*Mehdi Akbari, Hassan Rashidi. An Efficient Algorithm for Compile-Time task scheduling problem on heterogeneous computing systems. International Journal of Academic Research PART A / 2nd; 2015; 7(1), 192-202. **DOI:** 10.7813/2075-4124.2015/7-1/A.45*

Library of Congress Classification: HF5001-6182

AN EFFICIENT ALGORITHM FOR COMPILE-TIME TASK SCHEDULING PROBLEM ON HETEROGENEOUS COMPUTING SYSTEMS

Mehdi Akbari, Hassan Rashidi

Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
Department of Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran

E-mails: mehdi_akbari@hotmail.com, Hrashi@atu.ac.ir

DOI: 10.7813/2075-4124.2015/7-1/A.45

Received: 22 Feb, 2015
Accepted: 25 Mar, 2015

ABSTRACT

An efficient assignment and scheduling of tasks is one of the key elements in effective utilization of heterogeneous multiprocessor systems. The task scheduling problem has been proven to be NP-hard is the reason why we used meta-heuristic (guided-random-search-based) methods for finding a suboptimal schedule. Due to the large solution space that a meta-heuristic algorithm is required to cover, the search generally incurs considerably much higher computational cost than the heuristic-based algorithms. Further, meta-heuristic algorithms such as GA typically require sufficient sampling of candidate solutions in the search space and have shown robust performance on a variety of scheduling problems. For these reasons, in this paper we proposed a task scheduling algorithm on heterogeneous computing systems using Efficient State Space Search Genetic Algorithm (ESSSGA). The basic idea of this approach is to exploit the advantages of heuristic-based algorithms to reduce space search and the time needed to find good solutions. The proposed algorithm uses a novel list scheduling heuristic-based algorithm while using a heuristic-based earliest finish time approach to search for a solution for the task-to-processor mapping. The achieved results show that the proposed approach significantly surpasses the other approaches in terms of task execution time (makespan).

Keywords: Genetic algorithms, Task scheduling, Heuristic algorithm, Multiprocessors, State-space search

1. INTRODUCTION

The main objective of the task scheduling is to map tasks onto processors and to order their execution so that precedence constraints are satisfied and minimum of task execution time (makespan) is achieved. Many task scheduling algorithms using a variety of approaches have been proposed. Due to the NP-complete nature of task scheduling [1], the efficiency of schedule solutions presented by existing scheduling algorithms cannot be guaranteed and it is considered one of the most challenging problems in heterogeneous distributed computing systems.

Traditional task scheduling has focused on heuristic-based scheduling such as list-based scheduling algorithms. In the classical list-based scheduling, such as Heterogeneous Earliest Finish Time (HEFT) [2], a task sequence satisfying priority constraints, the priorities are based on computation and communication costs. The purpose of this algorithm is typically to schedule all the tasks on a set of available processors in order to minimize makespan without violating precedence constraints [3].

In spite of the heuristic-based algorithms, the meta-heuristic-based algorithm or guided-random-search-based [2], such as GA, includes a method in the search space which is less efficient and produces much higher computational cost. For this reason, trade-off between task execution time and speed of convergence is required. Since hybrid algorithm achieves better performance than heuristic algorithms [3], we develop a hybrid algorithm by integrating GA with some heuristic approaches. In this paper, we described the task scheduling problem and proposed a hybrid task scheduling algorithm using Efficient State Space Search Genetic Algorithm (ESSSGA) on heterogeneous computing systems.

2. PROBLEM DESCRIPTION

Compile-Time Task scheduling is the problem of assigning the tasks of an application to the processors of heterogeneous computing systems. An efficient schedule is a schedule that minimizes the schedule length of the application. In task scheduling algorithms, the sequence of task execution and relations is represented by a directed acyclic graph (DAG), $G\ (T, E)$, where T is a set of n tasks and E is a set of e edges as shown in Figure 2. An $edge\ (i, j) \in E$ between task T_i and task T_j represents the communication cost between two tasks denoted as $C(T_i, T_j)$. The communication cost is only required when two tasks are assigned to different processors, i.e., the communication cost when they are assigned to the same processor will be zero. A task with no predecessors is called an $entry$ task, T_{entry} and a task with no successors is called an $exit$ task, T_{exit}. The weight on a task, T_i, signified as $W(T_i)$, represents the computation cost of the task, and the computation cost of a task, T_i on a processor, P_j is $W(T_i, P_j)$ [2].

A heterogeneous computing system is represented by a set P of m processors .The $n \times m$ computation cost matrix C stores the execution costs of tasks where each $c_{i,j} \in C$ represents the execution time of task T_i on processor

P_j. The average computation cost of task T_i on all processors, denoted as $\overline{W(T_i)}$ and $\overline{C(T_i,T_j)}$ is the average communication cost of the $edge(T_i,T_j)$ [2, 3]. Table 1 provides a list of notations and their definitions used in the paper.

2.1. Critical path (CP)

The longest path of a task graph is namely the Critical Path (CP) which is defined as the path from an entry task to an exit task that has maximum sum of computation costs of tasks and communication costs of edges [4]. The sum of computation costs of the tasks on the CP defines the lower bound of the makespan. Therefore, an efficient list-based scheduling algorithm requires suitable scheduling of the tasks located in the CP [5].

Table 1 Description of notations

Notation	Description
T	A set of tasks in an application
E	A set of edges for precedence constraints among the tasks
$G\ (T,E)$	A Directed Acyclic Graph
P	A set of heterogeneous processors
n	Number of tasks
e	Number of edges
m	Number of processors in a heterogeneous computing system
T_i	The ith task in the application
$edge(T_i,T_j)$	The edge from task T_i to task T_j
P_j	The jth processor in the system
T_{entry}	The entry task with no predecessor
T_{exit}	The exit task with no successor
$W(T_i,P_j)$	The computational cost of task T_i on processor P_j
$W(T_i)$	The computational cost of task T_i
$\overline{W(T_i)}$	The average computational cost of task T_i
C	The computation cost matrix
$c_{i,j}$	The execution time of task T_i on processor P_j
$C(T_i,T_j)$	The communication cost from subtask T_i to subtask T_i
$\overline{C(T_i,T_j)}$	The average communication cost of the $C(T_i,T_j)$
SLR	Scheduling length ratio of a task graph
CP	The longest path of a task graph
CP_{min}	The critical path of the unscheduled application based on the computation cost of tasks on the fastest processor
$FT(P_i)$	The finishing time of the scheduled exit node on processor P_i.
$PopSize$	The population size
$elitism$	A set of the best chromosomes
$offspring$	The chromosome generated by operators
Pc	The probability of crossover
Pm	The probability of mutation

2.2. Heterogeneity model

The heterogeneity model of a computing system used in this paper is Task-based Heterogeneity Model (THM). In a THM computing system, a processor executes the tasks at the different speed, depending on characteristics of the tasks.

3. RELATED WORK

Static task scheduling algorithms can be generally classified into two main types: heuristic-based algorithms and meta-heuristic algorithms (guided-random-algorithms) [2]. Heuristic-based Task scheduling classified as list-based, clustering-based, and duplication-based heuristics (Figure 1) [2, 4, 6].

A list-based scheduling preserves a list of all tasks in DAG. Each task is assigned a given priority and the order of the list of tasks is created based on these priorities. Two phases are then repeated until all the tasks in the ordered list are scheduled: task selection and processor selection. In the first phase, an ordered task sequence using its priority is generated and in the second phase, a set of appropriate candidate processors for a task is chosen. It is to be noted that any of the priority scheme may be used to serialize the tasks in the DAG. List scheduling algorithms are mostly chosen since they generate good quality schedules with less complexity but they are seriously dependent on the effectiveness of the heuristics. Hence, they are not likely to produce reliable results for task scheduling. List scheduling algorithms such as Modified Critical Path (MCP) [7], Dynamic Critical Path (DCP) [4], Dynamic Level Scheduling (DLS) [8], Levelized Min Time (LMT) [9], Mapping Heuristic (MH) [10], Heterogeneous Earliest Finish Time (HEFT) [2], and Critical Path On a Processor (CPOP) [2] are popular task scheduling algorithms for heterogeneous system.

A weakness of the list-based scheduling algorithms is that the performance of these algorithms is seriously dependent on the effectiveness of the heuristics [3]. Consequently, they are not probable to generate reasonable results for task scheduling. Many list-based scheduling works only if a predefined simplifying assumption is maintained [5]. Some assumption can be acceptable in specific cases, but others cannot be gratified in real-world applications. Homogeneous processors, unlimited number of processors, and no precedence constraints among tasks, are examples of the simplifying assumptions [5].

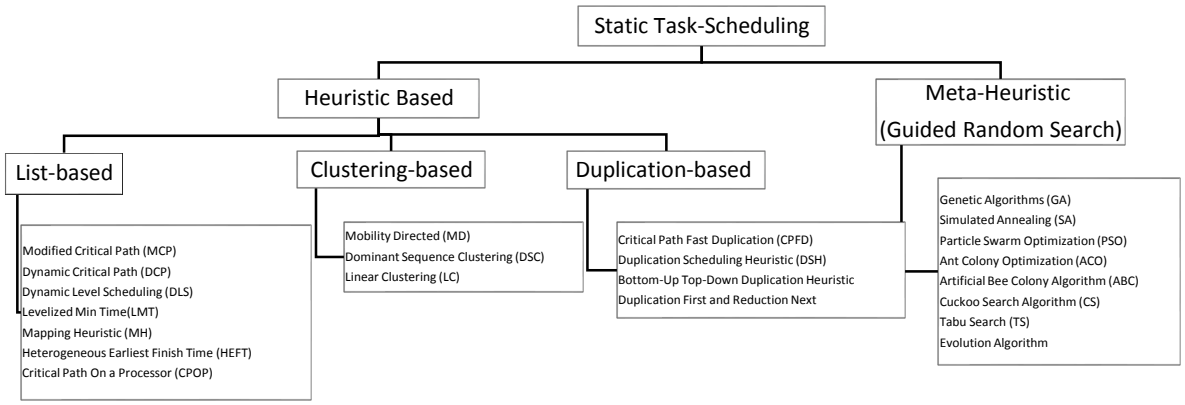


Figure 1 Classification of static task scheduling algorithms.

In clustering-based heuristics, a set of tasks that communicate with each other are grouped together to create a cluster. If the number of created clusters is greater than the number of available processors, clusters are combined so that the number of the remaining clusters equals to the number of processors. Finally, clusters are allocated to the available processors, and local execution of the tasks within each processor is determined. The complexity of clustering-based algorithms tends to be lower than list-based algorithms [2]. The Mobility Directed algorithm (MD) [7], Dominant Sequence Clustering (DSC) [11], and Linear Clustering [12] are examples of clustering algorithms.

The duplication-based scheduling attempts to reduce communication delays by allocating a key task on more than one processor. Some examples for this algorithm are Critical Path Fast Duplication [13], Duplication Scheduling Heuristic [14], Bottom-up Top-down Duplication Heuristic [15], and Duplication First and Reduction Next [16].

A meta-heuristic algorithm generally requires appropriate sampling of candidate solutions in the search space and has shown high performance on a diversity of scheduling problems. It is familiar that Ant Colony Optimization (ACO) [17, 18], Particle Swarm Optimization (PSO) [19-23], Tabu Search (TS) [24, 25], Simulated Annealing (SA) [26, 27], Artificial Bee Colony algorithm (ABC) [28], Cuckoo Search algorithm (CS) [29-31], Evolution Algorithm [32, 33], and Genetic Algorithms (GA) [3, 5, 25, 34-42] have been successfully applied to the task scheduling. Genetic algorithms have been broadly used to develop solutions for several task scheduling problems.

On the other hand, scheduling algorithms can be classified into three groups: heuristic-based, meta-heuristic, and hybrid algorithms [5, 43]. A hybrid algorithm combines both heuristic-based algorithms and meta-heuristic algorithms. Hybrid approach can achieve better performance in terms of makespan for DAG scheduling and reducing the scheduling overhead, when compared with the overhead of meta-heuristic algorithms [3]. It even has a better performance than heuristic algorithms [3]. The Hybrid Heuristic–Genetic Scheduling (H2GS) [5], and Multiple Priority Queues Genetic Algorithm (MPQGA) [3] are examples of this class of algorithms.

4. THE PROPOSED ALGORITHM

The Efficient State Space Search Genetic Algorithm (ESSSGA) has two-step scheduling algorithm. In the first step, the ESSSGA algorithm runs a list-based heuristic scheduling algorithm to reduce space search and generate a high quality task schedule. The solution generated by this step is located in an approximate area in the search space around the optimal solution. In the second step, a Genetic Algorithm searches the approximate area to improve the solution generated by the first step.

Algorithm 1. ESSSGA

```
Input parameters:
        Crossover Probability, Mutation Probability, Size of Run, Size of Population, Number of Processors;
Output:
        A task schedule.
1: Call Initial Population Algorithm        //Algorithm 2;
2: repeat
3:          Call GA Selection Algorithm     //Algorithm 4;
4:          Call GA Crossover Algorithm     //Algorithm 5;
5:          Call GA Mutation Algorithm      //Algorithm 6;
6:          Call GA Selection Algorithm     //Algorithm 4;
7:          Call Inversion Algorithm        //Algorithm 7;
8:          Call GA Crossover Algorithm     //Algorithm 5;
9: until size of run;
10: return a near-optimal schedule.
```

4.1. The outline of ESSSGA

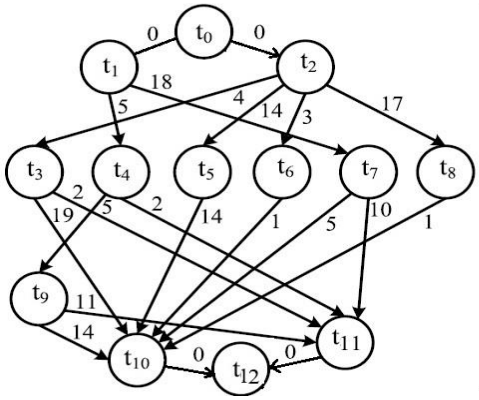
In this paper, the ESSSGA algorithm for task scheduling on heterogeneous systems is improved in order to utilize the advantages of both meta-heuristic-based and heuristic-based algorithms. The ESSSGA task scheduling algorithm implements task scheduling by integrating a GA-based approach to assigning the priority of task execution while using a heuristic-based approach based on Critical Path On a Processor (CPOP) with three proposed operations:

Segmentation, Extraction, and Load Balancing to complete task-to-processor mapping. We use a genetic-based algorithm employs roulette-wheel selection, crossover, mutation, replacement, and proposed new operation named Inversion.

The outline of ESSSGA for DAG task scheduling on heterogeneous computing systems is given in Algorithm 1.

4.2. Chromosome representation

One of the most fundamental tasks in a GA is formulating an encoding mechanism for representing solutions as chromosomes. The most generally practiced mechanism for solution representation used in scheduling problem is having an array of tasks that represents executing order. For this purpose, in this paper, each chromosome represented by a 2-D array, where position i represents task T_i and processor P_j to which the task has been allocated. A chromosome with n genes signifies a solution of the task scheduling problem. Each gene of the chromosome as shown in Figure 3 matches to one of the tasks in a DAG. The order of genes in the chromosome then signifies their execution order. Furthermore, the priority order of the tasks in the chromosome should be a valid topological order, where the entry task should be placed at the beginning of the chromosome, while the exit task should be located at the end. Each chromosome has a fitness value, which is the makespan that is resulted from the mapping of tasks to processors within that chromosome.



task	p0	p1
t1	4	6
t2	15	22.5
t3	4	6
t4	13	19.5
t5	10	15
t6	7	10.5
t7	8	12
t8	4	6
t9	12	18
t10	6	9
t11	9	13.5

Figure 2 A sample DAG containing 13 tasks with computation cost matrix [5].

4.3. Initialization

Random population would not offer a suitable area for discovering the search space effectively, while a too large random population would then diminish the efficiency of the technique that no solution could be expected in a reasonable total finish time. In our proposed algorithm, we adopted an approach using three evaluation criterion [3]: good seeding, good uniform coverage, and genetic diversity for generating the initial population. For this purpose, the ESSSGA algorithm runs a list-based scheduling to generate chromosomes for the initial population based on CPOP [2]. The basic idea of CPOP algorithm is to allocate tasks located in the critical path, on the same processor. Since the communication cost between two tasks mapped on the same processor is equal to zero. The other tasks are scheduled on the processor that is allocate to the task with greatest communication cost to the current task which minimizes the earliest execution finish time of the task.

In our proposed algorithm we used Segmentation, Extraction, and Load Balancing to complete task-to-processor mapping. Segmentation means that all nodes that can be run concurrently at the same level in DAG are classified as a group as shown in Figure 4. In each group, the node on the critical path is chosen to be allocated on the fastest processor which means Extraction. In Load balancing as demonstrated in Algorithm 2, tasks are divided among the processors.

Algorithm 2. Initial Population

```
Input:
        A DAG application;
Output:
        The initial population;
1: Call Segmentation for nodes classification;
2: PopulationN=0;
3: while PopulationN < PopSize do
4:      for each segment generated by Segmentation do
5:              LoadBalancingProcessor = 0;
6:              for each nodes in a segment do
7:                      LoadBalancingProcessor= LoadBalancingProcessor mod m
8:              if current node located on CP
9:                              Set it to the fastest processor;
10:                     else
11:                             Set it to the LoadBalancingProcessor;
12:             endif
13:                     LoadBalancingProcessor ++;
14:             endfor
15:     endfor
16: PopulationN = PopulationN + 1;
17: end while;
18: return the population.
```

The initial population contains $PopSize$ chromosomes. $PopSize$ is the population size to be kept continuously through the generations. Brought as a result of a too small population an appropriate area for exploring the search space effectively would not be available and a too large population would so harm the efficiency of the result, the population size is considered from 10 to 50 with an increment of 5 [5].

A detailed description of the initial population is given in Algorithm 2.

t0	t2	t1	t6	t3	t5	t4	t7	t8	t9	t10	t11	t12
p0	p1	p0	p0	p0	p0	p0	p1	p1	p0	p0	p1	p0

Figure 3 A chromosome that encodes a scheduling for a sample DAG as shown in Figure 2.

Seg 1	t1	t2				
Seg 2	t4	t7	t6	t3	t5	t8
Seg 3	t9					
Seg 4	t11	t10				
Seg 5	t12					

Figure 4 Segmentation for DAG in Figure 2.

4.4. Fitness functions and evaluation

The Fitness function is basically the objective function for scheduling problem. It is used to evaluate the solution, and also control the GA selection function. For scheduling problem we can consider factors, such as throughput, total schedule length, and processor utilization for the fitness function [44]. In this paper, we consider the total schedule length of the DAG as a fitness which is the total finish time among all tasks. For the task scheduling problem, the goal is to find a task assignment that certifies the minimum makespan as fitness. The fitness can be formulated as in Eq.(1)

$$\boldsymbol{fitness}(\boldsymbol{x}) = \max_{P_i \in P}(\boldsymbol{FT}(\boldsymbol{P}_i)), \quad \textbf{(1)}$$

where $FT(P_i)$ is the finishing time of the scheduled exit node on processor P_i. For example, if DAG in Figure 2 is scheduled on 2 processors, the fitness is determined as total execution time on P_0 as shown in Figure 5. The fitness evaluation improves the performance of the GA algorithm. This gives a hint of the best schedule length of near optimal schedules.

A description of Task-to-processor mapping and evaluating the fitness is given in Hence, a more fitted chromosome will be selected with higher probability and will get more children.

Algorithm 3. It runs latter all after other operators which change the chromosomes.

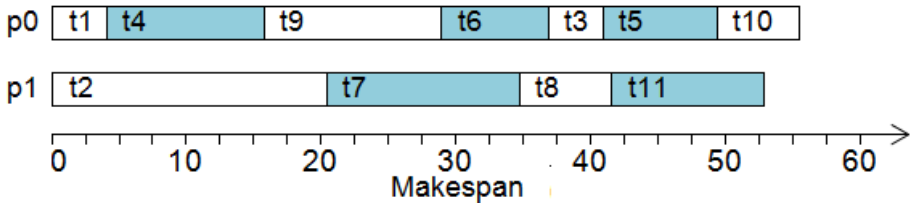


Figure 5 A schedule for the DAG in Figure 2 on 2 processors using the ESSSGA.

4.5. GA operators

In this section, the standard genetic algorithm operators for scheduling, such as selection, crossover, mutation, and replacement are presented. Furthermore, we address a new operator named inversion. The schedule generated by this new operator is located in an approximate area in the search space around the optimal schedule.

4.5.1. GA selection

In our algorithm, 10% of the fittest chromosomes in the population are copied without change to the $elitism$ set, which is a set of the best chromosomes that are copied to the next population. This method guarantees that the best chromosomes are never damaged by either the crossover or the mutation operators.

A detailed description of roulette-wheel selection is given in Algorithm 4. In this algorithm, the probability $Prob_i$ of each chromosome to be selected can be calculated according to the probability defined by Eq(2), Eq(3):

$$\boldsymbol{Prob}_i = \frac{\boldsymbol{fitness}_i}{\sum_{j=1}^{PopSize} \boldsymbol{fitness}_j}, \quad \textbf{(2)}$$

$$\boldsymbol{Sum}_i = \sum_{j=1}^{i} \boldsymbol{Prob}_j, \quad \textbf{(3)}$$

Hence, a more fitted chromosome will be selected with higher probability and will get more children.

Algorithm 3. Evaluate the fitness

```
Input:
        A DAG application;
Output:
        The makespan;
1: TotalExecutionTime=0;
2: Pi .ExecutionTime=0;
3: for each Task in Chromosome that parent_number =0 do
4:          if Processor Scheduled for this task is idle
5;                  assign this task to its processor;
6:                  for each child of this task do
7:                          reduce parent_number;
8:          endfor
9:          Pi .ExecutionTime = TotalExecutionTime +Ti.ExecutionTime;
10:     endif
11:     TotalExecutionTime++;
12:endfor
13:for each Task on processor Pi do
14:     if the parents are not executing on this processor;
15:             add weight between parent and the task to the Pi.ExecutionTime;
16:     endif
17:endfor
18:return Greatest Pi.ExecutionTime as a makespan
```

4.5.2. GA crossover

In order to create two $offspring$ chromosomes, crossover operator works on two chromosomes in the population called parent chromosomes. In our implementation, a crossover is a procedure of replacing only processors of the genes in one parent with the other parent as shown in Figure 6. Therefore, it can generate a new offspring without changing the topological order. Crossover is applied with probability of Pc to the chromosomes in the population.
A description of crossover operator is given in Algorithm 5.

Algorithm 4. GA Selection

```
Input:
        Current population;
Output:
        New Selected Pop;
1: Select 10% of best chromosomes base on makespan and copy to Selected Pop;
2: Generate a random number R ∈ [0,1];
3: for i = 1 to PopSize do
4:          if Sum_i > R then
5:                  Select the iTh chromosome;
6:                  return the iTh chromosome;
7:          end if
8: end for.
```

4.5.3. GA mutation

The mutation operator changes a gene with mutation probability. The mutation operator focuses on continuing the variety of the population so as to expand the search space, and allows the search process to escape from local optimal solutions. We can generate new chromosomes by exchanging two genes in the same segment (same level in DAG or mutation area as shown in Figure 7.), without changing the topological order as shown in Figure 8. Mutation is applied with probability of Pm to the chromosomes in the population.

A description of mutation operator is given in Algorithm 6.

Father (*i*, *j*)

t0	t1	t2	t8	t4	t3	t6	t5	t7	t9	t11	t10	t12
p0	p0	p1	p0	p0	p1	p0	p0	p1	p0	p0	p1	p0

Mother (*i*, *j*)

t0	t2	t1	t8	t4	t3	t5	t7	t6	t9	t10	t11	t12
p0	p0	p0	p1	p1	p1	p0	p0	p0	p0	p0	p0	p0

Son

t0	t1	t2	t8	t4	t3	t6	t5	t7	t9	t11	t10	t12
p0	p0	p1	p1	p1	p1	p0	p0	p1	p0	p0	p1	p0

Daughter

t0	t2	t1	t8	t4	t3	t5	t7	t6	t9	t10	t11	t12
p0	p0	p0	p0	p0	p1	p0	p0	p0	p0	p0	p0	p0

Figure 6 double-point crossover operator.

Algorithm 5. Crossover operator

Input:
Two parents from the current population.
Output:
Two new offsprings.
1: Choose randomly two crossover point i,j;
2: Cut the father's chromosome and the mother's chromosome from i to j and select processors;
3: Copy genes in father's chromosome to the son's chromosome;
4: Inherit the processor of the mother's chromosome to the same position of the son's chromosome;
5: Copy genes in mother's chromosome to daughter's chromosome
6: Inherit the processor of the father's chromosome to the same position of the daughter's chromosome;
7: replace these two new offsprings with two random selected chromosomes in population.

t0	t2	t1	t6	t3	t5	t4	t7	t8	t9	t10	t11	t12
p0	p1	p0	p0	p0	p0	p0	p1	p1	p0	p0	p1	p0

Figure 7 Mutation areas or segments.

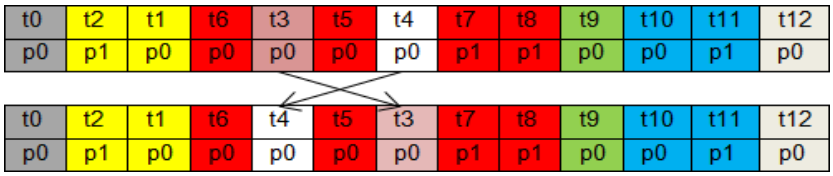


Figure 8 Mutation operator.

Algorithm 6. Mutation operator

Input:
A randomly chosen chromosome.
Output:
A new chromosome.
1: Choose randomly a gene T_i;
2: Generate a new offspring by interchanging gene T_i with a gene in the same level;
3: return the new offspring;

4.5.4. Inversion Algorithm

The inversion algorithm works on all chromosomes in the population. It is a procedure of replacing the order of genes in the same level. Figure 9 shows the inversion on the sample chromosome shown in Figure 7.

A description of the inversion operator is given in Algorithm 7.

t0	t1	t2	t8	t7	t4	t5	t3	t6	t9	t11	t10	t12
p0	p0	p1	p1	p1	p0	p0	p0	p0	p0	p1	p0	p0

Figure 9 Inverted chromosome shown in Figure 7

Algorithm 7. Inversion algorithm

Input:
Current population.
Output:
A new inverted population.
1: for each chromosome in population do
2: for each level in current chromosome do
3: Reverse the order of genes;
4: end for;
5: end for;

5. TIME AND SPACE COMPLEXITY EVALUATION

The time complexity of ESSSGA can be evaluated as following: According to Algorithm 1, the time is mostly spent in running the searching loop (Steps 2–9) in the proposed ESSSGA. In each iteration of the loop, the algorithm needs to execute selection operator, crossover operator, mutation operator, inversion, and fitness evaluation function. The time complexity of the selection operator (Step 3) is $O(n)$. The time complexity of the crossover operator (Step 4) is $O(n^2)$. The time complexity of the mutation operator (Step 5) is $O(n^2)$. The time complexity of the inversion algorithm (Step 7) is $O(n^2 \times l)$, where l is the number of levels in the DAG. The time complexity of the fitness evaluation function is $O(e \times m)$, where e is the number of edges in the DAG and m is the number of processors. Therefore, the time complexity of ESSSGA is $O(G \times (n + n^2 + n^2 + n^2 \times l + e \times m))$, where G is the number of generations performed by ESSSGA. For a dense graph where the number of edges is $O(n^2)$, the time complexity is $O(G \times n^2 \times m)$.

The space complexity of ESSSGA is analyzed as follows: In ESSSGA, to store each chromosome an array of size $n \times 2$ is needed. There are $PopSize$ chromosomes in the initial population, hence, the space complexity of ESSSGA is $O(PopSize \times n \times 2)$.

6. SIMULATION AND RESULTS

In this section, the performance of the ESSSGA is presented in comparison with the DCP[4], DSC[11], MD[7], MCP[7], DLS [8], HEFT [2], CPOP [2], LDCP [45], H2GS [5], and MPQGA [3] which are the best existing scheduling algorithms for heterogeneous multiprocessor systems. To evaluate our proposed algorithm, we have implemented it in C# using an Intel processor Core 2 Duo @ 2.1 GHz and 2 GB RAM.

6.1. Performance metrics

Several metrics were used for the performance evaluation. In our experiments, the performance metrics chosen for comparison are Makespan, Scheduling Length Ratio (SLR), Speedup, and Efficiency that are described as follows:

6.1.1. Makespan

The makespan is the total tasks execution time in scheduling (schedule length) formulated as Eq. (4)

$$\boldsymbol{makespan = FT(T_{exit})}, \quad \textbf{(4)}$$

where $FT(T_{exit})$ is the finishing time of the scheduled exit node [46].

6.1.2. Scheduling length ratio (SLR)

The Scheduling Length Ratio (SLR) of a given schedule is defined as the normalized schedule length normalized to the lower bound of the schedule length. Since a large set of application graphs with different characteristics is used, it is necessary to normalize the schedule length to the lower bound [2]. On the other hand, SLR is the ratio of the parallel time to the sum of weights of the critical path tasks on the fastest processor which is calculated using Eq. (5).

$$\boldsymbol{SLR = \frac{makespan}{\sum_{T_i \in CP_{min}} min_{P_j \in Q}\{W(T_i,P_j)\}}} \quad \textbf{(5)}$$

The CP_{min} is the critical path of the unscheduled application DAG based on the computation cost of tasks on the fastest processor P_j. The denominator is equal to the sum of computation costs of tasks located in CP_{min} when they are scheduled on P_j and provides the lower bound on the schedule length [3]. The SLR value of any algorithm for a DAG cannot be less than one, since the denominator in the equation is a lower bound for the completion time of the graph. In order to compare the performance of scheduling algorithms, the average SLR values over several task graphs were used in the experiments where the smaller the SLR the better the result [3].

In some papers, the Normalized Schedule Length (NSL) is used in place of SLR as shown in Eq. (6) [5, 45, 47].

$$\boldsymbol{NSL = \frac{\text{Schedule Length}}{\sum_{t_i \in CP_{lower}} C_{i,a}}} \quad \textbf{(6)}$$

6.1.3. Speedup and efficiency

The Speedup value of a schedule is defined as ratio of the sequential schedule length obtained by assigning all tasks to the fastest processor, to the parallel execution time of the task schedule [2] which is calculated using Eq. (7).

$$\boldsymbol{Speedup = \frac{min_{P_j \in Q}\left\{\sum_{T_i \in T} W(T_i,P_j)\right\}}{makespan}} \quad \textbf{(7)}$$

Efficiency is the ratio of the Speedup to the number of processors used to schedule the graph. In addition to providing minimum SLR values, the task scheduling algorithms target to maximize the speedup and efficiency [2].

In this paper, every experimental result is the average value of 50 separate runs.

Table 2 Comparative results based on sample DAG in Figure 2

Makespan	SLR	Speedup	No. of processors	Algorithm
64	2.06	1.44	2	Longest Dynamic Critical Path (LDCP)
65.5	2.1	1.4	2	Dynamic Level Scheduling (DLS)
65.5	2.1	1.4	2	Heterogeneous Earliest Finish Time (HEFT)
61.5	1.98	1.49	2	Hybrid Heuristic–Genetic Scheduling (H2GS)
56	1.8	1.64	2	ESSSGA

6.2. Performance results on sample graphs

Considering the application DAG shown in Figure 2, the schedule generated by the ESSSGA algorithm has a length of 56 (Figure 5) which is shorter than the schedules generated by the LDCP (64), DLS (65.5), HEFT (65.5), and H2GS (61.5) [5] as shown in Table 2.

The next results, simulated in two DAGs as shown in Figure 10 and Figure 11 are presented in Table 3, Table 4, and Table 5. The following parameters are used in the algorithms throughout the simulations [44]**.**

- Population size = 20 and 30.
- Maximum number of generation = 1000
- Crossover probability (Pc) = 0.7
- Mutation probability (Pm) = 0.3

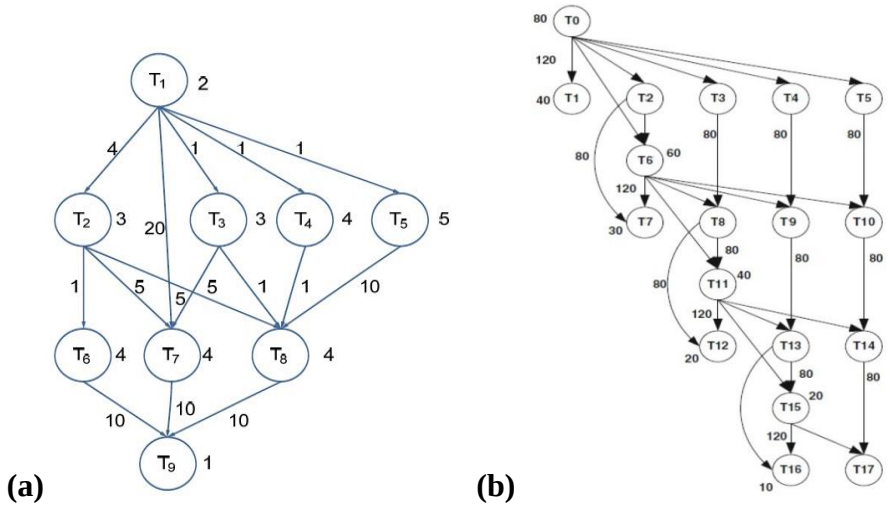


Figure 10 (a) A sample DAG with 9 tasks, (b) A sample gaussian graph [44]

Table 3 Comparative results based on sample DAG in Figure 10 (a)

Makespan	SLR	Speedup	No. of processors	Algorithm
29	4.14	1.03	3	Modified Critical Path (MCP)
32	4.56	0.94	2	Dynamic Critical Path (DCP)
27	3.85	1.11	4	Dominant Sequence Clustering (DSC)
32	4.56	0.94	2	Mobility Directed (MD)
20	2.85	1.5	2	
20	2.85	1.5	3	ESSSGA
20	2.85	1.5	4	

Table 4 Comparative results based on sample DAG in Figure 10 (b)

Makespan	SLR	Speedup	No. of processors	Algorithm
520	1.74	1.15	4	Modified Critical Path (MCP)
440	1.47	1.36	3	Dynamic Critical Path (DCP)
460	1.54	1.3	6	Dominant Sequence Clustering (DSC)
460	1.54	1.3	3	Mobility Directed (MD)
440	1.47	1.36	3	
410	1.37	1.46	4	ESSSGA
410	1.37	1.46	6	

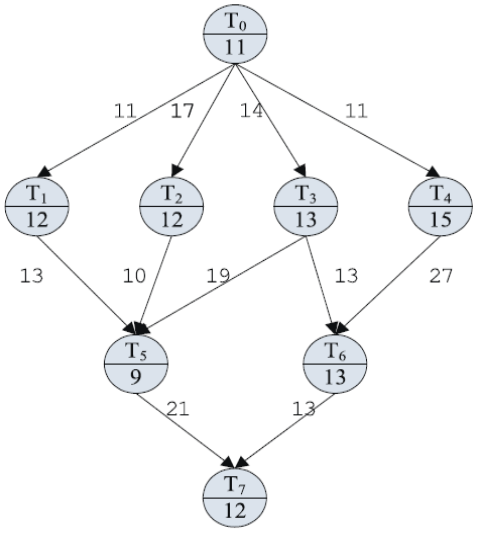


Figure 11 A sample DAG with 8 tasks [3]

Table 5 Comparative results based on sample DAG in Figure 11

Makespan	SLR	Speedup	No. of processors	Algorithm
80	3.74	1.14	3	Heterogeneous Earliest Finish Time (HEFT)
86	4.02	1.06	3	Critical Path On a Processor (CPOP)
65	3.04	1.4	3	Multiple Priority Queues Genetic Algorithm (MPQGA)
61	2.85	1.5	3	ESSSGA

7. CONCLUSIONS

In this paper, we present a new algorithm, named ESSSGA, for static task scheduling on heterogeneous multiprocessor systems. The objective of this algorithm is to find a schedule for minimizing the total execution time. For this purpose, the aim is to find a hybrid algorithm was presented to deliver the best schedule length of near optimal schedules using the advantages of both meta-heuristic-based and heuristic-based algorithms.

In future study, we found more factors which affect the efficiency of the ESSSGA algorithm, such as the initial population size where a method by which its individuals are chosen, the particular crossover, and mutation operators. Further we will test the ESSSGA on more standard task graphs, more processors, and variable amounts of heterogeneity among processors and also tasks on random graphs. We have even planned to extend ESSSGA using Cuckoo optimization algorithm [48] to minimize makespan.

REFERENCES

1. J. D. Ullman, "NP-complete scheduling problems," Journal of Computer and System sciences, vol. 10, pp. 384-393, 1975.
2. H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, pp. 260-274, 2002.
3. Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, vol. 270, pp. 255-287, 2014.
4. Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," Parallel and Distributed Systems, IEEE Transactions on, vol. 7, pp. 506-521, 1996.
5. M. I. Daoud and N. Kharma, "A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks," Journal of Parallel and Distributed Computing, vol. 71, pp. 1518-1531, 2011.
6. S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," Parallel and Distributed Systems, IEEE Transactions on, vol. 14, pp. 533-544, 2003.
7. M.-Y. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," IEEE Transactions on Parallel and Distributed Systems, vol. 1, pp. 330-343, 1990.
8. G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," Parallel and Distributed Systems, IEEE Transactions on, vol. 4, pp. 175-187, 1993.
9. M. A. Iverson, F. Özgüner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," in 4TH HETEROGENEOUS COMPUTING WORKSHOP (HCW'95, 1995.
10. H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," Journal of parallel and Distributed Computing, vol. 9, pp. 138-153, 1990.
11. T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," Parallel and Distributed Systems, IEEE Transactions on, vol. 5, pp. 951-967, 1994.
12. S. Kim and J. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," in International conference on parallel processing, 1988, p. 8.
13. I. Ahmad and Y.-K. Kwok, "A new approach to scheduling parallel programs using task duplication," in Parallel Processing, 1994. ICPP 1994 Volume 2. International Conference on, 1994, pp. 47-51.
14. B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," Software, IEEE, vol. 5, pp. 23-32, 1988.
15. Y.-C. Chung and S. Ranka, "Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors," in Supercomputing'92., Proceedings, 1992, pp. 512-521.
16. G.-L. Park, B. Shirazi, and J. Marquis, "DFRN: A new approach for duplication based scheduling for distributed memory multiprocessor systems," in Parallel Processing Symposium, 1997. Proceedings., 11th International, 1997, pp. 157-166.
17. F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 29, pp. 911-924, 2010.
18. H. Kim and S. Kang, "Communication-aware task scheduling and voltage selection for total energy minimization in a multiprocessor system using ant colony optimization," Information Sciences, vol. 181, pp. 3995-4008, 2011.
19. H. Li, L. Wang, and J. Liu, "Task scheduling of computational grid based on particle swarm algorithm," in Computational Science and Optimization (CSO), 2010 Third International Joint Conference on, 2010, pp. 332-336.
20. Q. Kang and H. He, "A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems," Microprocessors and Microsystems, vol. 35, pp. 10-17, 2011.
21. A. Al Badawi and A. Shatnawi, "Static scheduling of directed acyclic data flow graphs onto multiprocessors using particle swarm optimization," Computers & Operations Research, vol. 40, pp. 2322-2328, 2013.
22. A. Husseinzadeh Kashan, M. Husseinzadeh Kashan, and S. Karimiyan, "A particle swarm optimizer for grouping problems," Information Sciences, vol. 252, pp. 81-95, 2013.
23. F. Zhao, J. Tang, and J. Wang, "An improved particle swarm optimization with decline disturbance index (DDPSO) for multi-objective job-shop scheduling problem," Computers & Operations Research, vol. 45, pp. 38-50, 2014.

24. R. Shanmugapriya, S. Padmavathi, and S. M. Shalinie, "Contention awareness in task scheduling using Tabu search," in Advance Computing Conference, 2009. IACC 2009. IEEE International, 2009, pp. 272-277.
25. G. Garai and B. Chaudhurii, "A novel hybrid genetic algorithm with Tabu search for optimizing multi-dimensional functions and point pattern recognition," Information Sciences, vol. 221, pp. 28-48, 2013.
26. J. Wang, Q. Duan, Y. Jiang, and X. Zhu, "A new algorithm for grid independent task schedule: genetic simulated annealing," in World Automation Congress (WAC), 2010, 2010, pp. 165-171.
27. J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," Information Sciences, vol. 185, pp. 137-152, 2012.
28. A. R. Yildiz, "Optimization of cutting parameters in multi-pass turning using artificial bee colony-based approach," Information Sciences, vol. 220, pp. 399-407, 2013.
29. R. Babukartik and P. Dhavachelvan, "Hybrid Algorithm using the advantage of ACO and Cuckoo Search for Job Scheduling," International Journal of Information Technology Convergence and Services, vol. 2, 2012.
30. P. Guo, W. Cheng, and Y. Wang, "Parallel machine scheduling with step deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm," arXiv preprint arXiv:1309.1453, 2013.
31. A. R. Yildiz, "Cuckoo search algorithm for the selection of optimal machining parameters in milling operations," The International Journal of Advanced Manufacturing Technology, vol. 64, pp. 55-61, 2013.
32. A. R. Yildiz, "A comparative study of population-based optimization algorithms for turning operations," Information Sciences, vol. 210, pp. 81-88, 2012.
33. A. R. Yildiz, "Comparison of evolutionary-based optimization algorithms for structural design optimization," Engineering applications of artificial intelligence, vol. 26, pp. 327-333, 2013.
34. H. K. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," George Washington University, 1995.
35. A. S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," Parallel and Distributed Systems, IEEE Transactions on, vol. 15, pp. 824-834, 2004.
36. M. I. Daoud and N. Kharma, "An efficient genetic algorithm for task scheduling in heterogeneous distributed computing systems," in Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, 2006, pp. 3258-3265.
37. F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," Journal of Parallel and Distributed Computing, vol. 70, pp. 13-22, 2010.
38. O. Sathappan, P. Chitra, P. Venkatesh, and M. Prabhu, "Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system," International Journal of Information Technology, Communications and Convergence, vol. 1, pp. 146-158, 2011.
39. J. Singh and G. Singh, "Improved Task Scheduling on Parallel System using Genetic Algorithm," International Journal of Computer Applications, vol. 39, pp. 17-22, 2012.
40. M. Pedergnana, P. R. Marpu, M. D. Mura, J. A. Benediktsson, and L. Bruzzone, "A novel technique for optimal feature selection in attribute profiles based on genetic algorithms," Geoscience and Remote Sensing, IEEE Transactions on, vol. 51, pp. 3514-3528, 2013.
41. M. D. Robles-Ortega, L. Ortega, and F. R. Feito, "A new approach to create textured urban models through genetic algorithms," Information Sciences, vol. 243, pp. 1-19, 2013.
42. R. Köker, "A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization," Information Sciences, vol. 222, pp. 528-543, 2013.
43. A. Y. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: comparative studies and performance issues," Parallel and Distributed Systems, IEEE Transactions on, vol. 10, pp. 795-812, 1999.
44. M. R. Mohamed and M. H. Awadalla, "Hybrid Algorithm for Multiprocessor Task Scheduling," International Journal of Computer Science Issues, vol. 8, pp. 79-89, 2011.
45. M. I. Daoud and N. Kharma, "Efficient compile-time task scheduling for heterogeneous distributed computing systems," in Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on, 2006, p. 9 pp.
46. T. Hagras and J. Janeček, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," Parallel Computing, vol. 31, pp. 653-670, 2005.
47. M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," Journal of Parallel and distributed computing, vol. 68, pp. 399-409, 2008.
48. X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, 2009, pp. 210-214.