Available online at www.sciencedirect.com

SCIENCE DIRECT®

PERGAMON

Engineering Applications of Artificial Intelligence 15 (2002) 491–500

Engineering Applications of ARTIFICIAL INTELLIGENCE

www.elsevier.com/locate/engappai

A simulated evolution approach to task matching and scheduling in heterogeneous computing environments

Hassan Barada[a,*], Sadiq M. Sait[b], Naved Baig[b]

[a] *Computer Engineering, Etisalat College of Engineering, Emirates Telecommunications Co., P.O. Box 980, Sharjah, United Arab Emirates*
[b] *Computer Engineering Department, King Fahd University of Petroleum and Engineering, Dhahran, Saudi Arabia*

Abstract

This paper applies a simulated evolution (SE) approach to the problem of matching and scheduling dependent tasks in a heterogeneous suite of computers interconnected via a high-speed network. The various steps of the SE approach are discussed in details. *Goodness* functions required by SE are designed and explained. Experimental results applied on various types of workloads are analyzed. Workloads are characterized according to the connectivity, heterogeneity, and communication-to-cost ratio of the task graphs representing the application tasks. The performance of SE is compared with a genetic algorithm approach for the same problem with respect to the quality of solutions generated, and timing requirements of the algorithms.
© 2003 Elsevier Science Ltd. All rights reserved.

Keywords: Scheduling; Optimization; Heuristics; Heterogeneous systems

1. Introduction

Many scientific applications today can be decomposed into tasks that require different types of computations. These computations maybe classified as single instruction-multiple data (SIMD) computations, multiple instruction-multiple data (MIMD) computations, or special-purpose computations such as fast fourier transform (FFT), searching, sorting, to name a few. Single computer architecture cannot usually match equally well with all of these types of computations. This fact has led many researchers to look for alternative computer architectures such as heterogeneous computing systems (HCS). An HCS consists of a number of machines that have different architectures, organizations and sizes, interconnected via a high-speed network. heterogeneous computing (HC) is then defined as the well-orchestrated use of an HCS. HC is emerging as a major paradigm for scientific and high performance applications to exploit the heterogeneity in computations (Khokar et al., 1993).

However, in order to run an application efficiently in an HCS, it is first decomposed into coarse-grained (large-size) tasks; each task is computationally homogeneous and well suited to a single machine in HCS. The characteristics of tasks and machines are determined using *code-profiling* and *analytical benchmarking* which are both analytical steps that evaluate the matching of tasks to individual machines (Chen and Tsai, 1985). Each task is then *matched* to the most suitable machine and *scheduled* on it. Matching is defined as the assignment of tasks to machines in the HCS whereas scheduling comprises the ordering of execution of tasks on each machine. The goal of efficient HC is then to achieve the minimal execution time of the application when it is running using the available HCS (Khokar et al., 1993).

Task matching and scheduling in HC environments is known to be NP-complete (Braun and Siegel, 1999). Therefore, various heuristics that attempt to reach near-optimal solutions to this problem have been proposed in the literature (Wang et al., 1997; Flan et al., 1997; Braun and Siegel, 1999; Topcuoglu et al., 1999; Radulescu and van Gemund, 2000). A good survey of these heuristics is detailed in Braun and Siegel (1999). Different from the approaches discussed in the literature, this paper proposes a *simulated evolution* (SE) approach for solving

*Corresponding author. Tel.: +971-6-504-3545; fax: +971-6-561-1789.
E-mail addresses: hbarada@ece.ac.ae (H. Barada), sadiq@ccse.kfupm.edu.sa (S.M. Sait), naved_baig@hotmail.com (N. Baig).

0952-1976/02/$ - see front matter © 2003 Elsevier Science Ltd. All rights reserved.
doi:10.1016/S0952-1976(02)00088-X

this problem. SE is a powerful general iterative heuristic that has been applied to few optimization problems especially in VLSI design automation (Kling et al., 1987; Youssef and Sait, 1999; Sait et al., 1999).

In Braun and Siegel (1999), the authors analyze 11 heuristics to the task matching and scheduling problem. They concluded that a genetic algorithm (GA) approach "was the best heuristic for most cases" for this problem. In addition to applying an SE approach to this problem, this paper compares the performance of SE to a GA approach (Wang et al., 1997), with respect to the quality of solutions and run time requirement of the algorithms.

The rest of the paper is organized as follows. Section 2 defines the HC model assumed in this work. In Section 3, we introduce briefly SE. Section 4 describes the implementation of SE to the task matching and scheduling problem. In Section 5, we present and analyze experimental results including a comparison between SE and GA as implemented on this problem. Section 6 is a conclusion.

2. Problem definition

Different HC models have been used in the literature. The HC model used in this work is similar to the model assumed in Wang et al. (1997) and Braun and Siegel (1999) where the authors applied 11 heuristics to the task matching and scheduling problem.

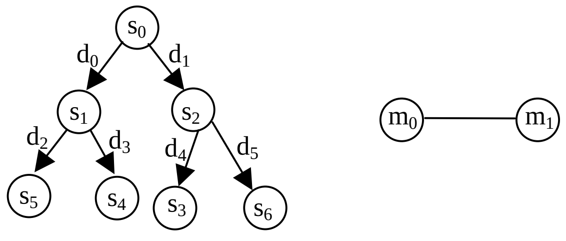
An application is decomposed into a set of coarse-grained tasks $\mathrm{Sb}=\{s_i, 0\leqslant i<k\}$, each task is computationally homogeneous and well suited to a single machine architecture. The application is usually described by a directed acyclic graph (DAG) where each vertex corresponds to a task and the directed edges between vertices correspond to dependencies between the tasks. The data items that need to be transferred between the tasks form a set $D=\{d_i, 0\leqslant i<p\}$. An HCS consists of a set of machines $M=\{m_i, 0\leqslant i<l\}$ each of which is characterized by a specific architecture such as SIMD, MIMD, special-purpose FFT, etc. An HCS is described by an undirected graph where vertices correspond to machines and edges correspond to paths between machines. For the purpose of this study, it is assumed that machines are fully connected.

The estimated execution times of the tasks in Sb on the machines in M are known a priori and are given by an $l\times k$ execution-time matrix E, where $E(i,j)$ is the estimate execution time of task s_j on machine m_i. The estimated transfer times of data items between tasks to be communicated through the HCS network are described by an $l(l-1)/2\times p$ transfer-time matrix Tr; p represents the number of data items while $l(l-1)/2$ represents the number of machine pairs in HCS. Approaches for finding these estimated times using code-profiling and analytical benchmarking are surveyed in Chen and Tsai (1985). As an example, Fig. 1a shows a DAG of 7 tasks and 6 data items describing a scientific application. Fig. 1b shows a graph representing a two-machine HCS on which the application is to be executed. The estimation of execution times of the tasks on both machines are given by the 2×7 E matrix shown in Fig. 1c. In Fig. 1d, the 1×6 Tr matrix gives the transfer times between m_0 and m_1 for the data items d_0 to d_5.

The task matching and scheduling problem is then defined as follows. Given is a set of k tasks Sb and a DAG representing the application, a set of l machines in HCS, an execution-time matrix E, and a transfer-time matrix Tr. It is required to match and schedule the k tasks in Sb on the l machines in M such that the total execution time of the application task on the HCS is minimized.

3. Simulated evolution

Kling and Banerjee (1987) proposed SE as a general iterative heuristic for solving combinatorial optimization problems. Since then, it has been used in the



(a) (b)

$$E=\begin{bmatrix}770 & 120 & 85 & 425 & 920 & 730 & 880\\ 790 & 730 & 60 & 375 & 910 & 650 & 800\end{bmatrix}$$

(c)

$$Tr=\begin{bmatrix}78 & 15 & 7 & 35 & 90 & 73\end{bmatrix}$$

(d)

Fig. 1. A sample HC model.

optimization of a number of problems in several areas of engineering (Lin et al., 1989; Ly and Mowchenko, 1993; Wang et al., 1993; Chen et al., 1995; Rao and Ramasubrahmanyan, 1996; Sait et al., 1999). The book by Youssef and Sait (1999) includes a good and thorough discussion of SE and its relation to other iterative heuristics such as Simulated Annealing (SA) and GA.

Sait et al. (1999) fuzzified SE for multi-objective VLSI cell placement problem. Ly and Mowchenko (1993) applied SE to the scheduling problem in high-level synthesis. Kuo et al. (1992) proposed a system for the automatic synthesis of asynchronous pipelines and used SE and some other techniques to realize the data path synthesis. Rao and Ramasubrahmanyan (1996) applied SE to optimize the coefficients of an FIR digital filter. In Lin et al. (1989) and Wang et al. (1993), SE has been used for solving VLSI routing problems. The global convergence of the SE algorithm using ergodic Markov chain is established by Mao et al. (1994).

SE starts from an initial solution to the given problem. Then following an evolution-based approach, it seeks to reach better solutions from one generation to the next. The algorithm has three basic steps: *Evaluation, Selection* and *Allocation*. These steps are carried out repetitively until a stopping criterion is satisfied.

The *Evaluation* step consists of evaluating the *goodness* of each *individual* e_i of the current solution. Individuals are components of the solution; they are the movable elements. The goodness measure must be highly correlated with the overall objective of the specific problem. It is a number expressible in the range $[0, 1]$ and it is defined, for each individual e_i, as $g_i = O_i/C_i$, where O_i is an estimate of the optimal cost of individual e_i and C_i is the actual cost of e_i in its current solution. O_i does not change from one generation to the next, and therefore it is computed for every individual only once during initialization.

One of the major difficulties in SE is the estimation of the optimal cost O_i required for g_i. O_i is problem specific and cannot be easily estimated for most combinatorial optimization problems. It is worth noting here that the goodness measure is defined for an individual. It is not the cost of the whole solution. In this paper, we propose two goodness measures specifically related to the task matching and scheduling problem in HCS.

During *selection* step, individuals of the current solution are divided into two disjoint sets; a selection set S and a set R of remaining individuals. The decision whether to include e_i in S or R is based solely on the individual goodness g_i. Selected individuals, i.e. individuals in set S, are passed to the next step for possible relocation. Usually, it is more likely to improve the current solution if individuals with lower goodness measures are relocated from their current *location*. Hence, individuals with lower goodness values are more likely to get selected and assigned to set S. However, individuals with higher goodness values should also have a non-zero probability of being selected.

```
ALGORITHM   Simulated_Evolution

/* Initialization */
For all $e_i$ calculate $O_i$;
Generate an initial solution sol;

Repeat

        /* Apply evaluation, selection and allocation on sol */

        /* Evaluation */
        For all $e_i$ Do $g_i = O_i / C_i$;

        /* Selection */
        For all $e_i$ Do
            If Selected ($e_i$, B) Then $S = S \cup \{e_i\}$
                                   Else $R = R \cup \{e_i\}$
        Sort the elements of S;

        /*Allocation */
        For all $e_i \in S$ Do Reallocate (ei);
        Generate new sol;

Until Stopping_Criteria are met;

Return (sol);

End Simulated_Evolution;
```

Fig. 2. Outline of SE algorithm.

Allocation relocates all individuals in the selected set S of the current generation to produce a new generation. It involves making several trial alterations for each individual before deciding on the final configuration of the new generation. The goal of allocation is to favor improvements of quality of the current generation over the previous generation, without being too greedy. It allows the search to progressively converge to an optimal configuration where individuals are optimally located. The choice of a suitable allocation function is problem specific. A general outline of SE is given in Fig. 2.

4. SE for matching and scheduling in HCS

To implement SE, it is necessary first to devise an efficient goodness measure that computes goodness values of each individual during the evaluation step of the algorithm. It is also necessary to develop an encoding technique that describes the state of the solution. Then, problem-dependent methods for generation of an initial solution, evaluation, selection, and allocation should be designed. Details of all components needed for the implementation of an SE-based approach for matching and scheduling in HCS (MSHC) are discussed next.

4.1. Encoding technique

A solution to MSHC is encoded using a string of k segments where k is the number of tasks in the DAG representing the application to be executed in HCS. Each segment consists of two parts: a task identifier and a machine identifier. Pairing a task s_i with a machine m_j in the same segment means that s_i is assigned to machine m_j. Obviously, in this type of encoding, we have to ensure that a string represents a valid solution that satisfies precedence constraints in the DAG. In our encoding scheme, if a task s_x is placed on the left of a task s_y and both are assigned to the same machine m_j then s_x is executed before s_y in m_j. Therefore, if in the DAG a task s_y depends on a task s_x, s_x is always placed to the left of s_y in a valid string.

Fig. 3 illustrates this encoding scheme. The string in the figure represents a valid solution to the HC problem described in Fig. 1. In Fig. 3, the order of execution of tasks in each machine is given by m_0: s_0, s_3, s_4 and m_1: s_1, s_2, s_5, s_6. This ordering means that in machine m_0, s_0 is executed first, s_3 is executed next, then s_4 last. In m_1, s_1 is executed first, s_2 second, s_5 third, then s_6 last.

4.2. Generating an initial solution

To generate a valid initial solution, each task in the DAG is first assigned randomly to a machine in the set of machines M. Then, the DAG is topologically sorted [20]. Following the sorted order, which guarantees that data dependencies are satisfied, the tasks are placed in successive segments from left to right. This initial valid string is then modified a random number of times as follows. A task s_i is selected and moved randomly to another segment in the string within its valid range of positions. The valid range of a task is the set of positions (segments) where the task can be placed without violating any data dependencies.

4.3. Evaluation

As discussed in Section 3, the evaluation step computes the goodness measure $g_i = O_i/C_i$ of each individual e_i of the current solution. In the SE-implementation of MSHC, an individual e_i represents a task s_i. The *location* of an individual refers to the pairing of s_i with m_j (the assignment of task s_i to machine m_j) and the position of this pair in the string (the order in which s_i is executed in machine m_j relative to all tasks assigned to m_j). C_i is defined as the finishing time of task s_i given the assignment of tasks and the order of their execution as described in the current solution. O_i is defined as the finishing time of task s_i if it is placed in its *optimal location* according to a defined function F. In our research, we have developed two goodness measures, details of which are described below.

4.3.1. Goodness measure 1

In this proposed goodness measure, F assigns task s_i and all its predecessors to their best-matching machine with respect to the execution time of the tasks on all machines. When O_i is calculated, we assume that s_i and its predecessors are the only tasks in the DAG representing the application. It is noted that O_i is only calculated once at the initialization step of the algorithm.

For example, for the DAG in Fig. 1, the optimal finishing time of task s_4, according to the above function F, is its finishing time if it is assigned to machine m_1 and tasks s_0 and s_1 are both assigned to machine m_0. In this case, $O_4 = E(0,0) + E(0,1) + \mathrm{Tr}(0,3) + E(1,4) = 770 + 120 + 35 + 910 = 1835$ units, where E is the execution-time matrix shown in Fig. 1c and Tr is the transfer-time matrix shown in Fig. 1d. O_4 is computed before SE starts. Assuming that after SE starts, the current solution is the solution shown in Fig. 2, $C_4 = E(0,0) + \mathrm{Tr}(0,0) + E(1,1) + \mathrm{Tr}(0,1) + E(1,2) + \mathrm{Tr}(0,4) + E(0,3) + \mathrm{Tr}(0,3) + E(0,4) = 3123$ units since s_0, s_3 and s_4 are assigned to m_0, and s_1 and s_2 are assigned to m_1. Therefore g_4 for the solution in Fig. 2 using the *goodness measure 1* is equal to $1835/3123 = 0.588$.

4.3.2. Goodness measure 2

In using this goodness measure, F assigns task s_i and all its predecessors to the best-matching machine of s_i with respect to the execution time. In this case, all communication costs associated with s_i and its predecessors become negligible as all of them are allocated to the same machine. As in goodness measure 1, when O_i is calculated, we assume that s_i and its predecessors are the only tasks in the DAG representing the application.

For example, for the DAG in Fig. 1, the optimal finishing time of task s_4, according to the above function

s_0 m_0	s_1 m_1	s_2 m_1	s_5 m_1	s_6 m_1	s_3 m_0	s_4 m_0

Fig. 3. A valid solution string.

F, is its finishing time if it is assigned to machine m_1 and tasks s_0 and s_1 are also assigned to machine m_1. In this case, $O_4 = E(1,0) + E(1,1) + E(1,4) = 2430$ units. Therefore g_4 for the solution in Fig. 2 using *goodness measure 2* is equal to $2430/3123 = 0.778$.

4.4. Selection

During the selection step, at every generation of the algorithm and for each task s_i, a random number in the range $[0,1]$ is generated and compared with $(g_i + B)$, where g_i is the goodness measure of s_i as defined above and B is the *selection bias*. If the generated number is greater than $(g_i + B)$, then s_i is selected and assigned to the selection set S; otherwise it is assigned to R. The selected tasks are ordered in ascending order according to their level in the DAG, and are considered for allocation in the next step in that particular order.

The value of the selection bias B is fixed and preset at the beginning of the algorithm. It is used to have some control over the selection process as a tradeoff between implementing a fast algorithm and having a more thorough search. In our work, we have used negative values for B (between -0.1 and -0.3) for small problem sizes (small number of tasks in DAG) and positive values (between 0 and 0.1) for large problem sizes. For smaller DAGs, a negative value of B will force the selection of relatively large percentage of tasks, hence allowing for a more thorough search. Since the number of tasks is small, the time requirement of the algorithm is not affected. In cases of large DAGs, the value of B is kept positive to restrict the number of tasks selected in the selection step. This will keep the time requirement of the SE algorithm under control, especially during the allocation step, which is the most time consuming step of the algorithm.

4.5. Allocation

The allocation step relocates all individuals in the selected set S. The strategy used in the SE algorithm for MSHC is constructive. It always chooses the *best location* for the task under consideration. The best location in this case is the location (as defined in Section 4.3) of the task that gives the best valid solution.

First, the valid moving range of the selected task is determined. Next, all various combinations are tried and the schedule lengths of corresponding solutions are computed. A combination consists of placing the task in a valid segment without violating the data dependency constraints and assigning it to a machine. Finally, the task is placed in the segment and assigned to the machine that results in the best overall schedule length. This process is tried for every task in the set S and the output of the allocation step is the next generation.

One important parameter, which has been analyzed and studied in our SE-implementation, is a parameter we call Y. Y defines the number of machines to which a particular task can be assigned, according to its execution time on all machines in HCS. For example, if Y is set to 1, each task in the selected set S can be assigned to only one machine; its best-matching (least execution time) machine. If $Y = 2$, then each task can be assigned to two machines, its best-matching machine and its next best. This parameter limits the number of combinations tried in the allocation step and therefore allows controlling the tradeoff between the time requirement of the SE algorithm and the quality of the solution generated.

4.6. Stopping criteria

In our experiments, we have used a fixed number of iterations (generations) as the stopping criterion. We have experimented with different values of iterations and found out that for all workloads tried, SE for MSHC converges with less than 1000 iterations.

5. Experimental results

To analyze the performance of the SE-based approach, randomly generated workloads are used. Each workload corresponds to a DAG representing an application task, the number of machines in HCS, the execution-time matrix E, and the transfer-time matrix Tr. Randomly generated workloads are used since a generally accepted set of HC benchmarks does not exist (Wang et al., 1997) and it is also desirable to obtain data that demonstrates the effectiveness of the approach over a broad range of conditions. We have conducted experiments on small (10 tasks and 3 machines) and large (up to 100 tasks and 20 machines) workloads.

Workloads are further classified according to their connectivity, heterogeneity and communication-to-cost ratio (CCR). The level of connectivity in a DAG defines the number of data items to be transferred between the tasks. The number of data items is set as $|\mathrm{Sb1}| \leqslant |D| \leqslant |\mathrm{Sb}||\mathrm{Sb1}|/2$. The lower bound is necessary for the DAG to be connected while the upper bound is necessary to generate acyclic DAG. This parameter classifies workloads as low connectivity (bottom $\frac{1}{3}$ of the specified range), moderate connectivity (middle $\frac{1}{3}$), and high connectivity (top $\frac{1}{3}$).

Heterogeneity classifies workloads according to the degree of heterogeneity of tasks. The degree of heterogeneity defines the difference in execution times of tasks on the different machines in the HC system. Three levels of heterogeneity are used: low heterogeneity which means that the execution times of a task on the various machines vary within $\pm 10\%$ of a base value, moderate heterogeneity (within $\pm 30\%$), and high heterogeneity (within $\pm 50\%$).

CCR is defined as the ratio of size of data item over execution time of the task generating this item. CCR values of 0.1, 0.5 and 1 are used. CCR = 0.1 indicates that the communication cost is low compared to the computation cost in the DAG. This describes lightly communicated tasks. CCR = 1 indicates that the communication cost is comparable to the computation cost. This describes heavily communicated tasks.

5.1. Choosing goodness measure

This set of experiments is carried out to compare the two proposed goodness measures with respect to the quality of solutions and then to choose the best measure in the subsequent experiments. Fig. 4 compares the performance of the two goodness measures when they are applied to a workload of 100 tasks and a network of 20 machines. In Fig. 4a, the characteristics of the workload are: low connectivity, low heterogeneity and

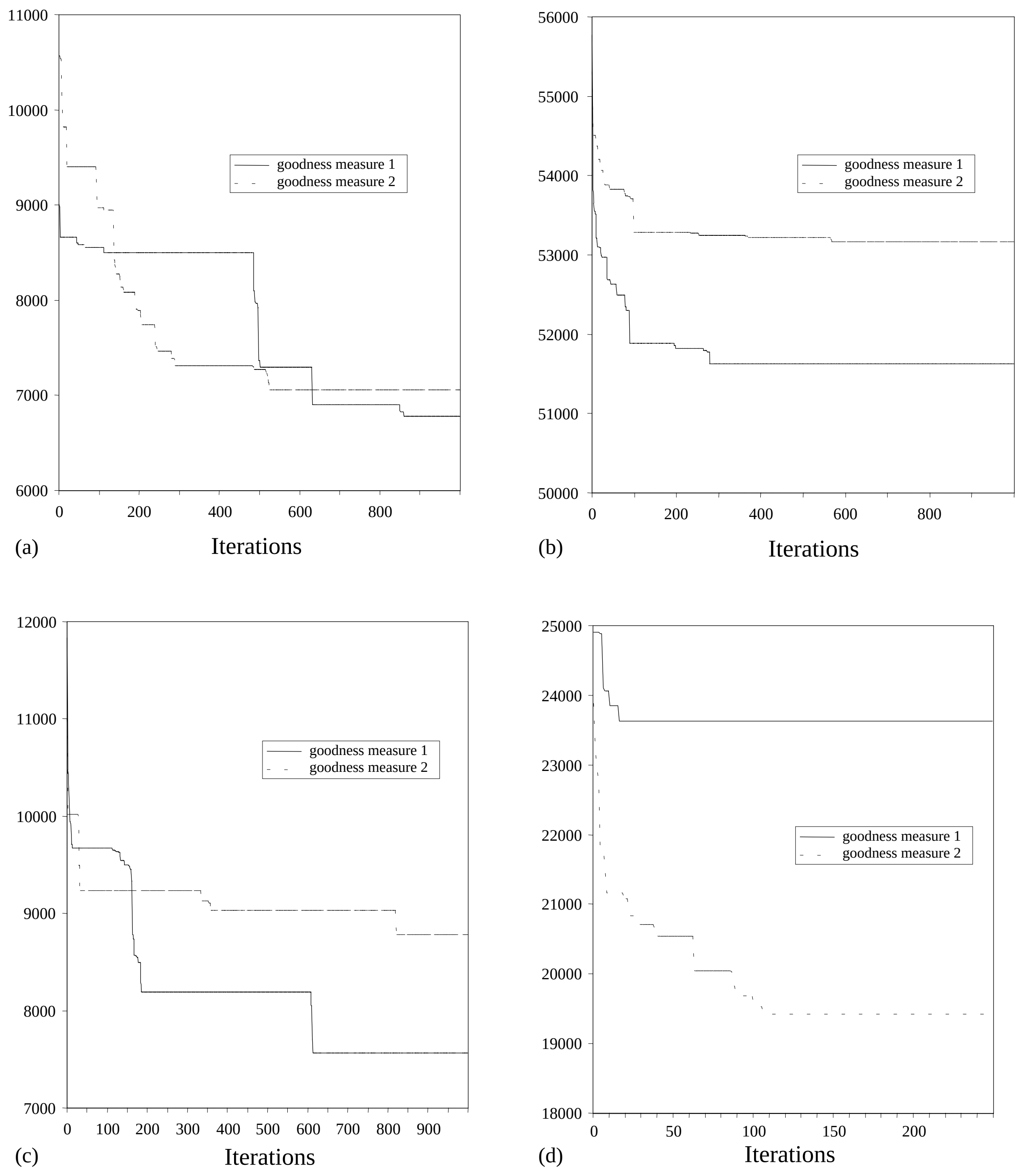


Fig. 4. Schedule length versus iteration for the two goodness measures: (a) workload with low connectivity, low heterogeneity, CCR = 0.1; (b) workload with high connectivity; (c) workload with high heterogeneity; and (d) workload with CCR = 1.

CCR = 0.1. Here, *goodness measure* 1 outperforms *goodness measure* 2 during later iterations. Fig. 4b plots the two goodness measures when SE is applied to a workload of high connectivity. Here again, goodness measure 1 outperforms goodness measure 2. Fig. 4c compares the two goodness measures when SE is applied to a workload of high heterogeneity. Goodness measure 1 again outperforms goodness measure 2. This can be explained in that the latter allocates the predecessor tasks to the same machine on which the task, whose goodness is being computed, is allocated. Hence, these tasks may be allocated to poorly matched machines. On the other hand, goodness measure 1 allocates the predecessor tasks to their best machines along with that of the task whose goodness is to be computed. For cases of low and moderate heterogeneity, the two measures return comparable results.

Fig. 4d depicts the performance of the goodness measures when applied to a workload with CCR = 1. In this case, goodness measure 2 outperforms goodness measure 1. This is because goodness measure 2 nullifies the effect of the data transfers by placing all the predecessor tasks on the same machine on which the task whose goodness is being computed is allocated. This makes the data transfer times negligible and hence goodness measure 2 performs better than goodness measure 1 for high CCR values. High CCR values mean that the communication times are comparable with the execution times.

From this set of experiments, it is concluded that in general goodness measure 1 performs better than goodness measure 2. However, goodness measure 2 outperforms goodness measure 1 when applied to workloads with high CCR values. For the rest of the experiments, we combined the two goodness measures by selecting the lower bound of the optimal finish time of individual tasks calculated by the two measures. By doing this, we make the SE algorithm independent of the workload used.

5.2. Effectiveness of SE for MSHC

To verify the effectiveness of the SE algorithm, one can monitor the number of selected individuals in the selection step of the algorithm as SE progresses. Initially, a large number of individuals should be selected for relocation since most individuals are not optimally placed. However, in later iterations, the number of selected individuals should decrease gradually since more individuals are placed optimally and should not be relocated.

In this set of experiments, various sizes and types of workload were generated randomly and the number of selected tasks at every iteration of the SE algorithm was logged. Fig. 5a shows the result of a sample of these experiments. The figure sketches the number of selected tasks versus the iteration number for a workload of large size and high connectivity. The shape of the graph, which is a typical shape of all the experiments we have run, illustrates clearly that the SE algorithm for MSHC is very effective in optimally placing the tasks in their best locations. Initially, a large number of tasks are selected, as they are not optimally placed. As SE progresses, more and more tasks are allocated to their optimal location, hence reducing the number of selected tasks in later iterations. Fig. 5b shows the corresponding current schedule length for the same experiment. This figure demonstrates that the algorithm converges after the first few hundreds iterations.

It should be noted that for most of the experiments we run, SE converges quickly. The algorithm is run for 1000 iterations but major improvement is obtained within the first couple of hundred iterations. However, SE incurs high time requirement per iteration during the initial

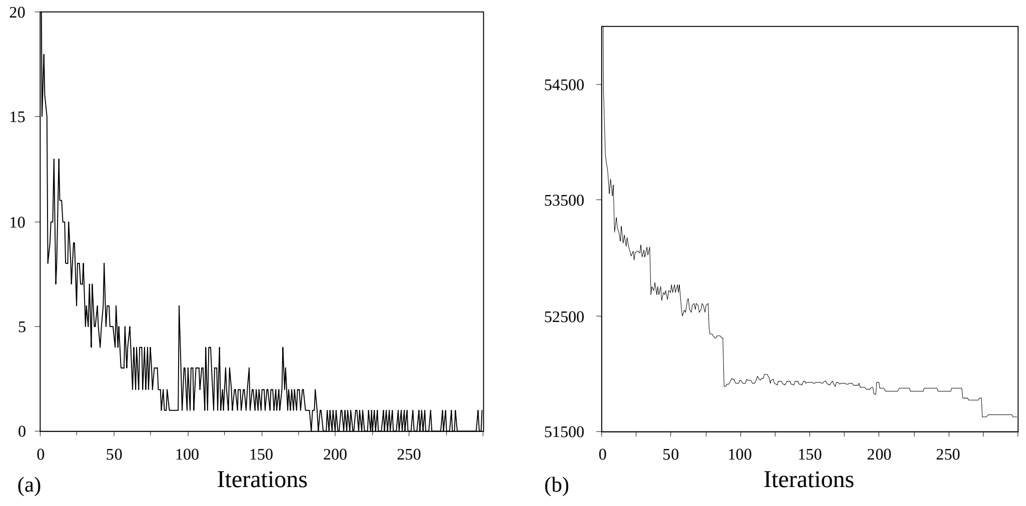


Fig. 5. Effectiveness of SE for MSHC: (a) *number of selected tasks* versus *iteration* and (b) *schedule length* of current solution versus *iteration*.

iterations when the number of selected tasks for allocation is relatively large. The high execution time of SE is due to the constructive allocation phase of the algorithm, which involves multiple trial moves within a single iteration. The time requirement per iteration in later iterations is reduced due to the low number of selected tasks for relocation.

5.3. Effect of *Y* parameter

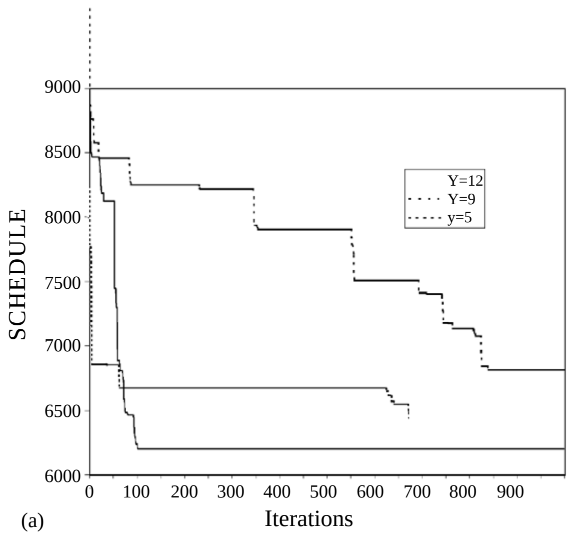
Y defines the number of machines to which a particular task can be allocated according to the ascending order of its execution times on all machines in HCS. It represents a tradeoff between the timing requirement of the algorithm and the quality of solution to the problem. It should be expected that higher *Y* generate better quality solutions. However, this parameter was studied because we anticipated that it could be set according to the heterogeneity of workloads and helps in reducing the timing requirement of the SE algorithm without jeopardizing the quality of solutions. It was predicted that if the workload is highly heterogeneous, a smaller *Y* will still give high quality solutions but with less time.

To analyze the effect of *Y* and heterogeneity of the workload, SE for MSHC was run on workloads of different sizes and varying levels of heterogeneity. Different values of *Y* (from 2 to the number of machines in HCS) were tried out and it was observed, as expected, that the timing requirements for the SE algorithm increase as *Y* increases. However, as far as the quality of solution is concerned, it was not as clear. For low heterogeneous (almost homogeneous) workloads, increasing *Y* almost always improved the quality of solution when running the algorithm for 1000 iterations. For workloads of medium and high heterogeneity, it was observed that increasing *Y* only improved the solution as long as *Y* was relatively small compared to the number of machines in the workload. Beyond a certain threshold, the quality of solutions actually started to get worse. Again, 1000 iterations were used in the algorithm. This can be explained that in heterogeneous workloads and for large *Y*, many low quality solutions have to be visited before reaching good solutions. Therefore, more iterations (probably more than 1000 iterations) are needed to reach a good solution.

Figs. 6a and b show typical samples of these experiments. In both experiments, the workload is of large size (100 tasks and 20 machines). Fig. 6a sketches the quality of solutions for the first 1000 iterations for *Y* equals 5, 9, and 12 if the heterogeneity of the workload is low. This figure clearly shows that as *Y* increases the quality of solution improves and the rate to reach good solutions also improves. However, in Fig. 6b where a highly heterogeneous workload was used, the best result was for $Y = 9$. Increasing *Y* beyond 9 actually made the quality of solutions worse after the first 1000 iterations. Therefore, for highly heterogeneous workloads, *Y* should be kept small relative to the number of machines. This should result in reaching better solutions with less time. The magic number that was satisfied in a larger number of experiments that we have conducted is *Y* being about 40% of the number of machines in the workload.

5.4. Comparison of SE and GA for MSHC

We have used the same HC model adopted by Wang et al. (1997) so that the performance of SE can be



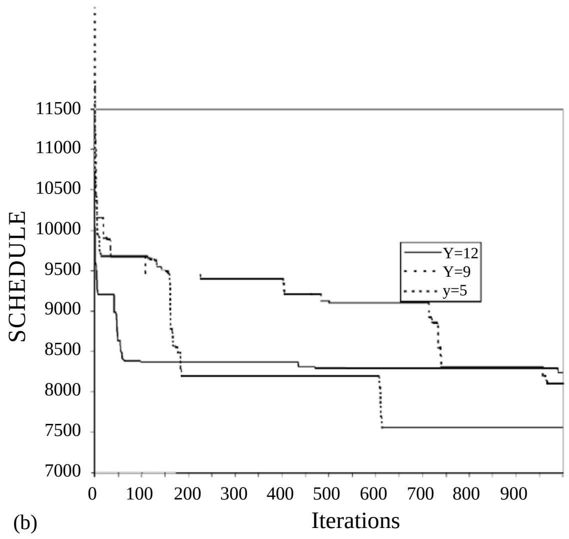


Fig. 6. Effect of parameter *Y* on schedule length: (a) workload with low heterogeneity and (b) workload with high heterogeneity.

Table 1
Comparison of GA and SE with respect to quality of solution

Size of workload	Type of workload	Schedule length GA	Schedule length SE
10 tasks, 3 machines	Low C, low H, CCR = 0.1	2026	2026
	High C, low H, CCR = 0.1	4538	4538
	Low C, high H, CCR = 0.1	1786	1786
	Low C, low H, CCR = 1	1915	2180
50 tasks, 10 machines	Low C, low H, CCR = 0.1	3535	3313
	High C, low H, CCR = 0.1	22,761	22,838
	Low C, high H, CCR = 0.1	3108	2461
	Low C, low H, CCR = 1	9281	10,857
100 tasks, 20 machines	Low C, low H, CCR = 0.1	6868	6441
	High C, low H, CCR = 0.1	50,778	51,630
	Low C, high H, CCR = 0.1	8894	7565
	Low C, low H, CCR = 1	20,726	19,420

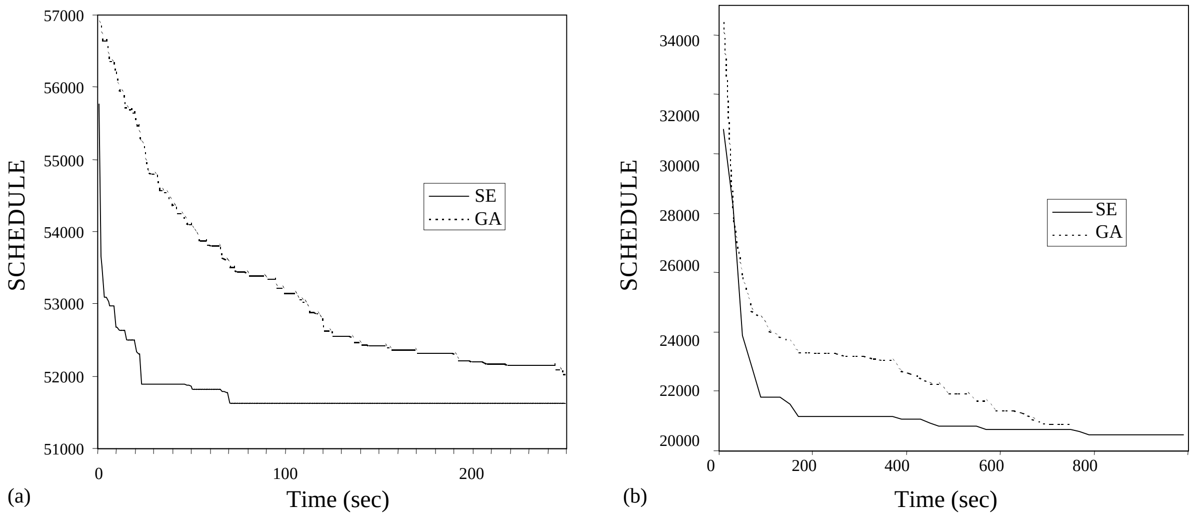


Fig. 7. Schedule length versus real time for SE and GA: (a) workload with high connectivity and (b) workload with CCR = 1.

compared to GA with respect to the quality of solutions and the timing requirements of these algorithms. GA was implemented using the algorithm in (Wang et al., 1997). We have also used workloads with different characteristics (connectivity, heterogeneity and CCR) as compared to workloads used in (Wang et al., 1997) that only considered the connectivity parameter.

It should be clear that the final solution generated by either algorithm cannot be compared to the optimal solution since exhaustive search cannot be applied to even small workloads. For example, exhaustive search was applied to a DAG of 10 tasks and an HCS of 3 machines and it took almost a day on a Sun Ultra Sparc workstation for the algorithm to visit all solutions and return the best schedule. Therefore, both algorithms are only compared to each other.

The two algorithms are compared on the basis of the quality of solutions and their convergence rate. Table 1 records the comparison of the results obtained by GA and SE with respect to the quality of solutions. It can be seen that for smaller workloads, both the algorithms return the same result. For larger workloads, SE returns significantly better results for workloads having low connectivity and low CCR or for workloads having high heterogeneity. For workloads having high connectivity or high CCR, the results are not so clear with GA outperforming SE in some cases. This is due to the fact that the goodness measures used in SE are primarily based on the matching characteristic of the tasks.

Fig. 7 represent sample results of the experiments that we have run to compare both heuristics with respect to their convergence rate. Fig. 7a shows the best schedules found by both algorithms as real time increases for a workload of 100 tasks and 20 machines with high connectivity. Fig. 7b shows the best schedules found by both algorithms as time increases for a workload of 100 tasks and 20 machines with CCR = 1. From these samples and other experiments we have conducted, it was clear that SE produced better solutions than GA with less time, for different workloads. However, as time increases SE and GA solutions were getting closer to each other.

6. Conclusion

To the best of our knowledge, simulated evolution was never applied before to the task matching and scheduling problem in heterogeneous computing environments. In this paper, we have proposed an SE-based task matching and scheduling scheme. Experiments were conducted on various types and sizes of workloads to demonstrate the effectiveness of the algorithm. Two goodness measures are proposed and their performance compared on various types of workloads. We have also compared our approach to the genetic algorithm approach proposed in (Wang et al., 1997). SE performed better than GA for workloads of certain characteristics as it generates better quality solutions with less time. For some workloads characteristics, the difference between the two algorithms was not clear.

Acknowledgements

The authors thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia and Emirates Telecommunications Corporation, Sharjah, UAE, for their support.

References

Braun, T.D., Siegel, H.J., et al., 1999. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. Proceedings of 8th Heterogeneous Computing Workshop, Puerto Rico, pp. 15–29.

Chen, S., Tsai, W., 1985. A graph matching approach to optimal task assignment in distributed computing system using a minimax criterion. IEEE Transactions on Computers C-34 (3), 197–203.

Chen, Ching-Dong, Yu-Sheng Lee, Wu, A.C.-H., Young-Long Lin, 1995. TRACER-fpga: a Router for RAM-based FPGAs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 14 (3), 371–374.

Flan, N.S., Freund, R.F., Shroff, P., Watson, D.W., 1997. Genetic Simulated Annealing for Scheduling Data-Dependent Tasks in Heterogeneous Environments. Proceedings of 7th Heterogeneous Computing Workshop, Florida, USA, pp. 98–103.

Khokar, A., Prasana, V., Shaaban, M.E., Cho-Li Wang, 1993. Heterogeneous computing: challenges and opportunities. IEEE Computer 26 (6), 18–27.

Kling, R.M., Banerjee, P., 1987. ESP: a new standard cell placement package using simulated evolution. Proceedings of 24th Design Automation Conference, pp. 60–66.

Kuo, Yau-Hwang, Shaw-Pyng Lo, 1992. Automated Synthesis of Asynchronous Pipelines. IEEE International Symposium on Circuits and Systems 2, 685–688.

Lin, Y.L., Hsu, Y.C., Tsai, F.H., 1989. SILK: a simulated evolution router. IEEE Transactions on Computer-Aided Design 8 (10), 1108–1114.

Ly, T.A., Mowchenko, J.T., 1993. Applying simulated evolution to high level synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits 12 (3), 389–409.

Mao, Chi-Yu, Yu Hen Hu, 1994. Convergence analysis of simulated evolution algorithms. Fourth Great Lakes Symposium on VLSI, pp. 30–34.

Radulescu, A., van Gemund, A.J.C., 2000. Fast and effective task scheduling in heterogeneous systems. Proceedings of 9th Heterogeneous Computing Workshop, Mexico, pp. 229–238.

Rao, S.S., Ramasubrahmanyan, A., 1996. Design of discrete coefficient fir filters by simulated evolution. IEEE Signal Processing Letters 3 (5), 137–140.

Sait, S.M., Youssef, H., Ali, H, 1999. Fuzzy simulated evolution algorithm for multi-objective optimization of VLSI placement. Proceedings of the 1999 Congress on Evolutionary Computation, pp. 91–97.

Topcuoglu, H., Hariri, S., Wu, M.Y., 1999. Task scheduling algorithms for heterogeneous processors. Proceedings of 8th Heterogeneous Computing Workshop, Florida, USA, pp. 3–14.

Wang, L.Y., Liu, B.D., Lai, Y.T., Yeh, M.Y., 1993. Performance-driven global routing based on simulated evolution. Proceedings of Computer, Communication, Control and Power Engineering, pp. 511–514.

Wang, L., Siegel, H.J., Rowchoudhry, V.P., Maciejewski, A.A., 1997. Task matching and scheduling in heterogeneous computing environments using a genetic algorithm-based approach. Journal of Parallel and Distributed Computing 47 (8–22), 8–22.

Youssef, H., Sait, S., 1999. Iterative Algorithms and Their Applications in Engineering. IEEE Computer Society Press, California.