

GAA: A New Optimization Technique for Task Matching and Scheduling in HCSs*

PO-JEN CHUANG, CHIA-HSIN WEI AND YU-SHIAN CHIU

Department of Electrical Engineering

Tamkang University

Tamsui, 251 Taiwan

E-mail: pjchuang@ee.tku.edu.tw

A new optimization technique, called the Genetic Annealing Algorithm (GAA), is proposed in this paper to solve the task matching and scheduling problem in a heterogeneous computing system (HCS). Simple in design and easy to implement, the GAA employs only a stir operation, a novel idea based on the annealing concept, to locate optimal solutions for the problem. Extensive simulation runs have been conducted to evaluate and compare the performance of the proposed GAA with that of other optimization techniques, such as the Genetic Algorithm, Simulated Annealing, and Guided Evolutionary Simulated Annealing approaches. The GAA is shown to consistently perform better than the other techniques in terms of speedup, running time, cost, and complexity.

Keywords: heterogeneous computing systems, task matching and scheduling, optimization techniques, simulation, performance evaluation

1. INTRODUCTION

A heterogeneous computing system (HCS) contains computing resources with different capabilities [1]. Data repositories in an HCS are interconnected by means of heterogeneous local and wide area networks. Applications submitted from various user sites share such resources as computing/communication resources and file servers. Programs executed in such an environment typically consist of one or more subtasks that communicate and cooperate to form a single application. Assigning application tasks to suitable resources and ordering task executions in time to optimize a specific objective function, thus, are the goals of task matching and scheduling in an HCS. For an HCS program to achieve high performance, scheduling decisions must be made based on all the required resources, including computing nodes and file servers. However, most existing algorithms devised to map applications in the HCS (e.g., [2-8]) focus mainly on computing resources.

In an HCS, a program is often executed by accessing remote files. Since files may be replicated and their locations are transparent to the executed program, the system needs to select a proper file server to transmit files effectively. Given that a task is assigned to a computing node that provides the best estimated computation time, if the re-

Received May 23, 2003; revised December 8, 2003; accepted June 15, 2004.

Communicated by Yu-Chee Tseng.

* A preliminary version of this paper was presented at the 9th International Conference on Parallel and Distributed Systems, 2002.

quired input file is retrieved from an improper file server, then performance can still get affected. To solve the task matching and scheduling problem more effectively, we take both computing nodes and file servers into account and employ a new optimization technique. The new technique preserves the advantages of previous optimization techniques, including the Genetic Algorithm (GA) [2, 3], Simulated Annealing (SA) [9] and Guided Evolutionary Simulated Annealing (GESA) [10], while offering advanced features of its own.

The proposed optimization technique, called the Genetic Annealing Algorithm (GAA), is simple in design and easy to implement. It involves the use of only a stir operation, which is a novel idea based on the annealing concept, to obtain optimal solutions for newly evolved generations. The solution representation of the GAA, similar to that of the other techniques, consists of a string of characters and represents the characteristics of each possible solution. The strings are like chromosomes, and the characters in the strings are like genes. Only when the genes inside the chromosomes are completely stirred can the positions of the genes be fully changed, thus preventing good solutions from being lost.

The GAA first selects the best solution from among all possible solutions as the desirable local solution and defines its region as the desirable region. The search space can be narrowed down to this region through genetic annealing. Then the GAA fully stirs up solutions in this region, divides it into several smaller regions, and finds a new desirable solution and region. The same searching process is repeated until the optimal solution is located. The stir operation of the GAA involves four parameters: the number of genes to be stirred, the decreased number of genes to be stirred, the number of stirs, and the decreased number of stirs. With the four parameters appropriately defined and the genes in the chromosomes completely stirred, the stir operation can bring up all possible solutions scattered around different regions of the search space, preserve good genes, and pass them down to new generations until a final optimal solution is obtained.

Extensive simulation runs have been conducted to evaluate and compare the performance of the proposed GAA with that of other previous optimization techniques, such as the GA, SA, and GESA. As the collected results indicate, the GAA consistently achieves better performance than the other techniques in terms of speedup, running time, cost, and complexity

2. THE PROBLEM STATEMENT

Tasks with various requirements are submitted from participant computing nodes. A program consisting of a number of tasks is represented by a Directed Acyclic Graph (DAG). File requirements are specified at the task level. The input data items of a task can be data from its parents and/or file sets from file servers. All input data items (the data and the files) must be retrieved before execution of the task begins. After a task is executed, the generated output data is forwarded to its child task. A task is ready to be executed only after all its parent tasks have been executed and the data have been transmitted (from its parents) to it. A ready task that is ready then starts to retrieve the needed file sets. Files are distributedly stored in the system's file servers. Each file may have several (a fixed quantity ≥ 2) identical copies, any of which can satisfy the computing

node requesting it. As multiple users share the resources, optimizing the performance of an individual program may dramatically affect the completion time of other programs. The main goal of this paper is to minimize the overall execution time for a collection of programs by appropriately matching tasks with computing nodes and adaptedly scheduling their execution based on all file requirements. For this purpose, we define our objective function as

$$\left\{ \max_{i=1}^a [PFT(A_i)] \right\}, \text{ where } PFT(A_i) \text{ is the finishing time of program } A_i.$$

When the above objective function value is minimized, the overall schedule length will be reduced to a minimum. To facilitate later discussion, the Directed Acyclic Graph (DAG) model, the system model, and some definitions are given below.

The DAG Model The DAG model is a generic model of a parallel program consisting of a set of interdependent tasks [11]. Each task is an undividable unit of execution, expressed by a vertex with one or more inputs. The vertex, triggered to execute when all inputs are available, generates its outputs following execution. A set of t tasks ($T = \{t_1, t_2, \dots, t_t\}$) in t vertices (n_1, n_2, \dots, n_t), respectively, are connected by a set of directed edges, each of which is denoted by (n_i, n_j) where n_i is the parent and n_j is the child. That indicates t_i should be completed before t_j begins to execute. The weight of a vertex (e_{iu}) is the estimated computation time of task t_i on computing node p_u ; the weight of an edge (d_{ij}) is the data volume which task t_i should transmit to task t_j [3, 11]. A vertex without a parent is an entry vertex; a vertex without a child is an exit vertex. A typical DAG is depicted in Fig. 1. The height value for each vertex is defined as

$$\text{height}(t_i) = \begin{cases} 0, & \text{if } \text{parent}(t_i) = \phi, \text{ where } \text{parent}(t_i) \text{ is} \\ & \text{the set of parent tasks of } t_i \\ 1 + \max_{t_k \in \text{parent}(t_i)} \text{height}(t_k), & \text{otherwise.} \end{cases}$$

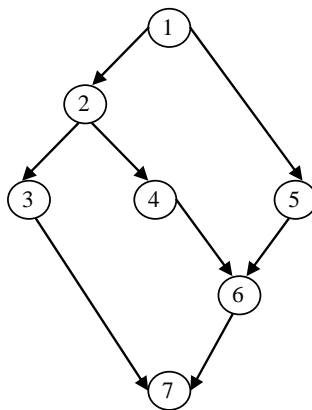


Fig. 1. An example DAG.

The System Model Our heterogeneous computing system, based mostly on new definitions and partly on previous works [6, 7], includes p computing nodes ($P = \{p_1, p_2, \dots, p_p\}$), s file servers ($S = \{s_1, s_2, \dots, s_s\}$), t tasks ($T = \{t_1, t_2, \dots, t_t\}$), and f files ($F = \{f_1, f_2, \dots, f_f\}$). F is a vector, where f_i is the file size of file i .

Matrices for Storing Related Information

PP_{comm.}(u, v) Communication cost between computing nodes p_u and p_v . (An example is given in Table 1.)

SP_{comm.}(u, v) Communication cost between file server s_u and p_v . (Table 2)

The DAG matrix Stores the Directed Acyclic Graph, where d_{ij} is the data value which task t_i should transmit to task t_j .

The ECT matrix Stores the estimated computation time of t_i on p_u . i.e., e_{iu} . (Table 3)

$t_Need_f_{ij}$ = 1 or 0 indicates whether t_i needs file f_j or not. (Table 4)

$fStore_{ij} = k$ indicates that f_i has copy j stored in file server s_k . (Table 5)

Table 1. The PP matrix.

PP	p_1	p_2	p_3
p_1	0	1	2
p_2	1	0	3
p_3	2	3	0

Table 2. The SP matrix.

SP	p_1	p_2	p_3
s_1	1	4	2
s_2	3	2	1
s_3	4	2	3

Table 3. The ECT matrix.

ECT	p_1	p_2	p_3
t_1	5	4	3
t_2	4	3	1
t_3	1	3	4
t_4	2	5	3
t_5	3	2	1
t_6	2	1	4
t_7	1	2	5

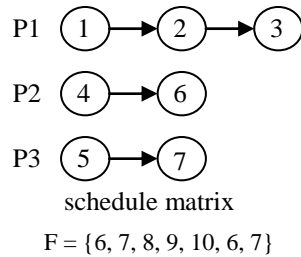


Table 4. The $tNeedf$ matrix.

t_Need_f	f_1	f_2	f_3	f_4	f_5	f_6	f_7
t_1	1	1	0	0	0	0	0
t_2	0	0	1	1	0	0	0
t_3	0	0	0	0	1	1	0
t_4	1	0	0	0	0	0	1
t_5	0	1	0	1	0	0	0
t_6	0	0	1	0	1	0	0
t_7	0	0	0	1	0	0	1

Table 5. The $fStore$ matrix.

$fStore$	Copy1	Copy2
f_1	1	2
f_2	2	3
f_3	1	3
f_4	1	2
f_5	2	1
f_6	3	2
f_7	3	1

Formulation

DCT(t_i, t_j) Data Communication Time between tasks t_i and t_j , determined by both the data volume and communication delay. If t_i and t_j are, respectively, assigned to p_u and p_v , then $DCT(t_i, t_j) = d_{ij} \times PP_{comm.}(u, v)$.

FCT(t_i, f_j, s_v) File Communication Time between t_i and f_j , determined by the file size and communication delay. If t_i is assigned to p_u and f_j is stored in file server s_v , then $FCT(t_i, f_j, s_v) = f_j \times SP_{comm.}(v, u)$.

PAT(t_i) Computing Node Available Time before executing t_i , i.e., the time p_u takes to execute all previously assigned tasks before executing t_i . If task t_k is the last assigned task before t_i , then $PAT(t_i) = \begin{cases} 0, & \text{if } t_i \text{ is the first task schedule on } p_u \\ TCT(t_k), & \text{otherwise.} \end{cases}$

DAT(t_i, t_j) Data Available Time that t_i takes to complete execution on p_v and to transmit all data needed for executing t_j on p_u , defined as $DAT(t_i, t_j) = TCT(t_i) + DCT(t_i, t_j)$.

MaxDAT(t_i) Maximum Data Available Time of t_i , defined as $MaxDAT(t_i) = \max_{t_k \in \text{parent}(t_i)} \{DAT(t_k, t_i)\}$. (As t_i may have many parent vertices, there will be many Data Available Times related to it.)

TRT(t_i) Task Ready Time, the time t_i takes to receive all data needed for execution from its parent and p_u (to which t_i is assigned), defined as $TRT(t_i) = \max\{PAT(t_i), MaxDAT(t_i)\}$.

AT(t_i, s_v) Access Time, the total time needed for t_i to read files from s_v , defined as $AT(t_i, s_v) = \sum_{j \in A_i} FCT(t_i, f_j, s_v)$, where $A_i = \{j \mid \forall l \ t_Need_f_{ij} = 1 \wedge fStore_{jl} = v\}$.

MaxAT(t_i) Maximum Access Time for t_i , defined as $MaxAT(t_i) = \max\{AT(t_i, s_v)\}$. (There can be many different access times for t_i .)

FAT(t_i) File Available Time for t_i , defined as $FAT(t_i) = TRT(t_i) + MaxAT(t_i)$. (When t_i is ready, it starts to read the files.)

TCT(t_i) Task Completion Time, the time t_i takes to receive all required files and to complete its execution on p_u , defined as $TCT(t_i) = FAT(t_i) + e_{iu}$.

PFT Program Finishing Time = $\max_{t_i \in \text{exitnode}} TCT(t_i)$. (The program is finished when all tasks are completed.)

To give an example, t_i in Fig. 2 (a) is assigned to p_x . After t_i is executed, the output data volume is k . t_i is a parent vertex of t_j , and $m = DCT(t_i, t_j)$. In Fig. 2 (b), t_i will read f_i from S_u , and $n = AT(t_i, S_u)$.

Tables 1-5 provide information related to task matching and scheduling. A schedule matrix is produced after all possible task matching and scheduling is performed according to the assumptions in the tables. The following steps can be adopted to get the schedule and its length. (Note that a schedule matrix includes multiple schedules, one schedule for each computing node. The schedule length is the PFT, the program finishing time.)

1. Re-draw the DAG using the representation illustrated in Fig. 2 (a).
2. According to the file requirement and schedule matrix, redraw the DAG (as shown in Fig. 3) using the representation shown in Fig. 2 (b).
3. Based on the information in Fig. 3 and the assumptions for the above formulation, devise the schedule shown in Fig. 4 to get the schedule length.

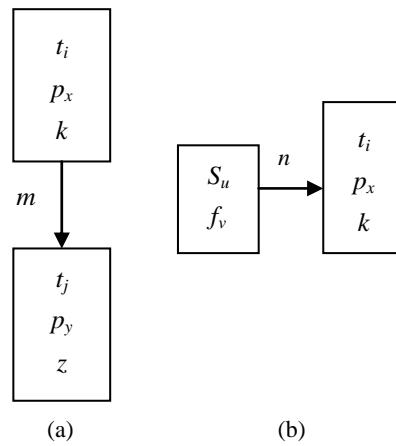


Fig. 2. The part of Fig. 3 used to illustrate (a) the data communication time; (b) the access time.

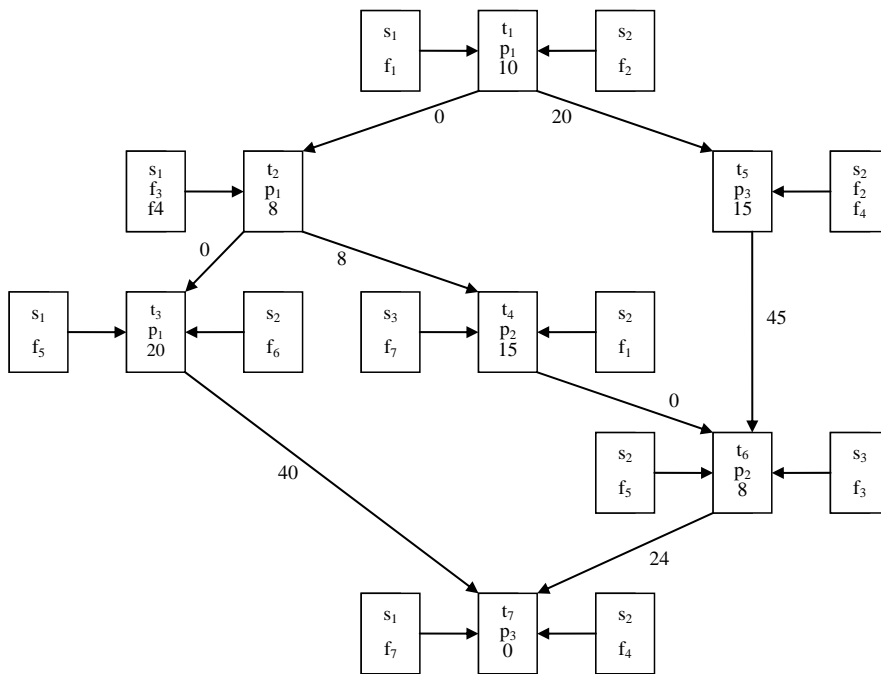


Fig. 3. The DAG with file accesses.

3. PREVIOUS OPTIMIZATION TECHNIQUES

In the case of the GA, SA, and GESA, a solution is represented by a chromosome; in the case of our GAA, it is defined by a schedule matrix. A legal schedule matrix needs to satisfy two conditions [12]:

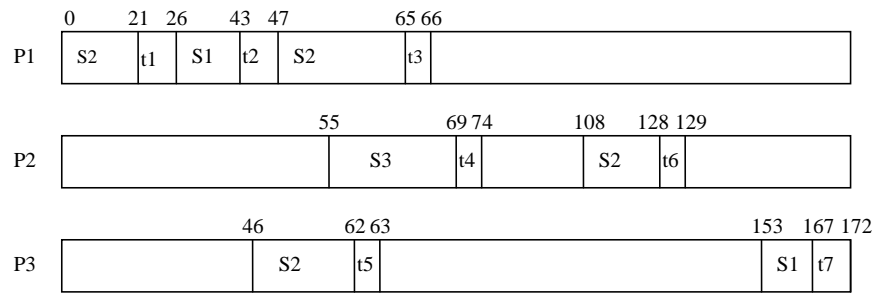


Fig. 4. An example schedule.

- (1) The precedence relations among the tasks are satisfied.
- (2) Every task is present and appears only once in the schedule matrix.

Assigning all tasks to one computing node is a possible solution. Thus, for P computing nodes, there will be P solutions producing P schedule lengths. The maximum schedule length is *Maxlength*, and the minimum is *Minlength*.

Fitness Value

1. If the schedule length \geq Maxlength, the fitness value = 0.1.
2. If the schedule length \leq Minlength, the fitness value = 0.9.
3. If $\text{Minlength} < \text{schedule length} < \text{Maxlength}$,
let unit = $(\text{Maxlength} - \text{Minlength}) \div 10$, $n = (\text{Maxlength} - \text{schedule length}) \div \text{unit}$, and get the fitness value = $n \div 10$.

Crossover Among the many possible ways to perform crossover, we adopt the one proposed in [3]. After a task, say t_i , is randomly selected, crossover is performed between t_i and its child tasks. We swap t_i and its child tasks in chromosomes X and Y as explained in [3].

Mutation

- (1) Randomly select a task t_i on p_u .
- (2) Randomly select a computing node p_v ($p_v \neq p_u$).
- (3) Move task t_i to p_v .

The Genetic Algorithm

Step1: Initialize population

Step2: Evaluate population

Step3: While (the stopping criteria are not met)

- (1) select solutions for the next population,
- (2) crossover,
- (3) mutation,
- (4) evaluate the population.

Step4: Output the best solution found.

Simulated Annealing**Step1:** Choose an initial solution S **Step2:** Set $t = t_0$ **Step3:** Repeat until the system is frozen

- (1) Perform a mutation on S and generate a new solution S' .
- (2) Set $\Delta C = C(S') - C(S)$, where $C(X)$ is the objective function value of X .
- (3) If $\Delta C \leq 0$, set $S = S'$.
- (4) If $\Delta C > 0$, set $S = S'$ with probability $\exp(-\Delta C/t)$.
- (5) Set $t = r \times t$ (r is a random number uniformly distributed between 0 and 1).

Step4: Output the best solution found.**Guided Evolutionary Simulated Annealing****Step1:** Set initial temperature t .**Step2:** Randomly select N parents.**Step3:** Generate children from the parents.**Step4:** Find the best child for each parent (first level competition or local competition).**Step5:** Find the parents for the next generation (selection).

For each family, accept the best child as the parent for the next generation

if $y_1 < y_2$ or $e^{\frac{-(y_1-y_2)}{t}} > \rho$,where y_1 is the objective function value of the best child, y_2 is the objective function value of its parent, t is the temperature coefficient, and ρ is a random number uniformly distributed between 0 and 1.**Step6:** Find the number of children that will be generated from the parents of the next generation (second level competition).**Step7:** Decrease the temperature coefficient.**Step8:** Repeat steps 3 to 7 until an acceptable solution is found or until a certain number of iterations have been performed.**4. THE PROPOSED OPTIMIZATION TECHNIQUE**

The proposed Genetic Annealing Algorithm (GAA) employs only one operation, the *stir* operation. The stir operation involves four parameters: the number of genes to be stirred (N), the decreased number of genes to be stirred (n), the number of stirs (M), and the decreased number of stirs (m). The solution representation of the GAA, similar to that of the other techniques, consists of a string of characters and represents the characteristics of each possible solution. The strings are like chromosomes; the characters in the strings are like genes. When the genes inside the chromosomes are completely stirred, the positions of the genes can be fully changed to prevent good solutions from being lost. Denoted by $\text{Stir}(X, N)$, where X represents any solution and N is the number of genes to be stirred, the stir operation works as follows. N genes (characters), are randomly selected from an X (string of characters). The values of the selected genes or their positions in the chromosomes are changed. The genes in the chromosomes are completely stirred to bring up all possible solutions scattering over the search space.

4.1 The Stir Operation and its Four Parameters

```

Randomly generate a solution X;
for (i = N; i > 0; i -= n)          /* i = i - n */
{
  for (j = 0; j < M; j++)
  {
    Y = stir(X, i);
    if (F(Y) < F(X)) X = Y;      /* F(X) is the object function value of X */
  }
  M -= m;                          /* M = M - m */
}

```

The number of genes to be stirred (N). The number of genes subject to changes in the operation. The proper value of N is defined as being equal to the number of genes in a chromosome in GA. (Note that $N >$ the number of genes in a chromosome indicates that some genes are changed more than once, which is not necessary, while $N <$ the number of genes in a chromosome indicates that search is not conducted from the whole search space, and that potentially good solutions are likely to be lost.)

The decreased number of genes to be stirred (n). The number of genes to be stirred decreases with each new generation. If n indicates such a decreased number, then $G = N/n$ (similar to the number of generations in the GA). The value of n can range from 1 to the number of genes in a chromosome and may affect the search time and the likelihood of obtaining optimal solutions. With a large n , the search for solutions can be accelerated, but the likelihood of obtaining optimal solutions may be impaired as rapid search tends to overlook a number of potential solutions scattered around initial solutions. $n = 1$ is desirable as a more extensive annealing process is performed, which helps to obtain optimal solutions and restrains the search time.

The number of stirs (M). The time spent finding the optimal solution under the same N . Similar to the population in the GA, M increases as the search space grows. When M increases, the search time also grows. Thus, the size of search space and the length of the search time must both be taken into account to decide on a proper M . We let $M =$ the population in the GA.

The decreased number of stirs (m). Like the number of genes to be stirred, the number of stirs also decreases with each new generation. During the search for optimal solutions, the number of stirs usually remains fixed (m is set to 0 in our operation), but when the search time becomes unreasonably large, m can be adjusted to reflect this situation.

4.2 Operating Steps of the Genetic Annealing Algorithm

Step 1: Fix the four parameters of the stir operation, i.e., N , n , M , and m .

Step 2: Randomly generate a solution X .

Step 3: Perform Stir(X, N) on X to generate a new solution Y .

Step 4: If $F(Y) < F(X)$, Y becomes the new solution; if $F(Y) \geq F(X)$, X remains the solution.

Step 5: Repeat steps 3 and 4 M times. Decrease N ($N = N - n$) and if necessary decrease M ($M = M - m$). Repeat steps 3 and 4 again until $N = 0$.

As the steps indicate, the GAA first selects the best solution from among all possible solutions as the desirable local solution and defines its region as the desirable region. When a desirable region is located, good genes are preserved and passed down to new generations by means of genetic annealing, which gradually shrinks the search space until a final optimal solution is found. The search proceeds as follows: solutions are fully stirred up in the desirable region; the region is divided into several smaller ones to find a new desirable solution and region; repeat the same search process until the final optimal solution is located. In such an evolutionary search process, the stir operation is able to preserve the fittest solution while eliminating unfit ones because extensive stirring increases the likelihood of obtaining optimal solutions in different regions of the search space.

Among the four parameters, the number of genes to be stirred (N) and the decreased number of genes to be stirred (n) determine the quality of the solutions, while the number of stirs (M) and the decreased number of stirs (m) determine the search time. With the four parameters appropriately defined, the GAA can carry out search in various regions of the search space, thus avoiding the need to take a local desirable solution as the final optimal one.

4.3 Advantages of the GAA

The process of selecting solutions is simple. Like the SA, the GAA selects possible solutions by continually generating new (superior) solutions to replace the old (inferior) ones. The process is simpler than those of the GA and GESA, which generate a fixed number of solutions at the beginning and replace them based on a certain selection policy. Experimental evaluation also shows that, among the four optimization techniques, the GAA searches for the least possible solutions in the search space to reach the stopping criteria (i.e., to generate the final optimal solution).

The new concept of cooling down gene-stirring helps preserve good genes. The process performed to decrease N with each new generation cools down the temperature in a more effective way than the SA does. In the case of the SA and GESA, cooling down the temperature means turning away from the local best solution and, the temperature reflects only the probability value. The GAA sets up more genes to be stirred with higher temperatures at the beginning and decreases the number of genes to be stirred throughout evolution to cool down the temperature. As the stir operation changes only some of the genes during the search process, good genes are more likely to be passed down to new generations than to be left out.

Performance can be easily evaluated. In the case of the GA and SA approaches, a single unique condition set for an operation, either reproduction, crossover, mutation, population, or convergence, may yield quite different results. In contrast, the GAA re-

tains more objectivity in solving problems as it fixes the values of the four parameters only and, without setting extra conditions for convergence, it converges upon reaching the stopping criteria.

Easy to implement. The GAA consists of two for nested loops, whose entire flow is easy and clear. The stir operation also makes implementation of the algorithm complexity-efficient.

Extensible. The complexity of the GA increases as the search space grows and may outgrow the search space. In terms of extensibility, the GAA turns out to be better as it oversees complexity by controlling the number of genes to be stirred and the decreased number of genes to be stirred.

Programmable. To change the pre-fixed operation of crossover or mutation in any genetic algorithm, the algorithm itself needs to be changed first. But to change the stir operation of the GAA, we need to change only the values of the parameters. (As a matter of fact, the four parameters of the stir operation represent certain operations in genetic algorithms. For instance, the number of stirs can be associated with the population in the GA, and when the number of genes to be stirred is set to 1 ($\text{Stir}(X, 1)$), the stir operation becomes a mutation operation ($\text{Stir}(X, 1) \equiv \text{Mutation}(\)$). With a fixed N , a preset M , n , and $m = 0$, the GAA performs like the SA. (The SA is, indeed, a particular case of the GAA.)

Adjustable. The random search in the GAA is not unconditioned free-spin search; it is, instead, an efficient approach that decides the next search step based on the result of the last search. This step-by-step adjustment approach enables the GAA to keep the entire search process on the right track.

5. PERFORMANCE EVALUATION

Extensive simulation runs were conducted to evaluate and compare the performance of the proposed GAA with that of other optimization techniques. Two terms are provided first [8].

Communication to Computation Ratio (CCR) CCR is the ratio of the average communication cost to the average computation cost. If a DAG's CCR value is low, then it can be considered to be a computation-intensive application; otherwise, it is a communication-intensive application.

Parallelism In a DAG, the average number of tasks at each height is called the parallelism. For instance, p_2 (parallelism = 2) in Figs. 5 and 6 indicates that two tasks, on average, are at each height of the DAG.

The Simulation Model In our simulation, the input parameters are assumed to be as follows: the number of computing nodes = {2, 4, 6, 8, 10}, the number of file servers =

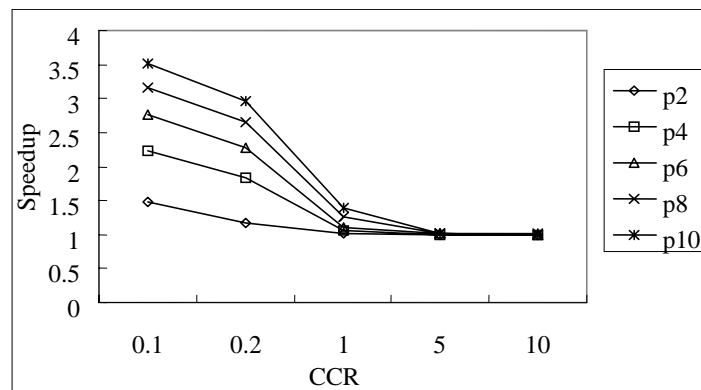


Fig. 5. Speedup versus CCR under different levels of parallelism with no file accesses.

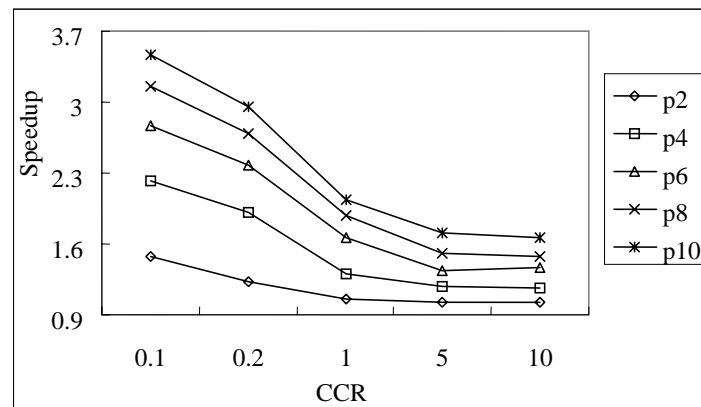


Fig. 6. Speedup versus CCR under different levels of parallelism with file accesses.

{2, 4, 6, 8, 10}, the number of tasks = {20, 40, 60, 80, 100}, the number of files = {10, 20, 30, 40, 50}, CCR = {0.1, 0.2, 1, 5, 10}, and parallelism = {2, 4, 6, 8, 10}. The values of the four parameters for the stir operation are N = the number of computing nodes, $n = 1$, M = the population in the GA, and $m = 0$. The control parameters used in [3] are adopted here for the GA: crossover probability = 0.9, mutation probability = 0.3, restarting rate = every 100 generations, and maximum generation = 600. Task matching and scheduling are conducted under two assumptions – with file accesses and without -- using the above four optimization techniques. Speedup, running time, cost, and complexity are the four performance matrices adopted in this simulation. The collected results are reasonably accurate. The first speedup value was 1.471 for $p2$ (parallelism = 2) and CCR = 0.1 as shown in Fig. 5. Given 95% confidence, the value has a calculated confidence interval half width over the 25 replications equal to 0.036, meaning that we are 95% confident that the true result would fall into the interval 1.471 ± 0.036 . The value involves only a 2.45% error.

Performance Evaluation

5.1 Speedup

$$Speedup = \frac{Minlength}{schedule\ length}$$

As mentioned previously, one possible solution of the task matching and scheduling problem is to assign all tasks to one computing node. That is, with P computing nodes, there will be P solutions, which will then produce P schedule lengths. We, thus, have $Minlength = \text{computation cost} + \text{file communication cost}$ (because $schedule\ length = \text{computation cost} + \text{communication cost}$, and $communication\ cost = \text{data communication cost} + \text{file communication cost}$; there is no data communication in this case).

Speedup vs. CCR, parallelism and computing nodes. Fig. 6 depicts the speedup obtained with file accesses. As the result shows, speedup decreases as CCR increases but increases as parallelism increases (because there is a file communication cost for $Minlength$). For the case without file accesses shown in Fig. 5, speedup also decreases as CCR increases. When $CCR \leq 1$, speedup increases as parallelism increases; when $CCR > 1$, speedup is very similar for different scales of parallelism (as communication cost for $Minlength$ is 0).

Figs. 7 and 8 show that with file accesses or without, speedup increases as the number of computing nodes increases. When parallelism ≤ 2 , increasing the number of computing nodes fails to shorten the schedule length; instead, it increases the data communication cost and, thus, reduces speedup. Parallelism increases speedup only when the number of computing nodes ≥ 4 . Clearly, speedup is affected more by parallelism than by the number of computing nodes.

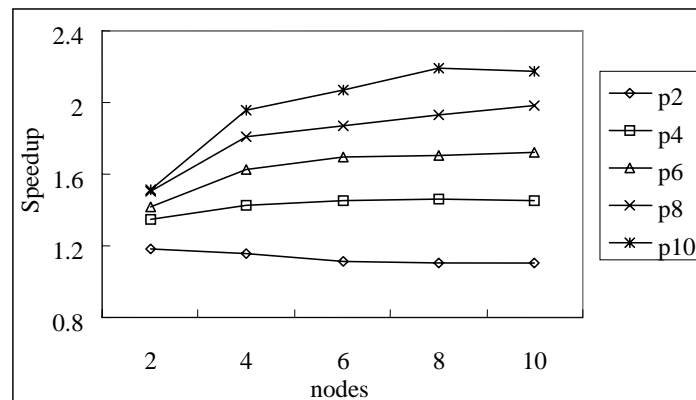


Fig. 7. Speedup versus the number of computing nodes under different levels of parallelism with no file accesses.

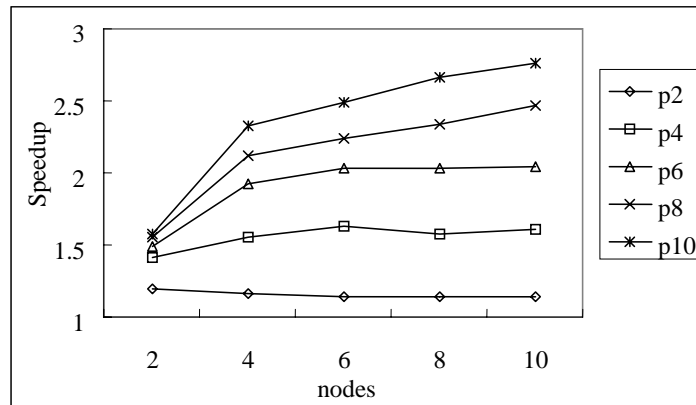


Fig. 8. Speedup versus the number of computing nodes under different levels of parallelism with file accesses.

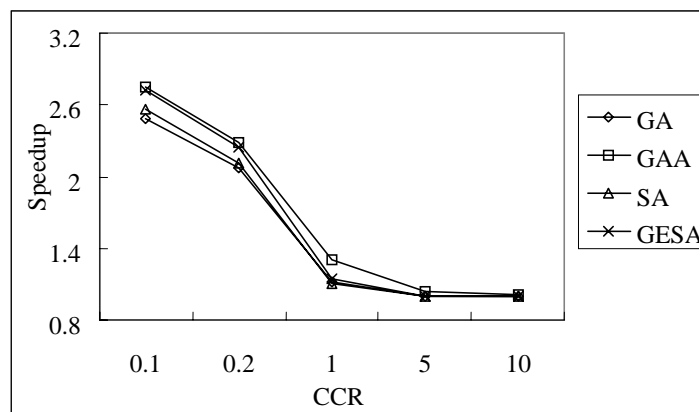


Fig. 9. Speedup versus CCR (with no file accesses) for schedules derived using different optimization techniques.

Speedup for the four techniques. Figs. 9 and 10 show the speedup results for schedules derived from the GAA, GA, SA, and GESA. It should be noted that either with or without file accesses, speedup obtained under different CCR's displays the same trend as shown in Figs. 5 and 6. Without file accesses (Fig. 9), the GAA has the largest speedup when $CCR < 10$; when $CCR = 10$, the advantage becomes less evident due to the high communication cost involved in the schedule. When file accesses are considered (Fig. 10), the GAA exhibits the best speedup under any CCR because the file communication cost is involved in both Minlength and the schedule length. Figs. 11 and 12 reveal that speedup increases as parallelism grows, and that among the four techniques, the GAA yields the highest speedup with or without file accesses.

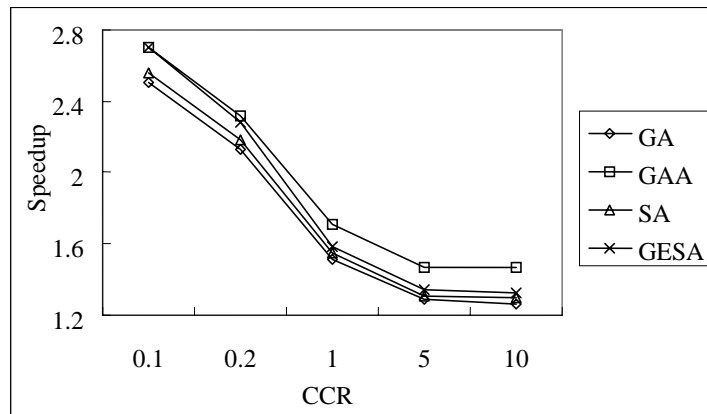


Fig. 10. Speedup versus CCR (with file accesses) for schedules derived using different optimization techniques.

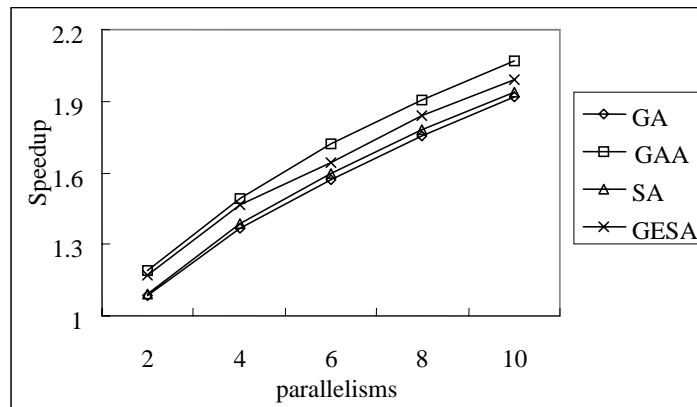


Fig. 11. Speedup versus parallelism (with no file accesses) for schedules derived using different optimization techniques.

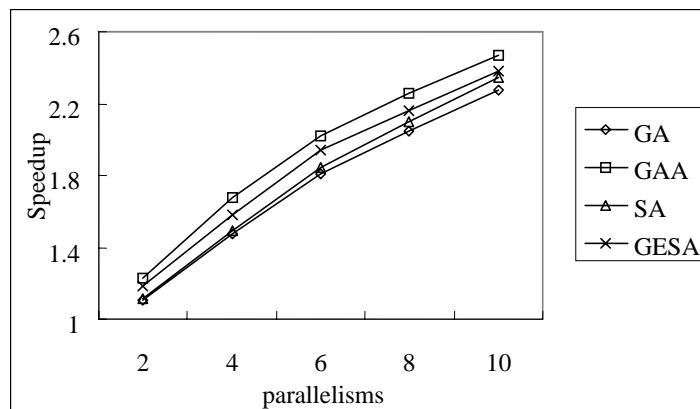


Fig. 12. Speedup versus parallelism (with file accesses) for schedules derived from different optimization techniques.

We can thus draw the following conclusion: Speedup can be enhanced by decreasing CCR, increasing parallelism, and adding computing nodes. Among them, decreasing CCR works the best, adding computing nodes proves least effective, and it is only when CCR is low that increasing parallelism and the number of computing nodes help enhance speedup. Furthermore, the GAA consistently yields higher speedup in different situations mainly because the stir operation performs better than the crossover operation in finding and passing down good genes.

5.2 Running Time

The running time is the time each optimization technique takes to obtain the optimal schedules under the given assumptions. It shows how fast an algorithm works and how soon it converges. As an algorithm stops when convergence is reached, its running time can be prolonged if it fails to reach convergence soon enough and has to run extra loops.

Running time for the four techniques. Figs. 13 and 14 show that either with or without file accesses, running time increases with the number of tasks. Without file accesses (Fig. 13), the GA has the longest running time due to its lengthy algorithm and complicated operations, but when the number of tasks increases, its running time increases only moderately (because its prefixed populations will not increase with the number of tasks). The proposed GAA has the shortest running time and the most modest buildup time due to the increased number of tasks, thanks to its neat algorithm and simple operation. The running time for the SA is slightly longer than that for the GAA, and the buildup time is also slightly longer. When the number of tasks increases, the GESA shows the most rapid increase in the running time of all the approaches. This is because when the number of tasks is small, the GESA needs to execute fewer loops before reaching convergence, but when the number of tasks increases to a certain degree, it needs to execute more loops due to the expanded search space, thus prolonging the running time. With file accesses, the results are similar (Fig. 14), but the calculation of the objective function becomes more complicated, which results in a longer running time and buildup time for all four techniques.

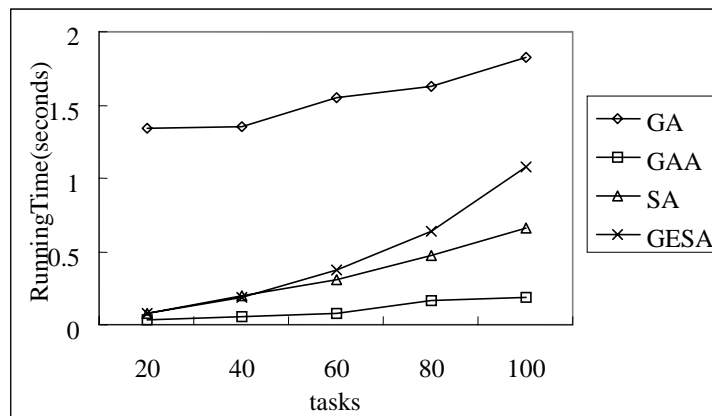


Fig. 13. Running time versus the number of tasks (with no file accesses) with different optimization techniques.

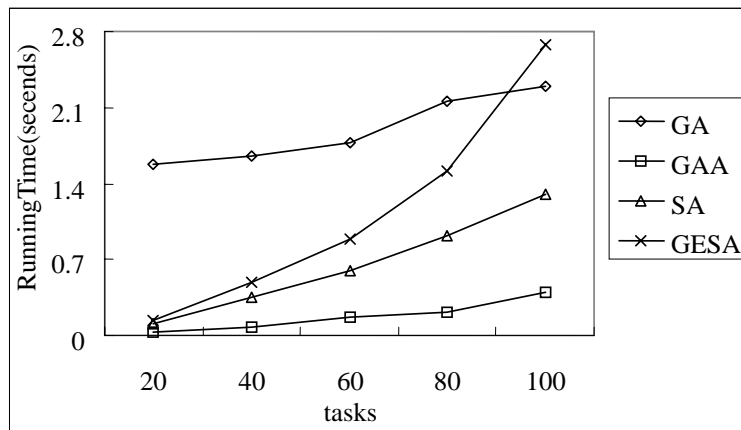


Fig. 14. Running time versus the number of tasks (with file accesses) with different optimization techniques.

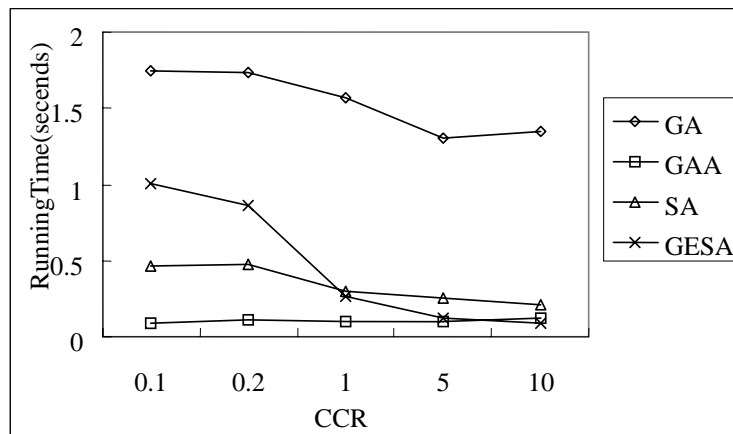


Fig. 15. Running time versus CCR (with no file accesses) with different optimization techniques.

Running time vs. CCR. Running time versus CCR is shown in Figs. 15 and 16. The result for the GESA reveals clearly that the running time of an algorithm can well indicate its convergence speed (i.e., how fast it reaches convergence). CCR is assumed to have no impact on running time, but the result shown here reveals that the running time for the GESA increases as CCR decreases. This happens because when CCR increases, it becomes more difficult to find the optimal solution; convergence will thus be reached at an early stage. Fig. 15 depicts the case without file accesses, where the suddenly shortened running time for the GESA is more visible due to the high communication cost involved in the schedule. This indeed reveals the adaptability of the GESA, whereby it is able to adjust its running time according to the simulation parameters.

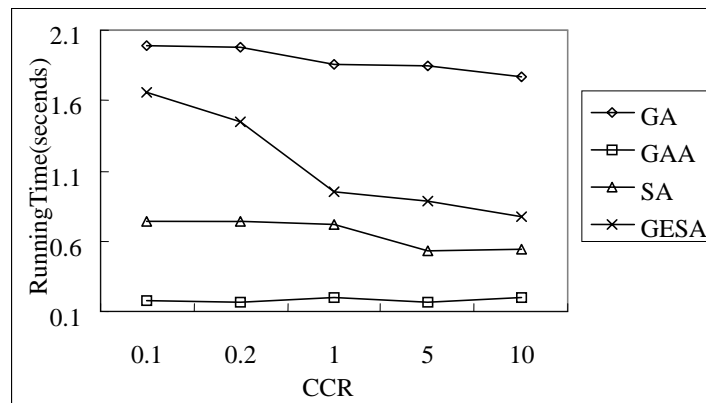


Fig. 16. Running time versus CCR (with file accesses) with different optimization techniques.

5.3 Cost

As previously mentioned, speedup represents the quality of the optimal solutions, while the running time indicates the speed of finding solutions. It is, nevertheless, likely that an algorithm that is able to locate the optimal solutions may have a longer running time, whereas an algorithm with the least running time may fail to find the best solution. To evaluate and compare the performance of different algorithms on a fair basis, we took both speedup and running time into account and defined cost as follows:

$$Cost = \frac{Running\ Time}{Speedup}, \text{ the running time needed to increase speedup by one unit.}$$

The defined cost for each optimization technique was obtained, and the results are shown in Figs. 17 and 18. The results show that the GA has the highest cost, the GAA yields the lowest cost, and the GESA exhibits the most rapidly growing cost (due to the increased number of tasks). The results reveal once again (1) the importance of the stir operation in bringing out more desirable solutions in the case of the GAA and (2) the advantage of simply-designed algorithms, like the GAA, in reducing the running time.

5.4 Complexity

Computing the complexity of different optimization algorithms is never an easy task. The execution times of programs may somehow reflect the complexity, but they should not be the only consideration taken into account when evaluating complexity. Recall that every optimization algorithm employs certain operations to search for optimal solutions. It is, thus, important to find out which operation locates the optimal solution using the simplest technique. As a solution is represented by a string of characters, each operation (generation, crossover, mutation, or stir) needs to perform *insert* and *delete* on the string of characters. When *insert* and *delete* are exercised through program implementation, they can be translated into link changes in the DAG. That is, whatever operation it performed, it will involve certain link changes carried out to rearrange the string of characters.

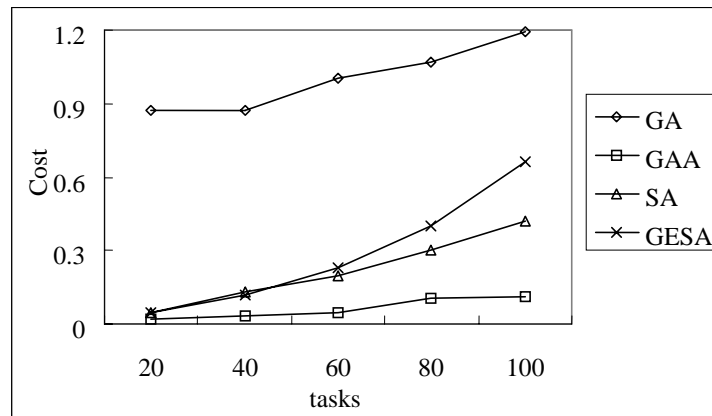


Fig. 17. Cost versus the number of tasks (with no file accesses) with different optimization techniques.

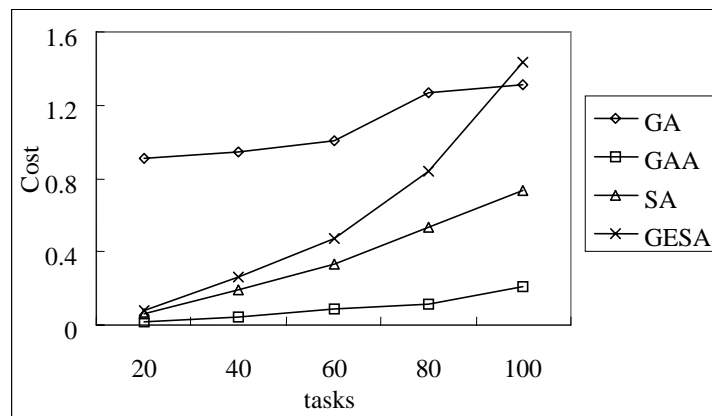


Fig. 18. Cost versus the number of tasks (with file accesses) with different optimization techniques.

To attain more reasonable complexity calculations for different algorithms, we thus take the number of link changes occurring in an operation as the complexity, and take the total number of link changes in all the operations of an optimization technique as its total complexity.

The complexity results for different operations are shown in Fig. 19 (where 1, Tasks/2, and Tasks are the numbers of genes to be stirred in the stir operation, i.e., the N in $\text{Stir}(X, N)$). As one can see, the complexity of the stir operation varies with different N 's and is controllable (because N can be preset). Among the operations, crossover has the highest complexity, while mutation ($= \text{Stir}(X, 1)$) has the lowest. Fig. 20 depicts the complexity results for the four optimization algorithms. As the figures show, the complexity is lower for the GESA and GAA but higher for the GA and SA, indicating that both the GESA and GAA employ operations which are smaller in amount and lower in complexity.

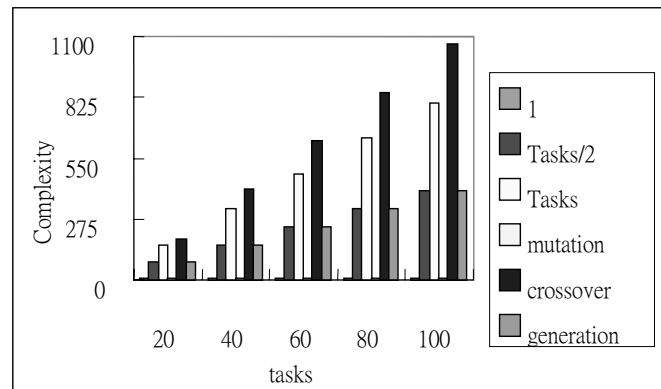


Fig. 19. Complexity comparison between the stir operation and other operations.

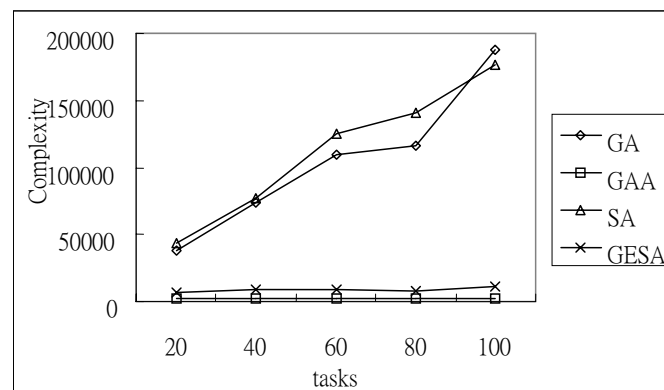


Fig. 20. Complexity versus the number of tasks with different optimization techniques.

6. CONCLUSIONS

A Heterogeneous Computing System (HCS) contains computing resources with different capabilities. For an HCS program to achieve high performance, task matching and scheduling decisions must be made based on all required resources, including the computing nodes and file servers. Taking both computing nodes and file servers into account to achieve optimal performance in the case of an HCS program has been the focus of this research.

A new optimization technique, called the Genetic Annealing Algorithm (GAA), has been proposed in this paper to solve the task matching and scheduling problem in an HCS. Combining the advantages of previous optimization techniques, the proposed GAA is simple in design, easy to implement, and yet efficient in finding optimal solutions for the task matching and scheduling problem. It involves only a stir operation, a novel idea based on the annealing concept. The stir operation is denoted as $\text{Stir}(X, N)$ and is dominated by four parameters: the number of genes to be stirred (N), the decreased number of genes to be stirred (n), the number of stirs (M), and the decreased number of stirs (m). By

completely stirring the genes in the chromosomes, the stir operation is able to bring up all possible solutions scattered around different regions of the search space. Extensive stirring increases the likelihood of obtaining optimal solutions and thus preventing good solutions from being lost during the search process.

Extensive simulation runs have been conducted to evaluate the performance of the proposed GAA and other optimization techniques. As the simulation results reveal, the GAA yields constantly better performance than the GA, SA, and GESA in terms of speedup, running time, cost, and complexity. That is, the GAA is able to locate a desirable solution more rapidly, at lower cost, and with less complexity for the task matching and scheduling problem in an HCS. With certain necessary modifications, the proposed GAA can also be used to solve other optimization problems. In addition, since the current matching and scheduling approaches require detailed information, and the DAG of the user program should be known prior to the execution of each algorithm, our future work will focus on conducting matching and scheduling in a more dynamic way.

REFERENCES

1. R. Freund and H. J. Siegel, "Guest editors' introduction: heterogeneous processing," *IEEE Computer*, Vol. 26, 1993, pp. 13-17.
2. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, 1997, pp. 8-22.
3. S. H. Woo, S. B. Yang, and S. D. Kim, "Task scheduling in distributed computing systems with a genetic algorithm," in *Proceedings of High Performance Computing on the Information Superhighway (HPC Asia '97)*, 1997, pp. 301-305.
4. T. D. Braun *et al.*, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," in *Proceedings of the 7th IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 330-335.
5. Y. K. Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," in *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, 1998, pp. 531-537.
6. A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra, "A unified resource scheduling framework for heterogeneous computing environments," in *Proceedings of the 8th Heterogeneous Computing Workshop*, 1999, pp. 156-165.
7. A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra, "A framework for mapping with resource co-allocation in heterogeneous computing systems," in *Proceedings of the 9th Heterogeneous Computing Workshop*, 2000, pp. 273-286.
8. H. Topcuoglu, S. Hariri, and M. Y. Wu "Task scheduling algorithms for heterogeneous processors," in *Proceedings of the 8th Heterogeneous Computing Workshop*, 1999, pp. 3-14.
9. Z. P. Lo and B. Bavarian, "Job scheduling on parallel machines using simulated annealing," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 1, 1991, pp. 391-396.
10. C. Y. Shen, Y. H. Pao, and P. Yip, "Scheduling multiple job problems with guided evolutionary simulated annealing approach," in *Proceedings of the 1st IEEE Con-*

ference on Evolutionary Computation, Vol. 2, 1994, pp. 702-706.

11. I. Ahmad, Y. K. Kwok, and M. Y. Wu, "Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors," in *Proceedings of the 2nd International Symposium on Parallel Architectures, Algorithms, and Networks*, 1996, pp. 207-213.
12. E. S. H. Hou, N. Ansari, and R. Hong, "A genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, 1994, pp. 113-120.



Po-Jen Chuang (莊博任) received the B.S. degree from National Chiao Tung University, Taiwan, R.O.C., in 1978, the M.S. degree in Computer Science from the University of Missouri at Columbia, U.S.A., in 1988, and the Ph.D. degree in Computer Science from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, U.S.A. (now the University of Louisiana at Lafayette), in 1992. Since 1992, he has been with Tamkang University, Taiwan, where he is currently a professor in the Department of Electrical Engineering. He was the department chairman from 1996 to 2000. His main areas of interest include parallel and distributed processing, fault-tolerant computing, computer architecture, and mobile computing. Dr. Chuang is a member of the IEEE, the IEEE Computer Society, the ACM, and the IICM.



Chia-Hsin Wei (魏家鑫) received the M.S. degree in Electrical Engineering from Tamkang University, Taiwan, in 2001. He is currently with ARinfotek Inc., Taiwan. He involves the development of embedded computers for industry control. His research interests include optimized algorithms, parallel and distributed processing, and computer architecture.



Yu-Shian Chiu (邱育賢) received the B.S. and M.S. degrees in Electrical Engineering in 2001 and 2003 from Tamkang University, Taiwan, where he is currently pursuing the Ph.D. degree. His research interests include parallel and distributed processing, computer architecture, and mobile computing.