.

PSGA uses a fast heuristic to map from problem-space to solution space, therefore it avoids disadvantages of other probabilistic approaches (such as local fine tuning in the last stage of standard GAs) and moreover, PSGA has a fast convergence rate as compared to the standard GAs [5]. PSGA-based techniques have been applied to solve many resource constrained combinatorial optimization problems such as datapath synthesis [5], static task assignment in homogeneous distributed computing systems [2] and static task scheduling onto homogeneous multiprocessors without considering the inter-processor communication cost [3].

The rest of the paper is organized as follows: Section 2 formulates the problem, and Section 3 presents the related work. Section 4 explains the problem-space genetic algorithm-based matching basic technique and the scheduling and assignment algorithm. Section 5 provides the experimental results, and finally Section 6 concludes the paper with some final remarks and summarizing comments.

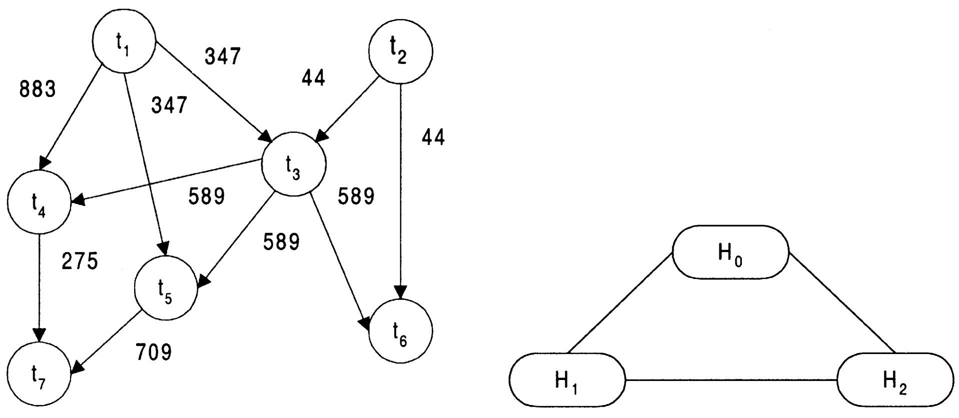
2. PROBLEM FORMULATION

In a DHC system, an application program is partitioned into a set of tasks modeled by a DAG and can be represented as $G = (T, <, E)$, where $T = \{t_i,\ i = 1, \ldots, n\}$ is a set of n tasks. $<$ represents a partial order on T. For any two tasks $t_i, t_k \in T$, the existence of the partial order $t_i < t_k$ means that task t_k cannot be scheduled until task t_i has been completed, hence t_i is a predecessor of t_k and t_k is a successor of t_i. E is the set of directed edges or arcs. A weight $D_{i,k}$ is associated with each arc that represents the amount of data to be transferred from task t_i to task t_k in bytes.

A mixed-machine distributed heterogeneous computing system consists of a set $H = \{H_j : j = 0, \ldots, m-1\}$ of m independent different types of machines (including sequential and parallel computers) interconnected by a high-speed arbitrary network. The bandwidth (data transfer rate) of the links between different machines in a DHC system may be different depending on the kind of the network. The data transfer rate is represented by an $m \times m$ matrix, $R_{m \times m}$. The estimated computation time (ECT) of a task t_i on a machine H_j is denoted as ECT_{ij}, where $0 \leqslant i < n$ and $0 \leqslant j < m$. The ECT value of a task may be different on different machines depending on the machine's computational capability. For static task scheduling, the ECT value for each task–machine pair is assumed to be available a priori. An example of a DAG consisting of seven tasks adopted from [30] is shown in Fig. 1(a) and a DHC system consisting of fully connected three heterogeneous machines is shown in Fig. 1(b). We assume that the data transfer rate for each link is 1.0, hence the arc weight and the data transfer rate will be the same. The ECT value of each task on different machines (H_0–H_2) for this example is given in Table 1.

Furthermore, we make the following assumptions:

- Each machine in the heterogeneous system can perform communication and computation simultaneously.



(a) A task graph (DAG). (b) A mixed-machine DHC system.

FIG. 1. An example DAG and a mixed-machine DHC system.

- Task execution can start only after all the data have been received from its predecessors tasks.
- All machines and inter-machine networks are available for exclusive use of the application program.
- Communication cost is zero when two tasks t_i, t_k are assigned to the same machine, otherwise data have to be transferred from the machine on which task t_i is assigned to the machine where task t_k is assigned. This data transfer incurs the communication cost (CommCost) given by

$$CommCost(t_i, t_k) = \frac{D_{i,k}}{R[H(i), H(k)]}, \tag{1}$$

where $D_{i,k}$ is equal to the amount of data to be transferred from task t_i to t_k and $R[H(i), H(k)]$ represents bandwidth (data transfer rate) of the link between the machines onto which tasks t_i, and t_k have been assigned.

The problem of static task matching and scheduling in a distributed heterogeneous computing environment is a mapping $\pi : T \rightarrow H$ that assigns the set of tasks T onto a set of heterogeneous machines H, determines the start and the finish times of each

TABLE 1

The ECT Values of the Tasks for the System of Fig. 1

Task	H_0	H_1	H_2
t_1	872	898	708
t_2	251	624	778
t_3	542	786	23
t_4	40	737	258
t_5	742	247	535
t_6	970	749	776
t_7	457	451	15

task, and determines the start and the finish times of each inter-machine data transfer so that precedence constraints are maintained and the schedule length (*SL*) that is, the overall program completion time, given by Eq. (2) is minimized.

$$SL = \max\{F_0, F_1, \ldots, F_{m-1}\}, \tag{2}$$

where F_j is the overall finish time of machine H_j, $j = 0, \ldots, m-1$. The finish time includes the computation time, the communication time and the waiting time because of the precedence constraints. This is an intractable combinatorial optimization problem with conflicting constraints.

3. RELATED WORK

Task matching and scheduling techniques [6, 7, 9, 15, 19, 22–32, 35] can be categorized into optimal selection theory-based approaches [9, 31], graph-based approaches [6, 22], genetic algorithm-based techniques [23, 25, 30] and other heuristics [6, 7, 19, 29, 35]. Most of the work has been carried out to find a near-optimal solution. We describe a levelized min-time (LMT) scheduling heuristic presented in [15] and three evolutionary techniques based on genetic algorithms that have been developed by Wang *et al.* [30], Singh *et al.* [25] and Shroff *et al.* [23].

Iverson *et al.* [15], proposed a two-phase approach called LMT heuristic to account for both precedence constraints and variable execution times of each task on different machines. During the first phase, the so-called level sorting is used to obtain non-precedence constrained sub-tasks in each level. In the second stage, an algorithm called Min-Time is applied for each level. In this stage for each task in a level a search is carried out for a processor that does not have a task assigned to it and the summation of task's execution time and the transfer time taken by all the required data for this task is minimum. The task is then assigned to that processor.

In [30], a GA-based approach is used to solve the optimization problem under discussion. A pre-defined number of chromosomes were generated for the initial population, which also includes a chromosome obtained by using the LMT heuristic [15]. Each chromosome is represented by the two-tuple $\langle mat, ss \rangle$, where *mat* is the matching string and *ss* is the scheduling string. When generating a chromosome, a new matching string is obtained by assigning each task to a machine randomly. While for the scheduling string the SPDAG is first topologically sorted to form a basic scheduling string, which later on is mutated a random number of times to generate the *ss* vector. The standard GA approach as described in [11] is then used for obtaining next generations. For small-scale problems, multiple optimal solutions obtained in reasonable CPU time, but the CPU times taken by the large problems are extremely large.

In [25], a search for a better set of parameters for the GA-based algorithm is carried out. Each parameter is varied over some range, keeping other parameters constant. The objective is to come up with such a parameter set for which the fitness of the chromosome improves. Fitness of the chromosome in this case is defined as the completion time of the last task. It is observed that window and linear scaling, tournament selection method, uniform crossover type enabled niching and elitism and high crossover probability values are better than their counterparts.

In [23], a genetic simulated annealing (GSA) technique is used for task scheduling in heterogeneous environments. GSA is a hybrid algorithm that is a combination of GAs and simulated annealing (SA). In this technique, all the standard GA operators such as generation of initial population, evaluation of the fitness, reproduction via crossover and mutation, etc., were used except the selection operator for selecting the mates for reproduction. For the selection operator, a SA approach was applied. The fitness function is the reciprocal of the task completion time. If the child's fitness is better than its parent, it is selected. If the fitness is smaller than its parent there is still a chance for its selection as a candidate for reproduction, depending upon a probability value that in turn is a function of current temperature. As current temperature decreases over time according to the schedule, the probability of accepting less fit solutions also decreases, which helps the algorithm to converge. For small-scale problems, optimal solutions are obtained. However, the CPU times taken by large problems, like traditional GAs, are too large.

4. MATCHING AND SCHEDULING TECHNIQUE

Combinatorial optimization techniques, based on analogy to certain principles of natural phenomena, have been successfully used in a variety of applications [12]. Evolutionary computation techniques or simply evolutionary techniques include: GAs [11, 13], genetic programming (GP), evolution strategies (ES), simulated evolution (SE) and evolutionary programming (EP). In general, the evolutionary techniques roughly simulate the "theory of evolution" proposed by Darwin in the 19th century [12].

The PSGA [5, 26] is an evolutionary technique that is an embodiment of a genetic algorithm and a problem-specific known fast heuristic. An outline of the proposed algorithm is given in Fig. 2.

Briefly, the proposed technique can be divided into two phases. The first phase (Steps 1–3) is an initialization phase. In this phase, first the DAG, ECT matrix, and data transfer rate matrix are read. The user also provides the population size (N_p), number of generations (N_g), probability of crossover (X_r), and probability of mutation (M_r). The static *b-level* (bottom level) and static *t-level* (top level) of each node in the DAG are calculated (see Section 4.1). Then, this phase involves the coding of the chromosome based on static *b-levels*, formation of an initial population by perturbing the gene values (priorities) in the first chromosome, application of the decoding heuristic to obtain solutions (schedules), and application of the *evaluate* function to calculate the objective value and the fitness of each chromosome in the current population.

In the second phase, an iteration is carried out for function *select* for the selection of parent chromosomes for reproduction, the *crossover* function and *mutate* function to obtain offspring from the selected parents for the new generation, assignment of the new population as the current population, application of decoding heuristic to obtain solutions (schedules), and the application of the *evaluate* function to calculate

```
Step 1. Read the DAG, associated ECT matrix, and the data transfer rate matrix $R$.
        Build a database (that includes the adjacency list) and the parent list.
Step 2. Get $N_p$ , $N_g$, $X_r$ and $M_r$ from the user.
Step 3. Calculate the b-level and the t-level of each task in the DAG;
        Generate Initial_Population ($P_{initial}$);
        $P_{current}$ ← $P_{initial}$;   /* copy initial population to current population */
        Schedules ← Decoding_heuristic ($P_{current}$);
        Best_Schedule ← evaluate ( Schedules );

Step 4. while stopping criterion not satisfied do begin
            $P_{new}$ ← { }; /* empty new population */
            repeat for $Np/2$ times
                dad ← select ($P_{current}$ , Sum_of_fitness);
                mom ← select ($P_{current}$, Sum_of_fitness);
                $P_{new}$ ← $P_{new}$ U crossover (dad, mom, child1, child2, $X_r$);
            endrepeat;
            for each chromosome ε $P_{new}$ do begin
                mutate (chromosome, $M_r$);
            endfor;
            $P_{current}$ ← $P_{new}$;
            Schedules ← Decoding_heuristic ($P_{current}$);
            Best_Schedule ← evaluate (Schedules );
        endwhile;
Step 5. Report the best schedule.
```

FIG. 2. The PSGA based task matching and scheduling algorithm.

the fitness of each chromosome's solution. The iteration terminates when the stopping criterion is met (that is, a fixed number of generations have been evaluated).

The above procedure is elaborated with the help of an illustrated example (shown in Fig. 1) in the following subsections.

4.1. Chromosomal Representation and Initial Population

One of the key differences between standard GAs and PSGA lies in the encoding of chromosomes. A chromosome in traditional GAs [11] is often a string that represents a valid solution to the problem to be solved. On the other hand, a chromosome in the PSGA represents some attribute of the problem data. This information is used by a problem-specific decoding heuristic to generate a solution to the problem. Thus, the PSGA obtains a particular solution by applying the decoding heuristic on a chromosome [26]. Each position of a chromosome is called a *gene*. In the case of the matching and scheduling problem, a *gene* i in the chromosome represents the priority of corresponding task t_i. For the first chromosome, the priority of a task (node) t_i is its *b-level*. The *b-level* of a task (node) t_i is the length of the longest path from node t_i to an exit node. The *b-level* of a node is bounded by the critical path of the DAG. The *b-level* is calculated by summation of two parameters encountered on the path, that are each task's average ECT value (*AvgECT*) taken over all machines and the communication cost (*CommCost*). A procedure for calculating the *b-levels* is shown below:

```
Procedure b-levels( ):
  Construct a reverse topological order list (RevTopOrdList) of tasks.
  for each task $t_i$ in the RevTopOrdList do begin
    $max = 0$;
    for each child task $t_j$ of task $t_i$ do begin
      if ($CommCost(t_i, t_j) + b\text{-}level(t_j) > max$) then
        $max = CommCost(t_i, t_j) + b\text{-}level\ (t_j)$;
      endif;
    endfor;
    $b\text{-}level\ (t_i) = AvgECT(t_i) + max$;
  endfor;
```

If a task has no children its *b-level* is equal to its *AvgECT*.

Another parameter called *t-level* for each task (node) is also calculated. The *t-level* of a task t_i is the length of the longest path between this node and an entry node in the DAG excluding the ECT of t_i. This level essentially determines the earliest start time of a node. A task that has no parent will have *t-level*= 0. The procedure for calculating the *t*-levels is given below:

```
Procedure t-levels( ):
  Construct a topological order list (TopOrdList) of tasks.
  for each task $t_i$ in the TopOrdList do begin
    $max = 0$;
    for each parent task $t_k$ of task $t_i$ do begin
      if $t\text{-}level\ (t_k) + AvgECT(t_k) + CommCost(t_k, t_i) > max$ then
        $max = t\text{-}level\ (t_k) + AvgECT(t_k) + CommCost(t_k, t_i)$;
      endif;
    endfor;
    $t\text{-}level\ (t_i) = max$;
  endfor;
```

The average ECT value and the *b*-, *t-levels* (using the above procedures **b-levels( )** and **t-levels( )**) for each task of the example DAG shown in Fig. 1 are given in Table 2. The *AvgECT* value of a task represents the average of the ECT values of a task over all the three machines shown in Fig. 1. Note that the *b-levels* of tasks t_6 and t_7 are the same as the *AvgECT* values because these tasks have no children. The *t-levels* of tasks t_1 and t_2 are 0 because these tasks have no parent (predecessors).

As described before, the gene value (that is, priority) of a task $t_i, (GT_1)^i$ in the first chromosome is set to the *b-level* of a task t_i. The gene values (priorities) of the rest of the chromosomes (that is, chromosome 2 to N_p) in the initial population are generated by a random perturbation of the gene values of the first chromosome as given below:

$$(GT_j)^i = (GT_1)^i + Uniform(t\text{-}level/2, -t\text{-}level/2), \tag{3}$$

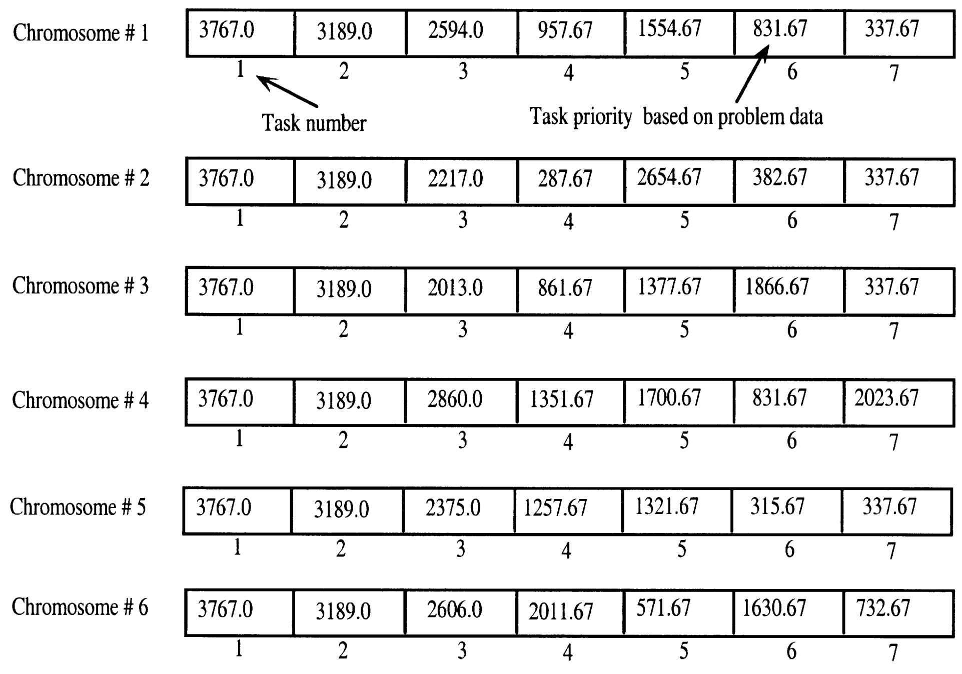
TABLE 2

The *AvgECT b-level* and *t-level* of Each Task for the DAG Shown in Fig. 1

Task	*AvgECT*	*b-level*	*t-level*
t_1	826.0	3767.0	0.0
t_2	551.0	3189.0	0.0
t_3	450.33	2594.0	1173.0
t_4	344.67	957.33	2212.33
t_5	508.0	1554.67	2212.33
t_6	831.67	831.67	2212.33
t_7	337.67	337.67	3429.33

where $(GT_l)^i$ is the priority of a task t_i in the first chromosome, $i = 1, \ldots, n$. *Uniform* (t-*level*$/2, -t$-*level*$/2$) is a random number generated uniformly between t-*level*$/2$, *and* $-(t$-*level*$/2)$, and $j = 2, \ldots, N_{\mathrm{p}}$.

Using the above rules, an initial population of size six for the DAG of Fig. 1(a) is shown in Fig. 3. Here, each chromosome has a different priority value for each task in different chromosomes. So each chromosome guides the heuristic to generate a different solution to the problem, which in this case will be a different schedule. Note that since the *t-levels* of tasks t_1 and t_2 for the example DAG are 0, there is no perturbation in the priorities of these tasks. Thus, the priorities (gene values) of these two tasks for the example DAG remain the same for all the chromosomes in the initial population.



	1	2	3	4	5	6	7
Chromosome # 1	3767.0	3189.0	2594.0	957.67	1554.67	831.67	337.67
Chromosome # 2	3767.0	3189.0	2217.0	287.67	2654.67	382.67	337.67
Chromosome # 3	3767.0	3189.0	2013.0	861.67	1377.67	1866.67	337.67
Chromosome # 4	3767.0	3189.0	2860.0	1351.67	1700.67	831.67	2023.67
Chromosome # 5	3767.0	3189.0	2375.0	1257.67	1321.67	315.67	337.67
Chromosome # 6	3767.0	3189.0	2606.0	2011.67	571.67	1630.67	732.67

FIG. 3. An initial population of six chromosomes.

4.2. Decoding Heuristic

One of the key factors in the success of the proposed technique is the embedding of a known problem-specific fast heuristic into the GA search. We use list scheduling [1, 14, 18] with two different policies, earliest finish time and earliest start time, to design two different decoding heuristics.

4.2.1. Earliest finish time heuristic. The earliest finish time (*EFT*) heuristic schedules a candidate task onto a machine on which the finish time of the task is the earliest. The objective is to generate a matching and schedule from a given chromosome having a minimum schedule length. Pseudocode for the decoding heuristic is given in Fig. 4. In this heuristic, we build a *task list*, in which the tasks are arranged in a descending order according to their priorities in the chromosome. If there are two tasks with the same priority, then the task with smaller number is placed first. A *ready list* is then initialized with tasks that have no predecessor. The task t_i from the ready list with the highest priority is selected, and scheduled to the most suitable machine H_j on which the finish time of the task is the earliest. Machine H_j is selected to be a candidate for matching the task t_i, and determines the start time of this task onto machine H_j by taking into account data available time and machine ready time. The data available time is the maximum communication time when the data of all the immediate predecessors of the t_i are available at machine H_j. Then the algorithm determines the finish time of task t_i on the candidate machine by summing the start time and the ECT of the task–machine pair. In case the finish time of a task on two machines is the same, the algorithm breaks the tie by matching and scheduling the task to the first machine in the sequence. Task t_i is then deleted from the *ready list*. The heuristic repeats the matching and scheduling with updates to the *ready list*, until the list becomes empty and all the tasks are scheduled.

By applying the *EFT* heuristic to each chromosome in the population, we can generate a different solution (schedule) corresponding to each chromosome. The solution generated for the first chromosome is shown in Fig. 5.

```
Heuristic EFT (chromosome, $m$, $n$):
    Build a task_list from the chromosome;
    ready_list ← Initialize_ready_list(task_list);
    while (not Done) do begin
        $t_i$ ← task with the highest priority value from the ready_list;
        $H_j$ ← machine_with_earliest_finish_time($t_i$);
        Schedule_message($t_i$, $H_j$);
        Schedule_task ($t_i$, $H_j$) ;
        Delete task $t_i$ from the ready_list ;
        ready_list ← update_ready_list (task_list);
    end while;
end EFT.
```

FIG. 4. Pseudocode for EFT heuristic.

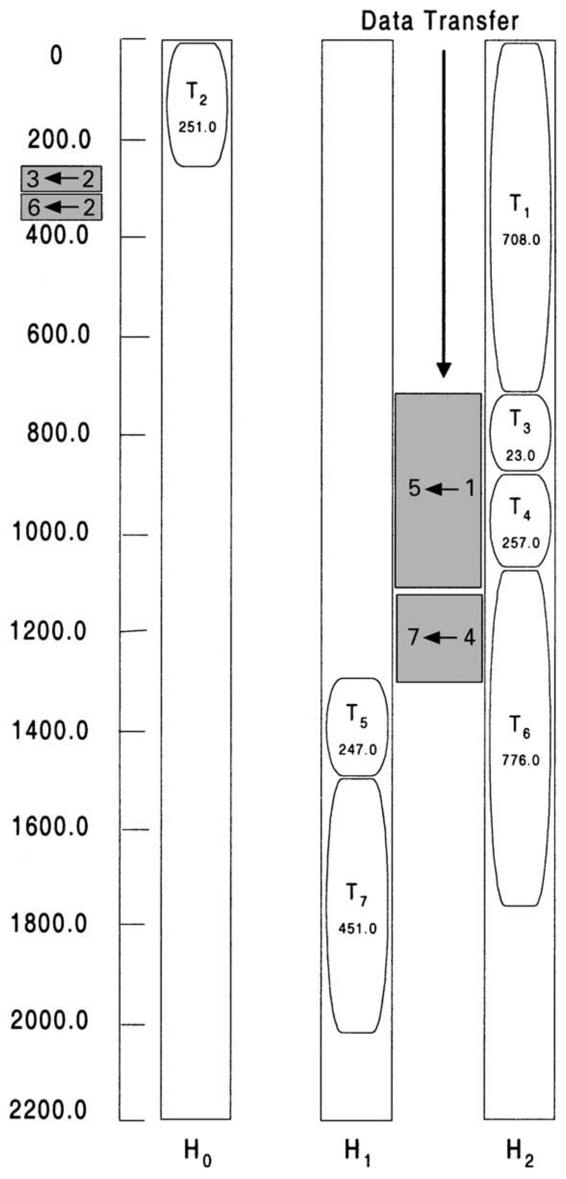


FIG. 5. Matching and schedule corresponding to the first chromosome ($SL = 2069$).

4.2.2. Earliest start time heuristic. We also embedded the earliest start time (*EST*) heuristic into our PSGA scheduler. The difference between this heuristic and the *EFT* heuristic described above is in the matching and scheduling policy. In this heuristic, a task t_i is assigned and scheduled onto a machine H_j for which the task's start time is the earliest. This heuristic works well for task assignment and scheduling in homogeneous multiprocessor systems. However, simulation results (see Section 5) show that this heuristic performs poorly as compared to the *EFT* heuristic in heterogeneous computing environment. The reason behind this relatively poor performance is that it cannot take into account the fact that in DHC the *ECT* values of a task are different on different machines.

4.3. Objective and Fitness Evaluation

We use the schedule length as the objective function for evaluating the schedules corresponding to each chromosome. However, its value may vary from problem to problem. To maintain uniformity over various problem domains, we use the fitness to normalize the objective function values to a convenient range of 0 to 1. The following objective-to-fitness mapping function [11] was used:

$$f(i) = \frac{SL(M) - SL(i)}{\sum_{j=1}^{j=N_p} [SL(M) - SL(j)]}, \tag{4}$$

where $f(i)$ is the fitness of chromosome i, $SL(M)$ is the maximum schedule length of a solution corresponding to a chromosome in the current population, $SL(i)$ is the schedule length corresponding to chromosome i, and N_p is the population size.

4.4. Selection, Crossover, and Mutation

PSGA combines the exploitation of the past results by selecting parent chromosomes for reproduction based on their fitness with the exploration of new areas in the search space via crossover and mutation. Chromosomes with higher fitness values have a higher probability of contributing one or more offspring in the next generation. This method is a simulated version of the natural selection, a Darwinian survival of the fittest theory [11]. A biased roulette wheel, where each chromosome in the population has a slot sized in proportion to its fitness, performs the selection method. Each time we require an offspring, a simple spin of the weighted roulette wheel yields a parent chromosome. In this method, a random number u is generated between 0 and the sum of the fitness. A population member j whose running sum of the fitness is greater than or equal to u is selected as a candidate for reproduction. GA-based techniques use many other selection methods such as tournament selection, simulated-annealing selection, etc., but a simple roulette-wheel selection method works well in PSGA.

In natural genetics, at the cellular level, a pair of chromosomes strike into one another, exchange chunks of genetic information and drift apart. In GAs this is generally referred to as crossover because of the way that genetic material crosses over from one chromosome to another. Crossover incorporates attributes of two parents into a new individual. We apply a 2-point crossover operator to the priority of the chromosome. Since a chromosome is based on the problem data and has to be decoded using an underlying heuristic, it always generates a feasible solution. The crossover is applied with a certain crossover rate (X_r) which is the ratio of the number of offspring produced by crossover in each generation to the population size. It controls the amount of crossover being applied. In a 2-point crossover operator, two cross sites are selected randomly and the values of the priorities between the cross sites are swapped among the two mating chromosomes.

Mutation is a background operator that is used for finding new points in the search space so that population diversity can be maintained. Mutation is done by selecting a gene at random with a probability M_r and perturbing its value in the range $-(t\text{-}level/2)$ to $t\text{-}level/2$, where *t-level* is the *t-level* of the selected node in the DAG. After perturbation if the priority value becomes more than $b\text{-}level + t\text{-}level$ of the node, then the priority is assigned the value $b\text{-}level + t\text{-}level$. If the priority value becomes less than *b-level*, it is assigned the value *b-level*. The objective is to explore a wider space of priorities, but within the proximity of the original problem.

4.5. Termination Criteria

The algorithm maintains the best solution found so far in the database since elitism is incorporated. The algorithm terminates when N_g generations are completed. We use a fixed number of generations as the stopping criterion because the proposed technique incorporates a problem-specific fast heuristic and it can search the design space in a small number of evaluations ($N_p N_g$). The result obtained by applying the heuristic to the first chromosome is fairly good. By applying this heuristic to a new set of problem data differing slightly from the original problem for a fixed number of times results in the best solution. The value of N_g is determined experimentally by running a large number of simulation tests.

Figure 6 depicts the best matching and the corresponding schedule obtained by the proposed technique for the example task graph (Fig. 1). The best schedule length obtained by the proposed PSGA was 2018 as compared to the schedule length ($SL = 2069$) obtained from the first chromosome of the first generation.

5. EXPERIMENTAL RESULTS

We implemented the proposed PSGA-based integrated technique for static task matching and scheduling onto a mixed-machine DHC system on a SUN SPARCstation 20. There are no common benchmark application programs for heterogeneous computing environments. Therefore, in order to test our system and compare its results with the published work, we requested the researchers who have published GA-based schedulers for heterogeneous computing environments [23, 30] for test graphs. Shroff *et al.* [23] provided us the source code written for their GSA scheduler. Their scheduler also includes a random test graph generation allowing the user to generate a variety of test graphs, with different numbers of nodes, their associated ECTs, and data transfer rate matrices. Wang *et al.* [30] also provided a task-graph generation program that has been used for their GA-based matching and scheduling technique. Performance of the proposed technique is compared with Wang *et al.* [30], GSA [23], and LMT [15] by running a number of simulation tests.

Simulation tests are organized in five major test suites. In test suite 1, we evaluate the performance of the proposed technique with a set of directed acyclic graph structures of well-known applications and with a set of randomly generated graphs. Test suite 2 includes small-scale performance simulation tests to compare with the work of in [30]. Test suites 3 and 4 include large-scale simulation tests to compare with GSA [23] and LMT [15], respectively. Test suite 5 includes simulation tests

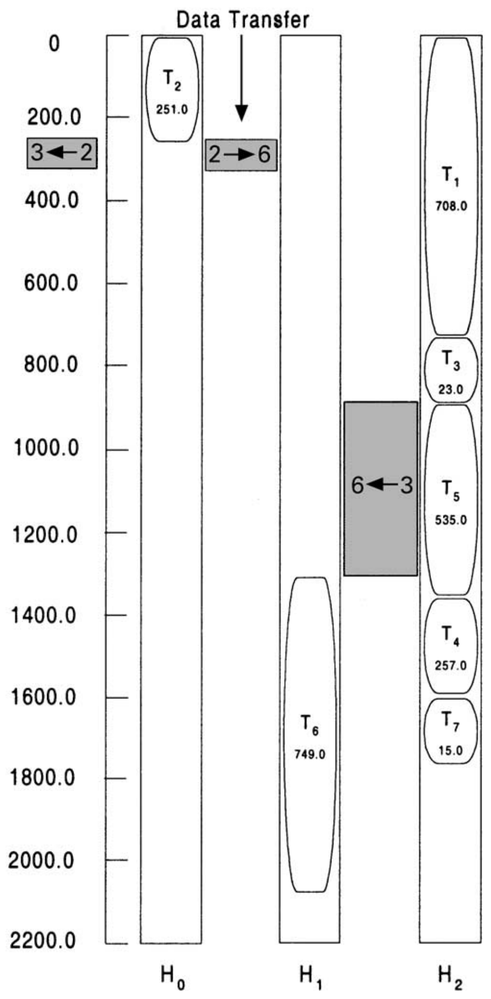


FIG. 6. The best assignment and schedule for the example DAG ($SL = 2018$).

carried out to demonstrate the effect of embedding different heuristics in PSGA. For all the test suites, the results obtained by the PSGA used fixed values of the input parameters ($X_r = 0.6$, $M_r = 0.05$, $N_p = 60$, and $N_g = 100$). Moreover, for each test the PSGA was run 10 times. Details of each test suite are given below:

Test suite 1: In this test suite, we evaluate the performance of the proposed PSGA-based technique for different types of DAGs. First, a set of graphs representing practical applications such as Out-Tree (OT), Fork-Join (FJ), Laplace equation solver (LAPLACE), and LU decomposition were used as input to the PSGA. The ECT values for these graphs were generated randomly for DHC systems consisting of four and eight machines. The performance of the proposed techniques with respect to the quality of schedule length due to the first chromosome (SL_1) and the best SL when the termination criterion is satisfied is given in Table 3 (here n is the number of tasks and m is the number of machines). There was 7.95% average

TABLE 3

Comparison of SL_1 (Schedule Length When EFT Heuristic Is Applied to the First Chromosome) and the Best SL for the Proposed Technique for a Set of Test Graphs Representing Practical Applications onto DHC Systems Consisting of Four and Eight Machines

DAG structure	n	m	SL_1	SL	CPU time (s)
OT	50	4	455	405	22
	100	8	562	501	80
FJ	50	4	1276	1175	29
	100	8	2251	2123	109
LAPLACE	49	4	2957	2787	29
	100	4	6892	6313	95
	100	8	5991	5678	110
LU	35	4	810	759	7
	35	8	769	656	10
	54	8	2177	1962	41

improvement in the *SL* over SL_1 for all of these tests. The CPU time to run these simulations on SPARCStation 20 was in the range of 7–110 s.

The second set of graphs (in the range of 20–200 tasks and for DHC systems consisting of up 20 heterogeneous machines) generated by using the graph generator in [30]. Using this set, we evaluate the performance of the proposed PSGA-based technique from three different perspectives. First, we show the performance with respect to quality of schedule length due to the first chromosome (SL_1) and the best *SL* when the termination criterion is satisfied. For this purpose, we have run 11 tests ranging from DAGs consisting of 7–200 tasks and the DHC systems consisting of 3–20 machines. The results of these tests are shown in Table 4. There was 21.2% average improvement in the *SL* over SL_1 for all of these tests. The CPU time to run these simulations on SPARCStation 20 was in the range of 2–789 s.

The amount of improvement over the schedule length can be obtained due to first chromosome, actually depends on the structure of the tasks graphs and ECT values. The PSGA approach is based on an entirely different neighborhood structure and presumes that the base heuristic yields reasonably good solutions to the problem at hand. If this heuristic is applied to a new set of problem data differing only slightly from the original problem, the resultant solution should also be a reasonably good solution to the original problem. Thus, if one applies the base heuristic to the problem data in the neighborhood of the original, one expects to generate a set of good solutions. Since good solutions tend to be clustered near the original problem, the search can be more effective and the best-possible solution could be found with smaller population size and generation size as compared to the standard GAs [11].

Next, we investigate the effect on the CPU time spent by the proposed scheduler by fixing the DAG size and by varying the number of machines. The results of this experiment for a DAG of 60 and 100 tasks on 2-, 4-, ..., 20-machine DHC systems are plotted in Fig. 7. It is evident that the CPU time varies almost linearly with the increase in the number of machines. In the third phase of this experiment, we assume

TABLE 4

Comparison of SL_1 (Schedule Length When EFT Heuristic Is Applied to the First Chromosome) and the Best SL for the Proposed Technique for a Set of Test Graphs onto DHC Systems Consisting of Different Numbers of Machines

n	m	SL_1	SL	CPU time (s)
7	3	1025	589	2
10	3	2203	1825	3
16	6	1936	1476	6
40	20	2971	2406	48
60	8	3999	3257	58
80	8	3975	3110	85
100	8	4478	3806	126
120	20	4048	3404	302
140	20	5060	3631	411
160	20	4783	4041	516
200	20	6269	4561	789

a DHC system consisting of a fixed number of machines and investigate the change in CPU time by increasing the size of DAG. We carried out one test for an 8-machine DHC system and another for a 16-machine DHC system. The results showing the CPU time versus the number of tasks in the DAG are plotted in Fig. 8. Note the CPU time varies polynomially with the increase in the number of tasks.

Test suite 2: This test suite consists of three small-scale performance simulation tests. We generated a 7-node task graph for the first test and a 10-node task for the second and third test using Wang's graph generator program [30]. Simulations were carried out on a 3-machine DHC system. Although no reported schedule lengths for these task graphs are available, the CPU time reported by Wang *et al.* [30] for a similar small-scale test, consisting of 10 tasks onto a 3-machine DHC system, was in the range of 60 s on a Sun SPARCstation 5. The PSGA however, completes the execution in less than 2.5 s.

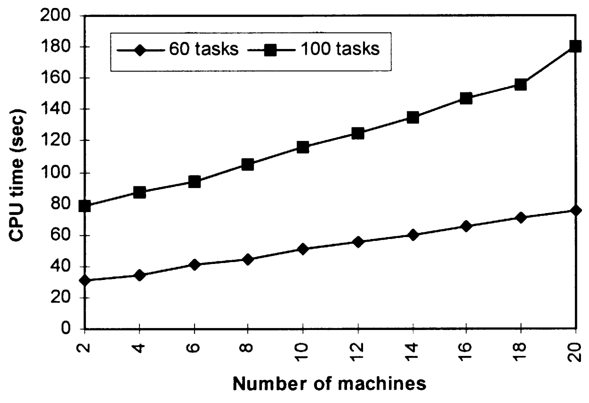


FIG. 7. The CPU time versus the number of machines for fixed size DAGs.

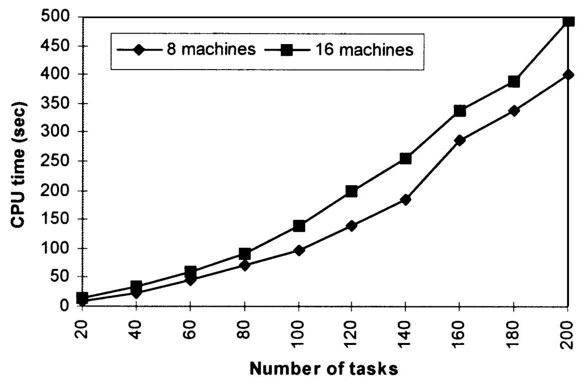


FIG. 8. The CPU time versus the number of tasks for fixed number of machines.

Test suite 3: For this suite, a large set of test graphs, their associated ECTs, and rate matrices were generated by using the DAG-generator portion of the GSA scheduler [23]. For both PSGA and GSA, values of 0.6 and 0.05 are used for X_r and M_r, respectively. However, N_p is selected as 60 and 256 for PSGA and GSA, respectively. The number of generations (N_g) is kept constant at a value of 100 for PSGA, while for GSA it starts from 150 and is increased as the problem size increases (as shown in Table 5). The reason is that the GSA uses blind search, and, therefore, needs a larger population size and number of generations to converge with a good solution quality. Different graphs were generated by varying the computation-to-communication ratio (CCR), that can be defined as the ratio of the time needed to execute the task to the time needed to transfer the data. CCR is considered as small, medium and large when $0.012 < CCR < 0.5, 1 < CCR < 50$ and $50 < CCR < 350$, respectively. Both the proposed technique and GSA were run on a SPARCstation 20 for all of these test graphs. Table 5 shows the comparison of results.

The CPU time required to solve a *DAG* of 30 tasks on 12 machines by the GSA [23] was 81858 s, whereas the CPU time taken by PSGA was 47 s.

We have computed the average of schedule lengths obtained by PSGA and GSA, for all tests shown in Table 5. By considering PSGA's average schedule length as unity, we calculated the relative solution quality of GSA by dividing the average schedule length of PSGA with the average schedule length of GSA. Similarly, we have calculated the average CPU time spent by PSGA and GSA for all the test graphs and then obtained the relative CPU time. The comparison of results is shown in Table 6 where it is clear that the quality of results (schedule lengths) generated by PSGA is comparable to that of GSA; on the average, PSGA is 1366 times faster than the GSA. The reason for this large difference in CPU times is because of two factors: (1) GSA has used simulated annealing as a selection operator that slows down it further than standard GAs [11], and (2) PSGA embeds a problem-specific heuristic in the genetic algorithm, and, therefore, it converges in a smaller number of generations and with smaller population size. The CPU time spent by the GSA [23] is extremely large, and therefore, it does not perform well for large *DAGs*.

TABLE 5

Comparison of Results for GSA and PSGA for Test Suite 3

			PSGA		GSA [23]		
CCR	*n*	*m*	*SL*	CPU time (s)	*SL*	CPU time (s)	N_g
Medium	12	5	193.1	4	193.1	7420	150
	15	3	285.2	7	285.2	7733	150
	15	5	199	8	199	7080	150
	16	6	189.5	11	189.5	23,501	300
	18	7	122.1	12	120.6	19,021	300
	20	6	250	15	250	26,364	200
	20	8	270.4	17	270.4	24,079	200
	22	8	241.1	20	240.8	38,518	300
	24	11	243.7	27	242.2	33,242	300
	25	8	316.4	25	314.7	46,548	250
	25	10	281.2	32	278.9	33,669	250
	26	13	290	37	290	62,233	300
	28	16	241.8	48	240	68,901	300
	30	9	326.7	37	326.7	55,411	300
	30	12	271.4	48	268.1	80,612	300
Large	15	5	1622	8	1622	6793	150
	20	8	2904.1	16	2904.1	21,690	200
	25	10	2996.9	31	2996.9	32,200	250
	30	12	3212.5	47	3212.5	81,858	300
Small	20	8	1141.6	22	1087.9	24,392	200
	25	10	1521.2	36	1625.2	45,818	250
	30	12	1695.2	71	1738.1	60,752	300
Unity	13	6	112.4	7	112.9	5480	150
	15	5	127.9	8	114.4	6904	150
	20	8	185.4	19	180.4	15,922	200
	25	10	213.5	29	207.6	40,794	250

Test suite 4: In this test suite, Wang's graph generator program [30] is used to generate large task graphs of size up to 200 nodes and for DHC systems consisting of up to 20 heterogeneous machines. For simulations, three different DHC systems consisting of four, eight, and 20 machines were selected. Three task graphs of the

TABLE 6

Average Performance Measures for All Test Graphs in Suite 3, When Normalized with Respect to PSGA

	PSGA	GSA
Relative solution quality	1	0.997
Relative CPU time	1	1366

TABLE 7

Comparison of SL Obtained by PSGA and LMT for DAGs Ranging from 20 to 200 Tasks and on the DHC System Consisting of Four Machines

	Light comm.		Medium comm.		Heavy comm.	
n	LMT *SL*	PSGA *SL*	LMT *SL*	PSGA *SL*	LMT *SL*	PSGA *SL*
20	2545	958	3306	1997	2741	1489
40	5494	2308	8260	3419	7062	2652
60	8289	2980	10,405	4002	9136	4031
80	11,163	3842	16,667	5423	12,282	4982
100	15,525	5047	16,257	6383	19,696	7571
120	16,032	6641	23,093	9293	22,066	8406
140	20,521	8102	23,733	9972	25,121	9769
160	25,401	9858	25,638	10,957	29,441	10,380
180	27,972	11,596	31,987	11,713	32,072	11,707
200	27,793	12,477	35,887	12,580	37,545	14,574

same size but with different numbers of edges were generated. Such task graphs are termed as light-communicating (light comm), medium-communicating (medium comm) and heavy-communicating (heavy comm) task graphs, when $0<|E|<\frac{1}{3}n$, $\frac{1}{3}n<|E|<\frac{2}{3}n$, and $\frac{2}{3}n<|E|<n$, respectively; where $|E|$ is the number of edges and n is the number of nodes of a task graph. For each of the three DHC systems (four, eight, and 20 machines), task graphs of different sizes were generated. For each size (e.g., 20 nodes), three task graphs with different numbers of edges were generated. Results were then obtained for both PSGA and LMT [15], Using these graphs as input.

Table 7 provides the comparison of schedule length (SL) for all of these graphs obtained by PSGA and LMT for a 4-machine DHC system. The results indicate that the solution quality in terms of SL obtained by LMT is poor in comparison with the SL obtained by the PSGA. For example, for the DAG consisting of 200 tasks with light communication, the SL obtained by LMT is 27793, while the SL obtained by PSGA is 12,477. Similarly, the results for other DAGs show a large difference between the solution obtained by the LMT and the PSGA.

To make the comparison easy, the solution quality of LMT is assumed to be 1.0, and the relative solution quality of PSGA, as compared to LMT, is calculated by dividing the schedule length generated by LMT with the schedule length generated by the PSGA. Figure 9 shows the comparison of the relative solution quality for the 4-machine DHC system. Each group of four columns in Fig. 9 corresponds to the results obtained for three graphs with different numbers of edges, therefore, different communication patterns, for a particular sized graph (fixed number of nodes). As relative solution quality (schedule length) of LMT is always taken as unity, only one column for all three task graphs is shown for LMT.

Similarly, the comparisons of relative solution qualities for 8- and 20-machine DHC systems, by using task graphs of different sizes and various communication

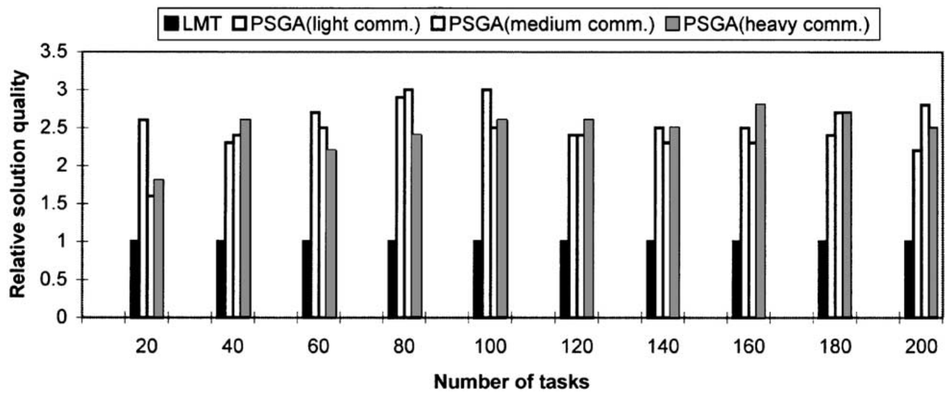


FIG. 9. Comparison of relative solution quality of PSGA and LMT for a set of test graphs, each with three different communication patterns (i.e., light, medium and heavy communication) for a 4-machine DHC system.

patterns, are shown in Figs. 10 and 11, respectively. We observe that the CPU time taken by the LMT heuristic is in the range of 0.5–2.0 s while the CPU time taken by PSGA is in the range of 5.0–789 s. However, in every case PSGA outperforms LMT in terms of the solution quality (that is, schedule length). Even though LMT is a fast heuristic, it may not be a practical approach due to its poor solution quality in term of schedule length.

Test suite 5: As the PSGA-based task scheduling technique embeds a problem-specific fast heuristic in a genetic algorithm, therefore, in this test suite we demonstrate the effect of embedding different heuristics. Task graphs, their associated ECTs and data transfer rate matrices for a 16-machine DHC system were generated by using Wang's graph generator program. We compared the performance of PSGA by embedding in it the EST and EFT as decoding heuristic,

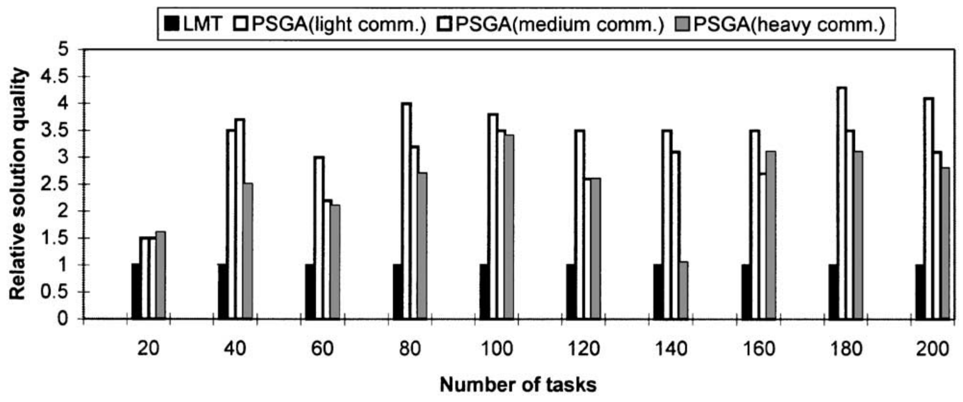


FIG. 10. Comparison of relative solution quality of PSGA and LMT for a set of test graphs, each with three different communication patterns (i.e., light, medium and heavy communication) for an 8-machine DHC system.

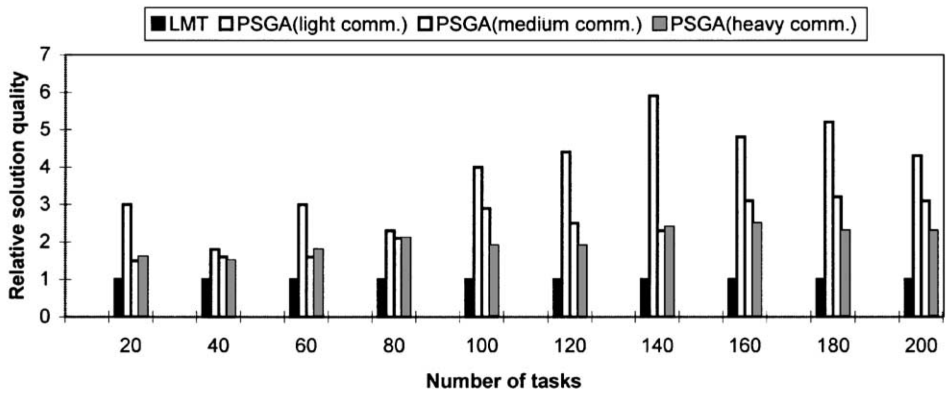


FIG. 11. Comparison of relative solution quality of PSGA and LMT for a set of test graphs, each with three different communication patterns (i.e., light, medium and heavy communication) for a 20-machine DHC system.

alternatively. Figure 12 shows the comparison of relative solution quality for the 16-machine DHC system. Each column in the figure corresponds to the average taken over light-, medium- and heavy-communication cases, generated for a same-size task graph. It is clear from the results that EFT as decoding heuristic outperforms EST.

6. CONCLUSIONS

In DHC systems, proper matching of tasks among machines, determination of their execution orders, and data transfer schedules are important factors for utilizing diverse available resources efficiently. In this paper, we presented a problem-space genetic algorithm-based integrated technique for the static task matching and scheduling problem for distributed heterogeneous computing systems including the

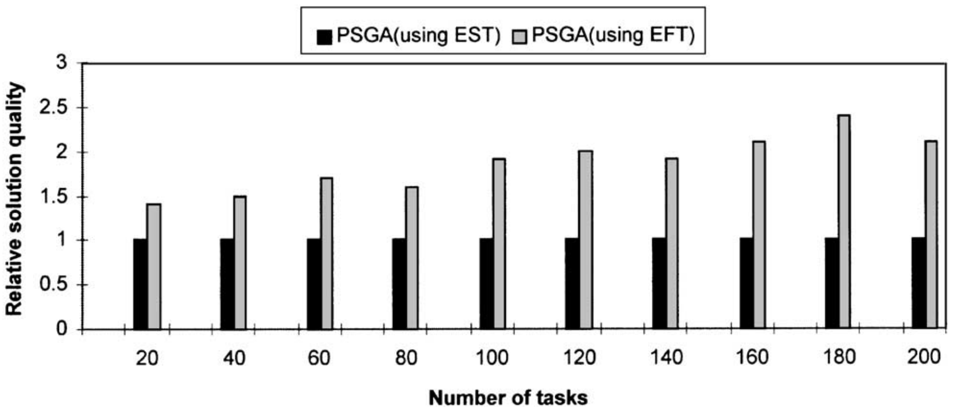


FIG. 12. The relative performance of the EST heuristic versus the EFT heuristic, when embedded in PSGA for a 16-machine DHC system.

communication delays, with the objectives to reduce the schedule length. The proposed technique is an embodiment of a genetic algorithm and a heuristic, which uses a different neighborhood structure to search a large solution space in an intelligent way in order to find the best possible solution within an acceptable CPU time. Simulation tests demonstrate the viability of the proposed technique that performs better as compared to the existing approaches.

ACKNOWLEDGMENTS

Authors thank Mr. Pankaj Shroff and Prof. D.W. Watson of Utah State University, Logan, UT, U.S.A. for providing their GSA code. Thanks are also due to Mr. Lee Wang of Purdue University, West Lafayette, IN, U.S.A. for providing the test-graph generator program and Prof. A. Youssef of George Washington University, Washington, DC, U.S.A. for giving information about his work.

REFERENCES

1. T. L. Adam, K. M. Chandy, and J. R. Dickson, A comparison of list schedules for parallel processing systems, *Comm. ACM* **17**(12) (December 1974), 685–690.
2. I. Ahmad and M. K. Dhodhi, Task assignment using a problem-space genetic algorithm, *Concurrency*: *Pract. Exp.* **7**(5) (August 1995), 411–428.
3. I. Ahmad and M. K. Dhodhi, Multiprocessor scheduling in a genetic paradigm, *Parallel Comput.* **22** (1996), 395–406.
4. L. Davis, Job shop scheduling with genetic algorithms, *in* "Proceedings of the First International Conference on Genetic Algorithms," pp. 136–140, 1985.
5. M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker, Datapath synthesis using a problem-space genetic algorithm, *IEEE Trans. Comput.-Aided Des. Integrated Circuits Systems* **14**(8) (August 1995), 934–944.
6. M. M. Eshagian and M. E. Shaaban, Cluster-M programming paradigm, *Internat. J. High Speed Comput.* **6**(2) (June 1994), 287–309.
7. D. Fernandez-Baca, Allocating modules to processors in a distributed system, *IEEE Trans. Software Eng.* **15**(11) (November 1989), 1427–1436.
8. R. F. Freund and D. Conwell, Superconcurrency: A form of distributed heterogeneous supercomputing, *Supercomputing Rev.* **3** (October l990), 47–50.
9. R. F. Freund, Optimal selection theory for superconcurrency, *in* "Proceedings of the Supercomputing," pp. 699–703, November 1989.
10. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide for the Theory of NP-Completeness," San Francisco, CA, 1979.
11. D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," Addison–Wesley, Reading, MA, 1989.
12. J. Heitkpetter and D. Beasley (Eds.), "The Hitchhikers Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ)," USENET: comp.ai.genetic, December 1997.
13. E. S. H. Hou, N. Ansari, and H. Ren, A genetic algorithm for multiprocessor scheduling, *IEEE Trans. Parallel Distrib. Systems* **5**(2) (February 1994), 113–120.
14. T. C. Hu, Parallel sequencing and assembly line problems, *Oper. Res.* **9** (6) (November 1961), 841–848.
15. M. A. Iverson, F. Ozguner, and G. J. Follen, Parallelizing existing applications in a distributed heterogeneous environment, *in* "Proceedings of the Heterogeneous Computing Workshop," pp. 93–100, April 1995.

16. A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and Cho-li Wang, Heterogeneous computing: Challenges and opportunities, *IEEE Comput.* **26**(6) (June 1993), 18–27.
17. Y. K. Kwok and I. Ahmad, Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm, *J. Parallel Distrib. Comput.* **47**(1) (November 1997), 58–77.
18. G. Liao, E. R. Altman, V. K. Agarwal, and G. R. Gao, A comparative study of multiprocessor list scheduling heuristics, *in* "Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences," pp. 68–77, January 1994.
19. V. M. Lo, Heuristic algorithms for task assignment in distributed systems, *IEEE Trans. Comput.* **37**(11) (November 1988), 1384–1397.
20. N. Mansour and G. Fox, Allocating data to distributed-memory multiprocessors by genetic algorithms, *Concurrency*: *Pract. Exp.* **6**(6) (September 1994), 485–504.
21. Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs," Springer-Verlag, New York, 1992.
22. B. Naharai, A. Youssef, and H. A. Choi, Matching and scheduling in a generalized optimal selection theory, *in* "Proceedings of the Heterogeneous Computing Workshop," pp. 3–8, April 1994.
23. P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments, *in* "Proceedings of the Heterogeneous Computing Workshop," pp. 98–104, April 1996.
24. 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, 1996.
25. H. Singh and A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms, *in* "Proceedings of the Heterogeneous Computing Workshop," pp. 86–97, April 1996.
26. R. H. Storer, D. S. Wu, and R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling, *Manage. Sci.* **38**(10) (October 1992), 1495–1509.
27. G. Syswerda, Schedule optimization using genetic algorithms, *in* "Handbook of Genetic Algorithms" (L. Davis, Ed.), pp. 332–349, Van Nostrand Reinhold, New York, 1991.
28. 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 the Heterogeneous Computing Workshop," pp. 109–120, April 1995.
29. L. Tao, B. Narahari, and Y. C. Zhao, Heuristics for mapping parallel computations to heterogeneous parallel architectures, *in* "Proceedings of the Workshop on Heterogeneous Processing," pp. 36–41, 1993.
30. L. Wang, H. J. Siegel, and V. P. Roychowdhury, A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments, in "Proceedings of Heterogeneous Computing Workshop," pp. 72–85, April 1996.
31. M. Wang, S. Kim, M. A. Nichols, R. F. Freund, H. J. Siegel, and W. G. Nation, Augmenting the optimal selection theory for superconcurrency, *in* "Proceedings of the Workshop on Heterogeneous Processing," pp. 13–22, March 1992.
32. 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 the Heterogeneous Computing Workshop," pp. 24–37, April 1996.
33. D. Whitley, T. Starketweather, and D. Daniel Shaner, The travelling salesman and sequence scheduling: Quality solutions using genetic edge recombination, *in* "Handbook of Genetic Algorithms" (L. Davis, Ed.), pp. 350–372, Van Nostrand Reinhold, New York, 1991.
34. D. Whitley, T. Starkweather, and D. A. Fuquay, Scheduling problems and travelling salesman: The genetic edge recombination operator, *in* "Proceedings of the Third International Conference on Genetic Algorithms," pp. 133–140, June 1989.

35. J. Yang, I. Ahmad, and A. Ghafoor, Estimation of execution times on heterogeneous supercomputer architecture, *in* "Proceedings of the International Conference on Parallel Processing," Vol. I, pp. 219–225, August 1993.

IMTIAZ AHMAD received his B.Sc. in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, an M.Sc. in electrical engineering from the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, and a Ph.D. in computer engineering from Syracuse University, Syracuse, New York, in 1984, 1988 and 1992, respectively. Since September 1992, he has been with the Department of Computer Engineering at Kuwait University, Kuwait, where he is currently an associate professor. His research interests include design automation of digital systems, high-level synthesis, and parallel and distributed computing.

ISHFAQ AHMAD received a B.Sc. in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985. He received his M.S. in computer engineering, and Ph.D. in computer science, both from Syracuse University in 1987 and 1992, respectively. He is a professor of computer science and engineering at the University of Texas at Arlington. Prof. Ahmad's research interests span the areas of high-performance computing, parallel and distributed systems, video compression, and networked multimedia information technologies. Prof. Ahmad has authored or co-authored over 100 technical papers in refereed journals and conferences, including the best paper awards at Supercomputing 90, Supercomputing 91, and ICPP 2001. He serves on the editorial boards of the Journal of Parallel and Distributed Computing, IEEE Concurrency, Cluster Computing, IEEE Transactions on Circuits and Systems for Video Technology, and the new IEEE Distributed Systems Online.

ANWAR ALYATAMA received his Ph.D. from Georgia Institute of Technology. He is an assistant professor with the Department of Computer Engineering and Assistant Vice-President for Academic Services at Kuwait University. His research interests include wireless and broadband networks, queuing analysis and simulation. Dr. Alyatama is a member of IEEE.

MUHAMMAD K. DHODHI received a B.Sc. (with honors) in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan in 1982. He received two Master's degrees, one in computer and systems engineering and another in electric power engineering, from Rensselaer Polytechnic Institute, Troy, New York, in 1984 and 1986, respectively. He received a Ph.D. in electrical engineering from Lehigh University, Bethlehem, PA, in 1992. Dr. Dhodhi has experience both in the industrial as well as academic environments. Dr. Dhodhi is currently a member of technical staff with Lucent Technologies, Inc., Westford, MA. He is working on the design and verification of ASICs/FPGAs to be used in a Multiservice Packet Switch. In the past, he has worked as a member of technical staff with distinguished organizations such as Nortel Networks, Ottawa, Canada, in 1997, and from 1998 to 1999, with VHDL Technology Group, Allentown, PA from 1991 to 1993, with IBM Corporation, Endicott, New York in 1990 and with Silc Technologies, Burlington, MA in 1987.

In academia, Dr. Dhodhi had worked as an assistant professor with the Department of Electrical and Computer Engineering, Kuwait University, from February 1993 to May 1997. He was an associate professor of electrical and computer engineering, Kuwait University, Kuwait from May 1997 to January 2000. While at Kuwait University, he took the leave of absence and worked with Nortel Networks on the design and verification of ASICs used in terabit switch/routers. Dr. Dhodhi's research interests are in parallel/distributed computing, VLSI design/verification, scheduling and traffic management in high-speed networks, and dense wavelength division multiplexing (DWDM)-based optical networks. He was a member of program committee for Lucent's Knowledge Sharing Summit in 2000 and served as a guest co-editor of Computer Communications Journal's special issue on high-speed networks, in 2001.

Dr. Dhodhi is a member of IEEE Computer and Communications Societies, member of IEEE Standards Association (IEEE-SA). He is also a member of ACM SIGDA and SPIE.