Hard Real-Time Multiobjective Scheduling in Heterogeneous Systems Using Genetic Algorithms

Mohammad Reza Miryani
Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
miryani@stu-mail.um.ac.ir

Mahmoud Naghibzadeh
Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
naghibzadeh@um.ac.ir

Abstract

Optimal tasks allocation is one of the most important problems in multiprocessing. Optimal assignment of tasks to a multiprocessor is an NP-hard problem in general cases, and precedence task graph makes it more complex. Many factors affect optimal tasks allocation. One of them is cache reload time in multiprocessor systems. These problems exist in real-time systems, too. Due to high sensitivity of 'time' in real-time systems, scheduling with respect to time constraints becomes very important. This paper proposes a suboptimal scheduler for hard real-time heterogeneous multiprocessor systems considering time constraints and cache reload time simultaneously, using multiobjective genetic algorithm. In addition, it tries to propose a generalized method for real-time multiobjective scheduling in multiprocessor systems using genetic algorithms.

1. Introduction

Computational activities and their responses should be performed within a specified time-frame in real-time systems. A task τ_i requested at time t_i needs c_i units of time for execution and this time shall be allocate to it before its deadline $t_i + d_i$. Otherwise, problems may arise in the system. Real-time systems are classified into two categories with respect to the severity of missing a deadline to Hard Real-Time and Soft Real-Time systems [1]. All deadlines shall meet, in hard real-time systems.

Some optimal scheduling of real-time tasks, on single processor systems to meet deadlines are already developed considering the task characteristics. Scheduling algorithm of EDF and RM are optimal. However, in multiprocessor systems there is no known optimal scheduler [1], [2].

To be optimal for a scheduler means the schedules satisfies one or more criteria of optimality [1], [2]. Generally, it means each task was allocated to a processor such that the overall system is optimal based on predefined criteria. In real-time systems, these criteria can be total tardiness, completion time, throughput, utilization, waiting time, etc. Finding an efficient optimal scheduler for multiprocessor systems is an open problem [3] - [5].

Reference [3] has shown that just minimizing total tardiness for N independent task on one machine is an NP-hard problem and in majority of cases, the solution is an NP-hard one. Therefore, developing heuristic algorithms is useable in many applications. Genetic Algorithms is one such algorithm with reasonable efficiency, in many cases [4]-[6].

In recent years, several genetic approaches have been proposed for multiprocessor environments. Reference [6] proposes a scheduler with genetic algorithm for nonpreemptive tasks with precedence and deadline constraints but it does not have suitable performance necessarily. Reference [7] presents a hybrid genetic algorithm, in which different operators are applied at different stages of the lifetime, for scheduling partially ordered nonpreemptive tasks in a multiprocessor environment. Reference [8] proposes a genetic algorithm implementation to solve a scheduling problem for real-time nonpreemptive tasks.

These algorithms minimize only one objective such as completion time, total tardiness, or cost. Reference [5] presents a multiobjective genetic algorithm for scheduling nonpreemptive tasks in a soft real-time system with symmetric processors. Nevertheless, some extra local improvement heuristics has been used to find the smallest number of processors. In addition, this work has not considered cache reload time. Other works are in [9], [10] for tasks without timing constraints, but [4] considers to a multiobjective scheduling problem for nonpreemptive soft real–time tasks with conflicting objectives, total tardiness and completion time without considering cache reload time.

With respect to processor affinity one way to decrease the execution time is to try to assign processors to execute tasks so that two related tasks that share their code and data segments be executed on same processor. Therefore second task does not need to fetch all its data from main memory or auxiliary memories and it can use the already fetched data.

In this paper, we propose a new scheduling algorithm for non-preemptive tasks with precedence relationship on heterogeneous multiprocessor systems, with cache reload time (CRT) and other timing constrains. CRT is important because in practical systems, all timing constrains should be considered otherwise the system may crash. The criteria are completion time and number of processors in a way that all of deadlines are met. Trying to minimize both criteria is done simultaneously. Since there is a conflict between objectives, we use Adaptive Weight Approach [4], [11]. Adaptive Weight Approach uses some useful current population's information in order to justify weights and to move searching in answer space towards positive answers.

The rest of the paper is organized as follows. In section 2, scheduling problem for hard real-time tasks on heterogonous multiprocessors will be defined mathematically. Section 3 describes the proposed genetic algorithm, applied procedures, genetic operators, and stopping condition. Section 4 and Section 5 explain validation and the experience results, respectively. Finally, conclusion and future works are in section 6.

2. Mathematical Model for Hard Real-Time Scheduling Problem

In this research, we consider the offline scheduling of a set of hard real-time tasks with precedence constraint with task graph on a set of heterogeneous processors in which completion time (f_1) and number of processors (f_2) are to be minimized under the following conditions:

- All tasks are nonpreemptive
- Every task is processed on only one processor at a time
- Every processor processes only one task at a given time
- All deadlines must be met.

In addition, there are these assumptions:

- A time unit is an artificial time unit
- Execution time of all tasks on each processor is given
- Precedence relationship or task graph is given prior to scheduling
- Cache reload time with respect to task graph and run time is computable.

Therefore, mathematical statements formulate problem as follows. Presented formulations are developed in [4]-[6] and we have done proper modifications based on new requirements, objectives and limitations of defined problem:

$$\min f_1 = \max\{t_i^F\} \quad (1)$$

$$\min f_2 = \text{Number of Processors} \quad (2)$$

$$s.t. \quad t_i^E \le t_i^S \quad \forall i \quad (3)$$

$$t_i^E \ge \max_{\tau_j \in pre(\tau_i)} \{t_j^F\} \quad (4)$$

$$\sum_{m=1}^{M} x_{im} = 1 \quad \forall i, \quad (5)$$

$$x_{im} \in \{0,1\} \ \forall i, m \quad x_i = [x_{im}]_{1\times M} \quad (6)$$

Equations (1) and (2) are fitness functions in this scheduling problem. Equation (1) defines minimizing completion time of tasks because minimization of finish time of each task (t_i^F) means that the completion time of the set of tasks is minimized, and (2) expresses minimizing number of processors. Constraint conditions have been shown in (3) to (6). Equation (3) means a task can be started after its own earliest start time begins [4] (t_i^S: real start time of τ_i). Equation (4) shows earliest start time (t_i^E) of the task which is based on task graph. In the other word, each task can execute on a processor after its precedence tasks is finished and its initial data reload from other processors in the cache is done. So we exert cache reload time by (4). Equation (5) means that each processor process only one task at a time. Equation (6) is a decision variable because the system is heterogeneous. Note it is required to meet deadlines in hard real-time systems. Thus, there is a default objective formulated as follows:

$$\min f_3 = \sum_{i=1}^{N} \max\{0, \sum_{m=1}^{M} (t_i^S + c_{im} - \max\{crt_{ji} x_i x_j^T : \tau_j \in pre(\tau_i)\} - d_i) x_{im}\} \quad (7)$$

Equation (7) shows that when completion time of task is carried out after the relevant deadline, the system would have tardiness. Otherwise, tardiness will be equal to zero. Tardiness is not acceptable in the hard real-time systems. It is unacceptable because it has deadly effects. So it has to be equal to zero in our study.

Following and developing definitions on [4], [5] the following notations are used for the above equations:

- Indices:
  - i, j: task index, i, j=1, 2, … , N
  - m: processor index, m=1, 2, … , M
- Parameters:
  - N: Total number of tasks
  - M: Total number of processors
  - $G(T,E)$: task graph
  - $T = \{\tau_1, \tau_2, \dots, \tau_n\}$: a set of N tasks
  - $E = \{e_{ij}\}$, $i, j = 1,2, \dots, N$, a directed acyclic graph representing precedence relationship
  - k_{ij}: $\forall eij \ \exists k_{ij}$ *: is a random value in* $\in [10^3, 10^6]$
  - e_{ij}: precedence relationship between task τ_i and task τ_j
  - c_{im}: computation time of task τ_i on m^{th} processor
  - $$crt_{ij} = \begin{cases} \gamma\, k_{ij}\ accessTime \text{ if a same processor} \\ \qquad\qquad \text{processes } \tau_i \text{ and } \tau_j \\ 0 \qquad\qquad \text{otherwise} \end{cases} \quad (8)$$
    γ is a random value in [0,1], *accessTime* is average time to access main memory and auxiliary memory. γ $k_{ij}accessTime$ is not more than $0.05 \times c_{im}$.
  - d_i: deadline of task τ_i
  - $pre^*(\tau_i)$: set of all predecessors of task τ_i
  - $suc^*(\tau_i)$: set of all successors of task τ_i
  - $pre(\tau_i)$: set of immediate predecessors of task τ_i
  - $suc(\tau_i)$: set of immediate successors of task τ_i
  - t_i^E : earliest start time of task τ_i

$$t_i^E = \begin{cases} 0 & \text{if } pre(\tau_i) = \varnothing \\ \max\limits_{\tau_j \in pre^*(\tau_i)} \left\{ t_j^E + \sum\limits_{m=1}^{M} \left(c_{jm} x_{jm} - \max\left\{ crt_{kj} \times x_k x_j^T \mid \tau_k \in pre(\tau_j) \right\} \right) \right\} \end{cases} \quad (9)$$

- t_i^F : finish time of task τ_i

$$t_i^F = \min\left\{ t_i^S + c_{im} x_{im} - \max\left\{ crt_{ji} \times x_i x_j^T \mid \tau_j \in pre(\tau_i) \right\}, d_i \right\} \forall i, \quad (10)$$

- Decision variables:
  - t_i^S : real start time of task τ_i
  - $$x_{im} = \begin{cases} 1 \text{ if processor } p_m \text{ is selected for} \\ \quad \text{task } \tau_i \text{ ,} \\ 0 \text{ otherwise.} \end{cases} \quad (11)$$

3. The Proposed Genetic Algorithm

In this paper, our proposed scheduler is based on genetic algorithm. In genetic algorithm, an initial population of feasible answers is shown by a set of chromosomes. Then, a new population of chromosomes is produced by applying operations, such as selection, crossover, mutation, etc. The process of producing new generation continues until a stopping criterion is satisfied. Encoding acts as a mapping of feasible answers space of the problem to initial population and decoding evaluates chromosomes towards an ideal answer.

For scheduling problem, several methods and versions for genetic's operators and procedures have been proposed and some of them can be found in [4]-[6]. In this paper, we propose a new encoding procedure. In addition, we have used the proposed decoding procedure in [4], and have extended it to be useful for our problem.

3.1. Encoding

A chromosome ch_k = 1,2,…,*populationSize* is a feasible map from set of tasks to set of processors, in which the *populationSize* is the total number of chromosomes. A chromosome has two parts: $u(.)$ and $v(.)$. $u(.)$ shows scheduling order and $v(.)$ means allocation information [4]. The length of each chromosome is equal to the number of tasks, because all of the tasks must be executed. Scheduling order must satisfy a 'Topologic Sort' result [12] with respect to task graph. Allocation information determines that each processor shall execute which task.

References [4], [5] propose an encoding procedure, while considering topological order but [5] has not implemented topological order. We noticed some errors in the implemented of [4]. In every next level of scheduling, we not only can schedule a task's children but also tasks without precedence. By doing this, we are able to produce more scheduling orders, and it will have positive effect on meeting deadlines. The proposed encoding procedure is shown in Figure 1. Line 16 is designed to satisfy topological sort.

In addition, for initial state, total number of processors is assumed to be equal to the total number of tasks. In other hand, in order to meet deadlines, each task must execute on a separate processor, in the worst case. Therefore, in line 13 of Figure1 M is equal to N.

Figure 2 is an example of task graph related to an assumed application with seven tasks.

Table 1 is information about this application. It shows execution time of each task on processors and Table 2 shows one run of the encoding procedure. Finally, the instance output chromosome of encoding procedure is shown in Figure 3.

```
procedure: Encoding for Scheduling Problem for Hard Real-Time
input: task graph data set, total number of processors M
output: u(.),v(.)
begin
    l ← 1 , W ← ∅;
    while T ≠ ∅
        W ← { τi | pre*(τi) = ∅ ∀i}
        T ← T − W
        j ← random(W);
        u(l) ← j;
        W ← W − { j }
        pre*(τi) ← pre*(τi) −{ τj} ∀i;
        m ← random[1: M]
        v(l) ← m;
        l ← l + 1;
        T ← T ∪ {τi, i ∈ W};
    endwhile;
    output u(.),v(.);
end;
```

Figure 1. Encoding procedure

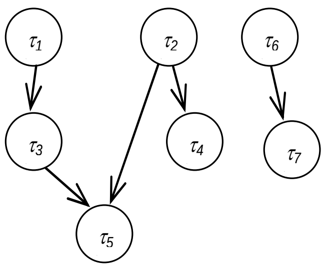

Figure 2. An example precedence task graph for an application with seven tasks

Table 1. Data set of assumed application related to Figure 2

i	$pre^*(\tau_i)$	c_{im}							d_i
		c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	c_{i7}	
1	-	3	10	5	2	10	3	2	20
2	-	11	7	10	11	15	11	12	17
3	1	12	6	13	10	7	12	10	31
4	2	8	9	9	8	5	8	10	23
5	1,2,3	12	5	10	6	10	12	6	45
6	-	10	3	8	22	13	25	22	30
7	6	4	11	7	7	9	4	5	32

l	1	2	3	4	5	6	7
$u(.)$	6	2	4	7	1	3	5
$v(.)$	3	1	2	5	1	4	3

Figure 3. An example of encoding procedure and a typical output chromosome (Chromosome ch_k) (Figure 1, 2 and Table 2)

Table 2. An instance run of task graph on encoding procedure

l	m	j	W	T
			∅	$\{\tau_1,\tau_2,\tau_3,\tau_4,\tau_5,\tau_6,\tau_7\}$
1	3	6	{1,2,6}	$\{\tau_3,\tau_4,\tau_5,\tau_7\}$
				$\{\tau_1,\tau_2,\tau_3,\tau_4,\tau_5,\tau_7\}$
2	1	2	{1,2,7}	$\{\tau_3,\tau_4,\tau_5\}$
				$\{\tau_1,\tau_3,\tau_4,\tau_5,\tau_7\}$
3	2	4	{1,4,7}	$\{\tau_3,\tau_5\}$
				$\{\tau_1,\tau_3,\tau_5,\tau_7\}$
4	5	7	{1,7}	$\{\tau_3,\tau_5\}$
				$\{\tau_1,\tau_3,\tau_5\}$
5	1	1	{1}	$\{\tau_3,\tau_5\}$
				$\{\tau_3,\tau_5\}$
6	4	3	{3}	$\{\tau_5\}$
				$\{\tau_5\}$
7	3	5	{5}	∅
			∅	∅

3.2. Decoding

Decoding procedure is shown in Figure 4 that is the same decoding in [4]. Total tardiness of each task is computed in line 14, completion time of all tasks is determined in line 18 and number of applied processors is calculated in line 19. Table 3 shows tracing of decoding procedure for the presented chromosome in Figure 3.The Gantt chart of Table 3 and obtained fitness functions f_1, f_2 and f_3 are shown in Figure 5.

```
1   procedure: Decoding for Scheduling Problem for Hard Real- Time
2   input: task graph data set, chromosome ch_k
3   output: schedule set S, completion time f1, number of
            processors f2, total tardiness of tasks f3
4   begin
5       l ← 1, S ← ∅;
6       while l ≤ N
7           i ← u(l)
8           m ← v(l)
9           if (exist suitable idle time) then
10              insert(i);
11          endif;
12          start(i);
13          update_idle();
14          f3 ← f3 + max {0 , ti^S + c_im - max{crt_ji x_i x_j^T |
                τj∈pre(τi)}- d_i } ;
15          S ← S ∪ { i, m : ti^S - ti^F };
16          l ← l + 1;
17      endwhile;
18      f1 ← max { ti^F };
19      f2 ← Different Numbers in v(.);
20      output S, f1, f2, f3;
21  end;
```

Figure 4. Decoding procedure

Table 3. Tracing of presented chromosome in Figure 3

l	*i*	*m*	t_i^S	t_i^F
1	6	3	1	25
2	2	1	1	12
3	4	2	14	23
4	7	5	10	19
5	1	1	13	16
6	3	4	18	28
7	5	3	30	40

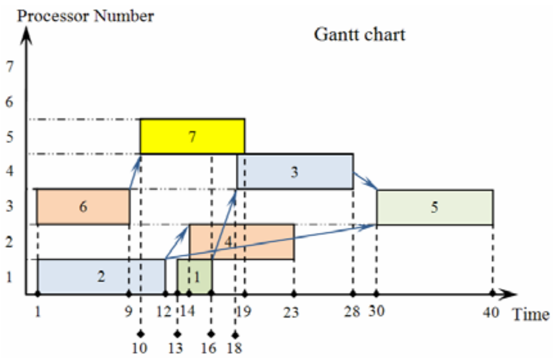

Figure 5. The output of decoding procedure and the Gantt chart of instance chromosome (chromosome related to Figure 3)

3.3 Evolution Function and Competitive Selection

We use Adaptive Weighted Approach (AWA) to move ideal positive point in this research [4], [11]. In AWA, maximum and minimum values are obtained among all the values of fitness functions of chromosomes by (11). Next, adaptive weight of each fitness function is calculated by (13). Then, the weighted-sum objective function for each chromosome is computed by (14). Finally, evaluated function for each chromosome is obtained as shown in (15).

$$f_i^y = \begin{cases} \max\{f_i(ch_k)\} & \text{for y = max} \\ \min\{f_i(ch_k)\} & \text{for y = min} \end{cases} \quad (11)$$

where i=1,2,3; k =1, … ,*populationSize* (12)

$$w_p = \frac{1}{f_p^{\max} - f_p^{\min}}, \quad p = 1,2,3 \quad (13)$$

$$F(ch_k) = \sum_{p=1}^{3} w_p f_p(ch_k) \quad (14)$$

$$= \sum_{p=1}^{3} \left(f_p(ch_k) / f_p^{\max} - f_p^{\min} \right)$$

$$eval(ch_k) = 1/F(ch_k) \quad (15)$$

For competitive selection we have used of Roulette Wheel Selection [11].

3.4. Genetic Operators

We have used modified one-cut crossover and standard mutation ([4], [11]) in this research as shown in Figure 6 and 7 respectively. Procedures of them operate on the *v*(.) part of chromosomes. Because, if they operate on the *u*(.) part, scheduling order might be changed. Therefore, it will not agree with task graph. So, our modified operators operate only on the *v* part of the chromosomes.

```
procedure: Crossover
input: parent chromosomes ch_k, ch_k'
output: proto-offspring chromosomes ch_k, ch_k'
begin
      r ← random[1:N];
      temp ← v([r + 1: N]);
      v([r + 1: N]) ← v'([r + 1: N]);
      v'([r + 1: N]) ← temp;
      output ch_k, ch_k';
end;
```

Figure 6. One-cut crossover operator operates on the v(.) part

```
procedure: Mutation
input: chromosome ch_k,
output: offspring chromosomes ch_k
begin
      r ← random[1:N];
      v(r) ← random[1:M];
      output ch_k;
end;
```

Figure 7. Mutation operator operates on the v(.) part

3.5. Proposed Genetic Algorithm

Proposed genetic algorithm is presented in Figure 8. Algorithm terminates when main loop in line 7 reaches a default value. In the other hand, it is iterated for a fix number of times.

4. Validation

For evaluation of the proposed genetic algorithm several numeral experiments were preformed. Numeral experiments are done with a random precedence task graph.

We used P-Method [13] to produce the random precedence task graph. P-Method is based on an adjacency matrix of a task graph. If there is a precedence relation between tasks τ_i and τ_j then element a_{ij} of adjacency matrix will be one, otherwise it will be zero. An adjacency matrix is made with all its lower triangular and diagonal elements equal to zero. Each of the remaining upper triangular elements of the matrix is examined independently as part of a Bernoulli process with factor ε, which represents the probability of a success. For each element, when the Bernoulli test is a success, then the element is assigned a value of one; for a failure the element is given a value of zero. The parameter ε can be considered to be the sparsity of the task graph. With this method, a probability parameter of $\varepsilon = 1$ creates a totally sequential task graph, and $\varepsilon = 0$ creates an inherently parallel one. Values of ε that lie in between these two extremes generally produce task graphs that possess intermediate structures [13], [4], [5].

```
procedure: Proposed_Genetic_Algorithm
input: task graph data set
output: best schedule set S
begin
  numberOfGeneration ← 0;
  initialize population(numberOfGeneration) byEncoding
  procedure;
  while (NumberOfGeneration ≤ maxGeneration) do
    Evaluate f1, f2 ,f3 by Decoding procedure;
    eval(population); /*eavl(chk): k=1,..,populationSize (15)
    if (not NumberOfGeneration ≤ maxGeneration) then
      creating new_population by roulette wheel selection;
      new_population ← crossover(new_population);
      new_population ← Mutation(new_population);
      population ← new_population;
      numberOfGeneration ← numberOfGeneration +1;
    endif;
  endwhile;
  output best schedule set S;
end;
```

Figure 8. The proposed genetic algorithm

For the tasks' computation time, deadline and cache reload time between them, we use random numbers based on exponential distribution and normal distribution as follows ([4], [14], [15]):

- c_{im}^{Exp} : a random value based on exponential distribution with mean 5,
- c_{im}^{Nor} : a random value based on normal distribution with mean 5 and variance 2,
- r_1^{Exp}: a random value based on exponential distribution with mean c_{im}^{Exp},
- r_1^{Nor}: a random value based on normal distribution with mean c_{im}^{Nor},
- d_i^{Exp} : equals to $t_i^E + \max_m\{c_{im}^{Exp}\} + r_1^{Exp}$,
- d_i^{Nor} : equals to $t_i^E + \max_m\{c_{im}^{Nor}\} + r_1^{Nor}$,
- r_2^{Exp}: equals to a random value based on exponential distribution between 0.05 to 0.1,
- r_2^{Nor}: equals to a random value based on Normal distribution between 0.05 to 0.1,
- crt_{ij}^{Exp} : a random value between $[0, 0.05 \max_m\{c_{im}^{Exponential}\}]$,
- crt_{ij}^{Nor} : a random value between $[0, 0.05 \times \max_m\{c_{im}^{Normal}\}]$.

The parameters of genetic algorithm were set to 0.7 for crossover, 0.3, and 1000 for number of generation.

5. Experiments

Here, we have designed some experiments based on previous sections.

5.1. Experiment 1

The first experiment is taken from [4]. In this experiment we have some information as well as it is shown in Table 4 that has been created by the P-Method (same as our method).

Table 4. Data set of experiment 1

i	$pre^*(\tau_i)$	c_{im}			d_i
		c_{i1}	c_{i2}	c_{i3}	
1	8	5	3	10	13
2	6	3	7	12	17
3	4, 5	3	4	1	12
4	6, 7, 8	2	16	6	12
5	6, 10	12	2	4	27
6	9	2	4	7	24
7	-	2	15	4	13
8	-	3	5	4	18
9	10	5	5	8	27
10	-	1	5	6	29

We divided this experiment into two parts. In first part we scheduled it without considering cache reload time, in Figure 9. In our method, the best answer as the completion time (f_1) is 13, total tardiness (f_3) is 0, and number of applied processors (f_2) is 3 whereas report of this experiment in [4] is as completion time; 15, total tardiness is 6 and minimizing of number of processors is not their objective (Figure 9-A). In addition, another run of our proposed algorithm is shown in Figure 9-B. For this run; $f_1 = 18$, $f_2 = 2$, $f_3 = 0$ whereas in an optimistic way, [4] has reported $f_1 = 18$, $f_2 = 3$, $f_3 = 2$. In Figure 10 has been shown a comparison between our proposed method and the proposed method in [4].

In the second part with respect to cache reload time, some obtained results are shown in Figure 11 and two suboptimal schedulers are shown in Figure 12. In Figure 11 we computed average of completion time of best obtained scheduler in 50 iterations of experience with considering and nonconsidering cache reload time when population size increases.

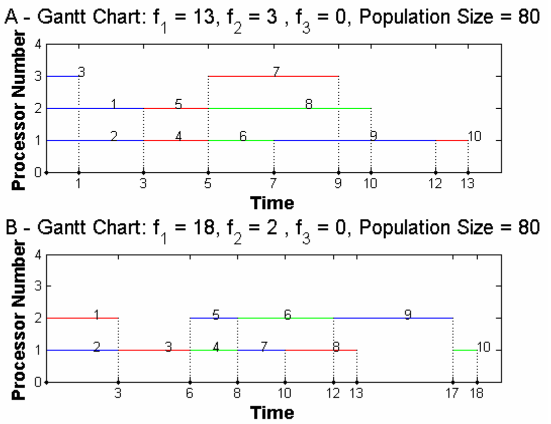

Figure 9. Experiment 1 without cache reload time

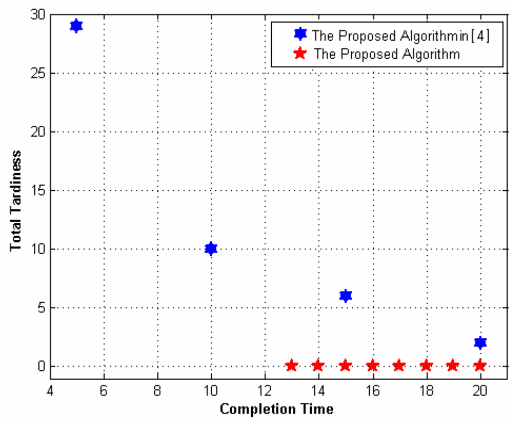

Figure 10. A comparison between our proposed method and the proposed method in [4]

5.2. Experiment 2

Second experiment was taken from [4] too. Execution time of each task on processers is given in Table 5, and for testing of local optimal problem we duplicate these processors in Table 6.

Figure 13 shows three instances of the best output answers that they describe the best answer of our proposed method according to Table 5 without considering cache reload time.

In Figure 14 has been shown 2 schedulers considering cache reload time. It is obvious scheduler with 3 processors has a lower completion time than scheduler with 4 processors. In fact this Comparison shows importance of considering cache reload time. An overall result has been presented in Figure 15.

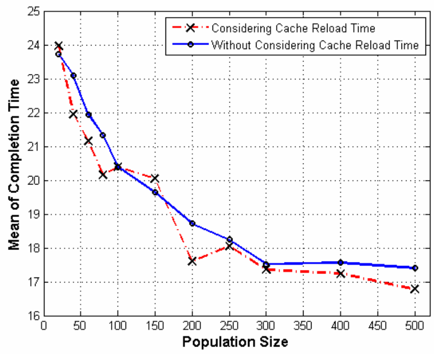

Figure 11. A comparison of completion time considering and nonconsidering cache reload time

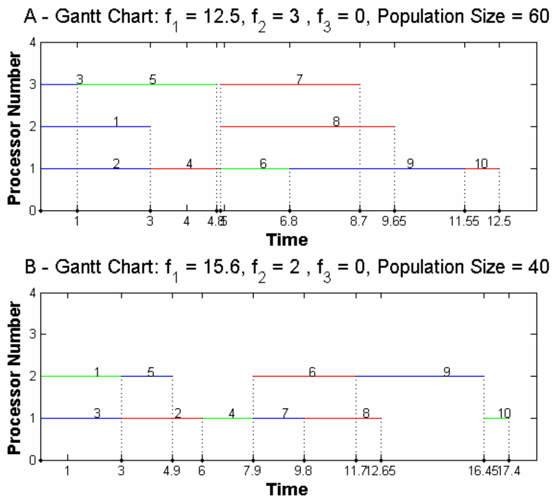

Figure 12. Experiment 1 with cache reload time

Table 5. Data set of experiment 2

i	$pre^*(\tau_i)$	c_{im}				d_i
		c_{i1}	c_{i2}	c_{i3}	c_{i4}	
1	8	5	3	11	8	19
2	6	6	5	4	13	9
3	4, 5	11	8	6	7	18
4	6, 7, 8	10	13	5	6	37
5	6, 10	10	13	8	11	38
6	9	2	11	11	3	37
7	-	3	10	11	8	44
8	-	4	12	10	5	30
9	10	6	9	7	10	37
10	-	11	12	6	4	58

Table 6. Data set of experiment 2 with duplicated

		processors								
i	$pre^*(\tau_i)$	c_{im}								d_i
		c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	c_{i7}	c_{i8}	
1	8	5	3	11	8	5	3	11	8	19
2	6	6	5	4	13	6	5	4	13	9
3	4, 5	11	8	6	7	11	8	6	7	18
4	6, 7, 8	10	13	5	6	10	13	5	6	37
5	6, 10	10	13	8	11	10	13	8	11	38
6	9	2	11	11	3	2	11	11	3	37
7	-	3	10	11	8	3	10	11	8	44
8	-	4	12	10	5	4	12	10	5	30
9	10	6	9	7	10	6	9	7	10	37
10	-	11	12	6	4	11	12	6	4	58

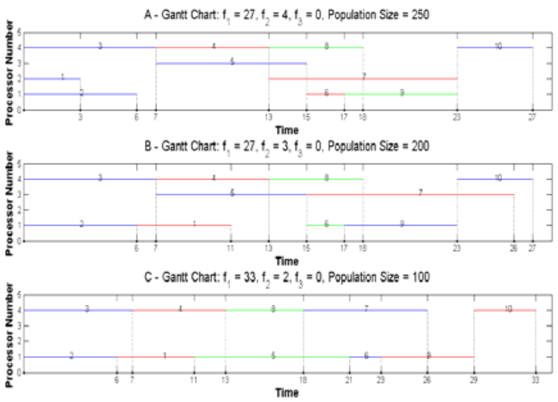

Figure 13. Schedulers for experiment 2 considering Table 5 and without considering cache reload time

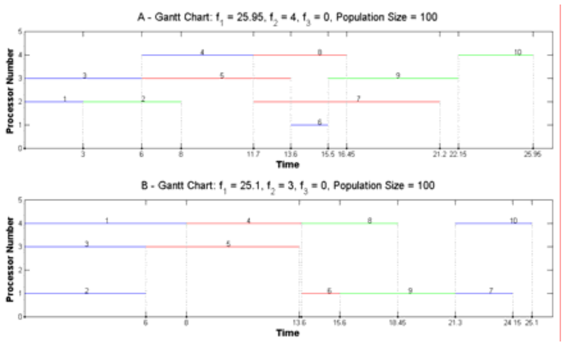

Figure 14. Two best schedulers considering cache reload time

In this part for testing of optimal locality of proposed algorithm we duplicated the assumed processors from Table 5 to Table 6. Overall results has been shown in Figure 16 which this figure declares not only considering cache reload time in scheduling can cause using lower processor and decreases completion time simultaneously but also our proposed algorithm could gain obtained results when we use data set Table 5 without duplication.

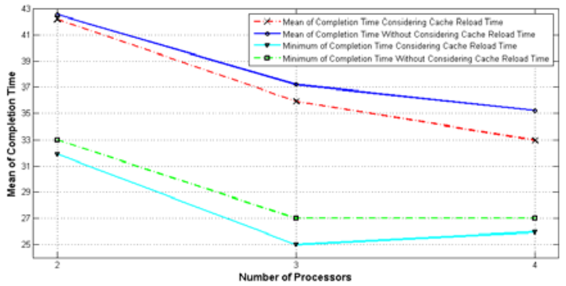

Figure 15. Comparisons between obtained completion time considering and nonconsidering cache reload time

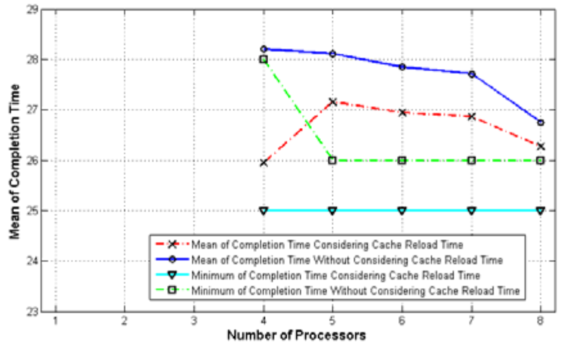

Figure 16. Comparisons between obtained completion time considering and nonconsidering cache reload time

6. Conclusion and Future Works

In this paper we have tried generalization on latest works ([4] – [6]) and covered their shortcomings. Paying attention to cache reload time in heterogeneous real-time systems is one of the aspects of this work.

Improving the encoding procedure which has large influence on the proposed scheduler is part of this paper. For this part, we designed a suitable encoding procedure based on topological sort. Same chromosome like [4] has been used to be able for comparing the proposed algorithm and the proposed algorithms in [4].

Also, trying to minimize the number of heterogeneous processors while all deadlines are met is done by genetic algorithm, whereas, in [5] this work is done by some extra local improvement heuristics moreover we have considered heterogeneous processors as oppose to [5] which assumes homogeneous processors. So our proposed algorithm is a generalization on [5].

Unlike [6], designed chromosome is simple and efficient, has fewer limitations, and while using limited information can minimize conflicted objective simultaneously.

For future works, we will try to design some better stopping conditions and some improvements on convergent conditions too.

7. References

[1] CM. Krishna and GS. Kang, *Real-time systems*, New York, McGraw-Hill, 1997.

[2] P. Marwedel, *Embedded System Design*, Netherland, Springer, 2006.

[3] J. Du and J. Y. T. Leung, "Minimizing total tardiness on one machine is NP-hard," *Mathematics of Operational Research 15*, pp. 483–495, 1990.

[4] M. Yoo and M. Gen, "Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system," *Journal of Computers & Operations Research 34*, pp. 3084-3098, 2007.

[5] J. Oh and C. Wu, "Genetic-algorithm-based real-time task scheduling with multiple goals," *Journal of Systems and Software*, pp.245-258, 2004.

[6] H. Mitra and P. A. Ramanathan, "Genetic approach for scheduling non-preemptive tasks with precedence and deadline constraints," *In: Proceedings of the 26th Hawaii international conference on system sciences*, pp. 556–64, 1993.

[7] M. Lin and L. Yang, "Hybrid genetic algorithms for scheduling partially ordered tasks in a multi-processor environment," *In: Proceedings of the sixth international conference on real-time computer systems and applications*, pp. 382–87, 1999.

[8] Y. Monnier, J. P. Beauvais and A. M. Deplanche, "A genetic algorithm for scheduling tasks in a real-time distributed system," *In: Proceedings of the 24th euromicro conference*, pp. 708–14, 1998.

[9] A. J. Page and T. J. Naughton, "Dynamic task scheduling using genetic algorithm for heterogeneous distributed computing," *In: Proceedings of the 19th IEEE international parallel and distributed processing symposium*, pp. 189.1, 2005.

[10] M.K. Dhodhi, I. Ahmad, I. Ahmad and A. Yatama, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing 62*, pp. 1338–61, 2002.

[11] M. Gen and R. Cheng, *Genetic Algorithm and Optimization Engineering*, John Wiley and Sons, INC, New York, 2000.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, New York, McGraw-Hill, 2nd Edition, 2001.

[13] S. Al-Sharaeh, and B. E. Wells, "A comparison of heuristics for list schedules using the box-method and P-method for random digraph generation," *In: Proceedings of the 28th Southeastern symposium on system theory*, pp. 467–71, 1996.

[14] M. Cosnard, M. Marrakchi, Y. Robert, and D. Trystram, "Parallel Gaussian elimination on an MIMD computer," *Journal of Parallel Computing*, Vol. 6, No. 3, pp. 275-296, 1998.

[15] M.Y. Wu, and D.D. Gajski, "Hypertool: A programming aid for message-passing system", *IEEE Trans. Parallel and Distributed Systems*, Vol. 1, No. 3, pp. 330-343, 1990.