ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/350547703

MOGATS: a multi-objective genetic algorithm-based task scheduling for heterogeneous embedded systems

Article *in* International Journal of Embedded Systems · January 2021

DOI: 10.1504/IJES.2021.113811

CITATIONS
7

READS
177

2 authors, including:

Mohsen Raji
Shiraz University
78 PUBLICATIONS **384** CITATIONS

SEE PROFILE

All content following this page was uploaded by Mohsen Raji on 13 August 2021.

The user has requested enhancement of the downloaded file.

MOGATS: A Multi-Objective Genetic Algorithm based Task Scheduling for Heterogeneous Embedded Systems

Mohaddaseh Nikseresht
School of Electrical & Computer Engineering
Shiraz University
Shiraz, Iran
m.nikseresht@cse.shirazu.ac.ir

Mohsen Raji
School of Electrical & Computer Engineering
Shiraz University
Shiraz, Iran
mraji@shirazu.ac.ir

***Abstract*— Multi-objective optimization is an unavoidable requirement in different steps of embedded systems design, including task mapping and Scheduling. In this paper, a new Multi-Objective Genetic Algorithm-based Task mapping and Scheduling (abbreviated as MOGATS) is presented for heterogeneous embedded system design. In MOGATS, the architecture of the hardware platform and the set of tasks in the form of a task graph are assumed to be given as the inputs. Task mapping and scheduling problems are modeled as a genetic algorithm-based optimization approach in which the execution time, energy consumption, and reliability of the Scheduling are considered as the objectives of the optimization method. In MOGATS, we are interested in finding the Pareto frontier of the solutions (scheduled tasks) in order to help the designer to pick out the best outcome according to different design considerations. The experimental results on real application task graphs show that, MOGATS gains a better solution than the greedy algorithm if it is applied as a single-objective task scheduling method. Moreover, the superiority of MOGATS in comparison to the state-of-the-art single- and multi-objective task scheduling algorithms are shown in terms of standard performances factors such as Scheduling length Ratio and Speed up parameters. In summary, our task scheduling tool is the first multi-objective task scheduling in the design stage of Embedded Systems to help the designer to figure out which set of scheduling would provide their desired outcome. Additionally, In comparison to EGA-TS, the state-of-are task scheduling algorithm, in terms of speedup and SLR, we have gain 27.8 and 28.6 immediate improvements respectfully.**

Keywords—Embedded Systems, Task Scheduling, Multi-Objective Optimization, Genetic Algorithm.

I. INTRODUCTION

Embedded systems are widely used in industrial, military, and commercial services and products [1]. These various fields of application have led to more heterogeneity and ever-increasing complexity in modern embedded systems making the design of these systems a critical engineering challenge [2]. One of the main parts of this challenge includes the problem of mapping and scheduling of tasks during the design time of embedded systems [2]. In the mapping and scheduling problem, the functionality of the embedded system is partitioned into a series of software tasks represented as a task graph, and the order of the executing tasks should be determined based on the existing hardware infrastructure [3].

Embedded systems are typically faced several constraints such as energy consumption, execution time, and reliability [4]. These constraints may be conflicting in some cases; e.g., the execution time and the energy consumption are contradictory because less energy consumption requires less operating voltage leading to higher execution time [4]. Hence, all design stages, including task scheduling step, should be solved by simultaneously considering these parameters in the form of a multi-objective optimization problem so that a system designer can apply the desired balance between various design options [5].

The task scheduling problem is an NP-hard problem [6]; i.e., by increasing the dimensions of the problem, it is not possible to reach the best answer at a reasonable time. Hence, it is required to use heuristic algorithms to find solutions that are as close as possible to the global optimum solution. Genetic Algorithm, inspired by the evolution of humans, is capable of solving the optimization problem with the help of applying genetic operators after modeling the problem in the form of chromosomes. Due to the efficiency and yet simplicity of implementation, the genetic Algorithm has been widely used to solve various multi-objective optimization problems. We need to emphasize again that our tool is working in the stage of Designing Embedded Systems. There is no Operational System at this level, only the set of expected tasks and the hardware platform. It helps the designer to figure out whatever the set of tasks can be scheduled on the demanded hardware in the desired indicators or not and make any required change in the hardware platform or set of tasks if necessary. Evaluation tools in the first stage with detailed system analytics enable designers to discover different design options that can provide the desired design indicators. Design decisions made at an early stage of the design process are critical to preventing potentially costly changes to more advanced stages of the process. As a result, the designer should provide the necessary design, analysis, and evaluation at an early stage of the design to ensure that the resulting embedded system meets the desired performance concerning the design parameters desired.

II. RELATED WORKS

Several works based on genetic algorithms have been used to solve the problem of mapping and task scheduling during the design of embedded systems. We can separate them based on the fact if the input task graph is static or dynamic. In case they are working on dynamic task graphs like works presented in [7][8] The task graph dynamically changes. However, as mentioned before, we are working in the design stage of Embedded Systems with static task graphs. Those works may

not be related. In the field of static task graphs we can mention following works:

The task graph Embedded system [9], a multi-objective genetic algorithm for the synthesis of hardware/software of distributed embedded systems, is presented for the first time. In this work, the system is synthesized considering the power consumption and the execution time of the tasks. However, the reliability parameter is neglected in this work. In [10], an approach for mapping and scheduling an embedded system based on a genetic algorithm is presented. In this work, the run-time and cost are considered. Additionally, in order to improve the reliability of the design, the scheduling problem is applied to the modified task graph in which redundant tasks are added to the task graph. This approach does not compute the reliability of the original task graph in the scheduling problem, but instead, the scheduling is done on the modified graph by imposing a high cost. In [11], an integrated framework for the synthesis of embedded systems is presented at the system level. In this framework, the genetic algorithm applied to optimize the multi-objective mapping parameters and various architecture. However, this framework does not consider the design space with respect to parameters such as energy consumption and reliability. In another category of works, different genetic algorithms are proposed to optimize the execution time of tasks during scheduling for embedded systems. In these works, important parameters of the embedded systems, including energy consumption and reliability, are not considered. In [10], the problem of scheduling is solved, with the aim of reducing energy consumption. However, the execution time of the tasks and reliability metric are not considered. There are also some works such as [35], [36], and [37] which have developed a modeling and simulation environment for the efficient design space exploration of heterogeneous embedded systems. However, these works have focused on mapping problem and neglected to address the scheduling problem. Moreover, these works did not consider the reliability aspect either. In MOGAC [30], the scheduling problem in the real-time heterogeneous distributed systems is considered. However, it did not also address the reliability metric. Previous works have serious limitations; i.e. some previous works only deal with the problem of mapping tasks or in some works, the parameters of the execution time, energy consumption, and reliability are not considered simultaneously.

In this paper, a Multi-Objective Genetic Algorithm-based Task mapping and Scheduling method called MOGATS is presented for the design of embedded systems. In MOGATS, the tasks, which are modeled as a task graph, are mapped and scheduled on the existing hardware architecture. To this end, task mapping and scheduling solutions are modeled in the form of chromosomes, while each chromosome is evaluated using different objective functions considering the parameters of execution time, energy consumption, and reliability of the scheduled tasks in the embedded system. Using the well-known Non-dominated Sorting Genetic Algorithm II (NSGA II) algorithm, the multi-objective genetic algorithm optimization method finds Pareto frontier solutions (i.e. scheduled tasks) during the task scheduling optimization flow. The experimental results on real applications show that, MOGATS achieves better solutions comparing to the greedy algorithm when considering each objective separately. Moreover, MOGATS outperforms other single- and multi-objective heuristic task scheduling algorithms in terms of common performance metrics such as Scheduling Length Ratio and Speed up.

The rest of the paper is organized as follows: In Section III the proposed methodology is explained and the model used for tasks and hardware architecture is clarified, additionally, a developed genetic algorithm is presented to solve the problem of mapping and task scheduling in embedded systems. In Section IV, experimental results are presented and, finally, the paper is concluded in Section V.

III. MOGATS: The Multi-Objective Genetic Algorithm-based Task Scheduling Method

This section presents the proposed Multi-Objective Genetic Algorithm-based Tasks Scheduling Approach called MOGATS. At first, tasks and hardware architecture models are described and then, the design space exploration approach based on MOGATS is presented.

A. Task set and hardware platform architecture modelling

For modelling tasks, the model introduced in [13] has been used. This model is based on the Kahn Process Network (KPN), a very well-known computational model for modelling embedded systems. The KPN is represented by a directed graph $G_{AP}(V_{AP}, E_{AP})$ in which a vertex $V_i \in V_{AP}$ for $i = 1, \dots, |V_{AP}|$ represents a task or process and, for each vertex V_i, the set of tasks associated with this vertex are shown in the form of $E_i = \{e_j \epsilon E_{AP}\}$. In a case where Vi is assigned to a hardware component in a scheduling, ht_i represents the execution time on this hardware component. If it is possible for a task to be run on multiple hardware processing cores (i.e. FPGA or ASIC), its execution time is shown as the set of $ht_i = \{ht_{i1}, ht_{i2}, \dots, ht_{iU}\}$ where U represents the number of hardware processing cores on which a specific task can be executed. On the other hand, when a vertex is assigned to a software processing element (i.e. CPU or DSP) for scheduling, st_i represents the execution time on that software processing element. When it is possible to run a task on multiple software elements, the execution time is shown as the set of $St_i = \{st_{i1}, st_{i2}, \dots, st_{iV}\}$ where V represents the number of software processing elements.

Each edge $e_j \in E_{AP}$ for $j = 1, \dots, |E_{AP}|$ represents the relationship between two tasks of the task graph G_{AP}. If this link is allocated to a memory, mt_j represents the memory access time. If during scheduling, the edge can be allocated to multiple memories, the corresponding access time is represented as to the set of $mt_j = \{mt_{j1}, mt_{j2,}, \dots, mt_{jW}\}$ where W denotes the number of memories that the task can be assigned. Also, the architecture model is represented as the graph $G_{AR}(V_{AR}, E_{AR})$ in which the V_{AR} and E_{AR} respectively represent the architectural components and the communication links between them. A set of architectural components consists of two independent sets: a memory set (M), as well as a set of processors (P) that includes hardware and software components; i.e. $V_{AR} = P \cup M$. A delay between communication links is indicated by lt_{pq}, where $p, q \in \{1 \dots |E_{AR}|\}$. The energy consumption related to running an task on a processor p is shown as w_{pe} while w_{me} and w_{le} indicates the energy consumption of memory m and

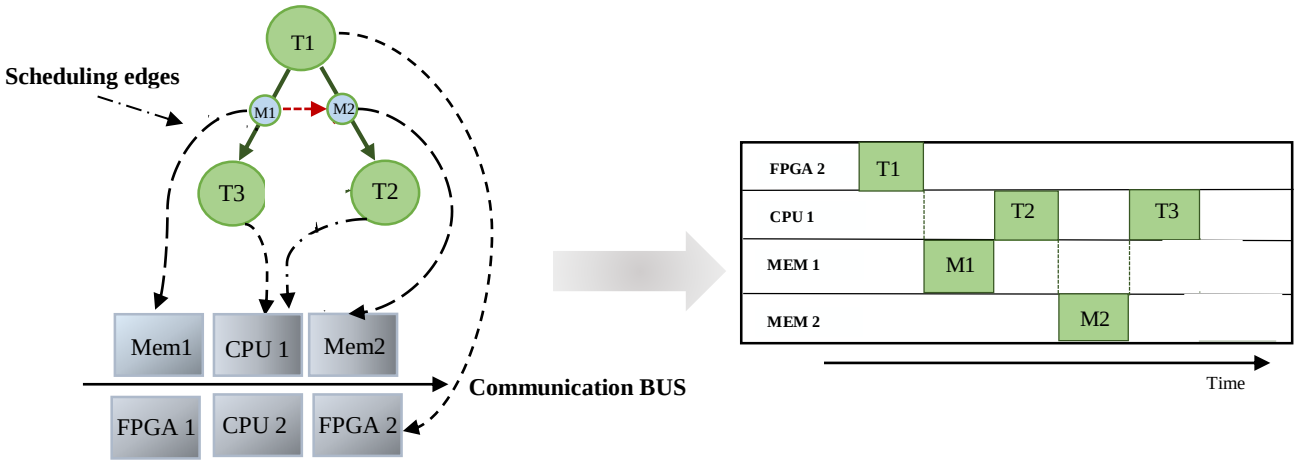
communication link. In this paper, it is assumed that the architecture platform is available because the main goal is not to solve the problem of determining the ideal architecture, but to propose a method for solving the problem of mapping and scheduling in embedded systems.

The problem of mapping and scheduling is finding the optimal assignment of tasks and communications between them on the hardware architecture platform. An example of mapping and scheduling tasks on the hardware platform is shown in Fig.1. As shown in the figure, each vertex of the graph represents a task, and each edge indicates a relationship between two tasks. Fig. 1(a) shows a task mapping in which tasks are allocated to processors from the hardware architecture platform, and communication links are mapped to memories. Fig. 1(b) illustrates the task scheduling step in which the orders of task execution/communication on the processing elements/memories are determined.

In the following, the proposed approach called MOGATS is introduced for exploring the best mapping and scheduling concerning the critical objectives of the embedded systems, including execution time, energy consumption, and reliability.

B. Design space exploration using MOGATS

Due to the complexity and limitations in the design of embedded systems, the design space exploration is a challenging process and known as an NP-hard problem [14]. Hence, it is required to use a multi-objective heuristic algorithm to guide the designer to make the best scheduling and mapping decisions for the tasks. In MOGATS, the problem of task scheduling is modelled using a genetic algorithm. At first, a task scheduling solution is modelled in forms of a chromosome and then, using genetic operators, new population (task scheduling solutions) are generated. Then, the objectives (i.e. execution time, energy consumption, and reliability) are defined, and the new population is formed using a selection approach. In the following, the details of each step are described.



a. Mapping

b. Scheduling

Figure 1. The Problem of mapping and scheduling in the design of Embedded Systems.

C. Segmentation Method

In this work, in order to handle the dependency between tasks in the task graph, we have used a segmentation approach described in [12], in this method with considering the dependency graph, the segmentation method is used to divide scheduling into several segments. Each segment should cover independent tasks that allow for parallel execution on different processors [12]. Each element of the segment shows a task and a processor to which it is assigned. The feature of the segmentation method is that the tasks in each segment are independent. However, the segments must execute in order. Figure 2 shows the segmentation result from the task graph in figure 1. As an example, in Seg1, task T1 is executed on FPGA F2. After completing execution of Seg1 task list, Seg2 can execute its task list. Notice that, for a specific task graph, each segment indexes are similar in all chromosome sets, which is based on the task graph. However, the assigned cores (the inputs of each segment) are the outputs of a random function which randomly assigns cores from the core domain to each task. We have chosen this method because, in single-objective optimization, this simple technique brings a significant improvement in comparison to other methods to deal with dependency graphs.

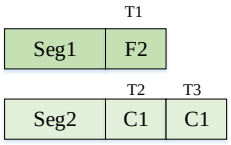


Figure 2. The Segmentation result from the task graph in figure 1

D. Chromosome Representation

The processor allocation and scheduling order of tasks on processors and communication edges are coded within each chromosome. Given a task graph G with N tasks and M communication nodes, and system architecture with P processors, a chromosome is described as a set of 2N + M genes. The structure of a representative chromosome is presented in Figure 3.

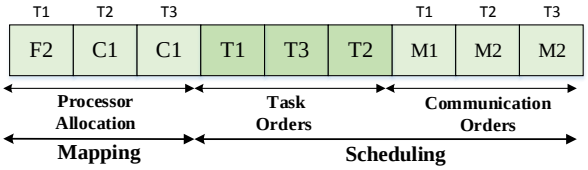


Figure3. Chromosome Representation

1) Mapping part in which the allocation of the tasks to the processing elements is shown (the first N genes), corresponding to the mapping N tasks to P available processors. For example, task T_2 is mapped on processor C_1. The point here is different processors are mapped to the same task orders shown as processor allocation indexes. The task orders are based on segmentation order (Figure 2) and this order is the same for all chromosome sets. However, the input of each element (the assigned cores) is the output of random function which randomly assigns core from the core domain to each task in the first population. Appendix A shows the details on how to recover the processor allocation (mapping part) from the task graph segmentation.

2) **Scheduling part** in which the remaining N+ M genes represents the order of execution tasks on the allocated processors in the first sub-part and the order of the communications to the communication links (i.e. memories in this example) in the second sub-part. In the first sub-part, we first fill the genes with task order segmentation (initialization state), and then, we randomly reorder the task placements in each segment by calling reorder function to generate other chromosomes in the first population. In the second sub-part in which the order of the communication nodes on the memories is coded, the procedure is similar to the mapping part. Appendix A shows the details on how to recover the processor allocation (scheduling part) from the task graph segmentation.

3) Genetic operators

Genetic operators are used in a genetic algorithm to generate and maintain genetic diversity from one generation of a population to the next (mutation operator) and to combine existing solutions into others (crossover operators). In the following, the genetic operators, which are used in MOGATS, are described.

1) *Crossover*

As one of the genetic operators, the crossover is used to combine the genetic information of two parents to generate a new child. In MOGATS, at first, one of the parents is selected randomly, and a local crossover mechanism proposed in [33] is applied in order to prevent local optimums. According to this mechanism, chromosomes are classified based on NSGA II Pareto set classification [17] and then, the other parent is randomly selected from the same class of the first one; i.e. if the first parent is selected from the Pareto set class 1, the second praetor will also be selected from the Pareto set class 1. After the parent's selection step, two children are generated with probability as described in the following.

In the mapping part of the child chromosome, the algorithm is based on uniform crossover [34]. According to this algorithm for each task of the child, a random number between 0 and 1 is generated. If the number is lower than P_C, the mother's processor will be selected to execute the child's task (i.e. the processor allocation of child chromosome is copied from the mother's chromosome); otherwise, the father's processor will be chosen. The pseudocode of the crossover operator in the mapping part is provided in Figure 4.b. similar steps are done for the crossover operation in the communication links scheduling part because we do not change the order of tasks, there would be no problem on correctness, completeness and uniqueness of the mapping-part are provided.

In the task order section of the scheduling part, the algorithm is based on the single point crossover [34]. A cross over point *K* is randomly selected between 0 to *N*-1 where *N* denotes the number of tasks then, we cut the mother's and father's task orders from the point *K* and copy tasks from the first part of mother's task orders to the same part of the child1's chromosome. The same will be repeated for child2 with the first part of fathers' task orders. After that, we scan the father's task order part from the beginning and copy the tasks of father's task order part which do not appear in the left side of child1's task orders to fill it. The same will happen for the child2 with mother's task orders. The pseudocode of the crossover operator for this part is provided in Fig. 4.c. In appendix B, we prove that the proposed crossover operator is a valid genetic operator as it fulfils the necessary conditions such as the correctness, completeness and uniqueness of chromosomes [16].

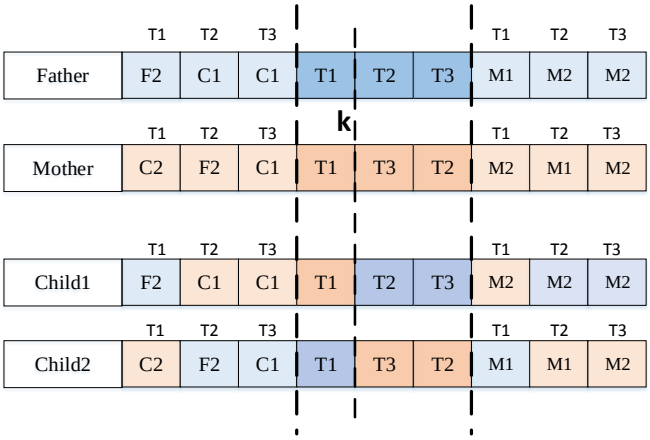


Figure 4.a Crossover operation

Inputs:

1. One parent is randomly chosen from current population
2. The other parent randomly chosen from the same class of

first parent

Output: Two new chromosomes in mapping part

For (all tasks in the task set**)**

1. Generate a random number R∈ $[0-1]$
2. **If**(R ≤ P_C)
   3. Copy mothers 'processor to child1's gen and Copy fathers' processor to child2's gen
4. **Else if**(R > P_C)
   5. Copy fathers 'processor to child's gen and Copy mother's processor to child2's gen

Figure 4.b. Crossover- mapping part Algorithm

Inputs:

1. The same parents from crossover-mapping part

Output: Two new chromosomes in scheduling part

For (all tasks in the task set**)**

2. Random cross over point K between [1, N-1] is chosen
3. The initial selected tasks from Mother [0 to K] are directly transmitted to the child1 and tasks from Father[0 to k] are directly transmitted to child2
4. To fill [k+1 to N-1], fathers chromosomes are scanned from beginning and each node that is not yet in the child is added to the next empty position of child. The same will be repeated with mother's chromosome for child2.

Figure 4.c. Crossover- scheduling part Algorithm

Figure 4. The cross over operation and its mapping and cross over algorithms

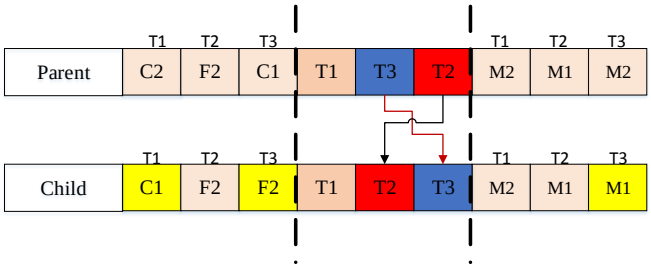
2) *Mutation*

The mutation is a genetic operator used to maintain genetic diversity from one generation of genetic algorithm to the next one. These changes should not affect the prioritization of genes. In MOGATS, a parent is chosen randomly from the current population, and one child is generated from the selected parent. These changes should not affect the prioritization of genes, and therefore, changes in each segment must take place by segmentation. The mutation operator is performed in each part of the chromosomes as described in the following.

In the mapping part, uniform mutation [34] is applied for each task of the child in which the parent's processor is selected to execute the child's task with probability P_m; i.e. a random number between 0 and 1 is generated and if the number was lower than P_m the parent's processor would be selected to execute child's task; otherwise, another processor from processor pool is chosen randomly. An example of mutation operator is shown in Figure 5.a. The pseudocode of mutation-mapping operation algorithm is provided in Figure 5.b. A similar procedure is used for the scheduling the communication links. Since the order of tasks is not changed in the applied mutation operator, the mutation operator is a valid genetic algorithm as it keeps the correctness, completeness and uniqueness of the chromosome in the mapping part.

The mutation operator in the task order section of the scheduling part in the chromosome is as follows: a task is randomly selected in each segment, and it is changed with a task of the same segment; i.e. their order can be changed. Figure 5.c shows the pseudocode of mutation-scheduling part.

In order to prove that the mutation operator maintains the order of tasks and will not result in invalid orders, a theorem is proved in appendix B.



D. Tasks Order

Figure 5.a. Mutation operation

Inputs:

1. One parent is randomly chosen from current population

Output: A new chromosome in mapping part

For (all tasks in the task set**)**

2. Generate a random number R∈ $[0-1]$
3. **If**(R ≤ P_m)
   4. Copy parent's processor to child's gen
5. **Else if**(R > P_m)
   6. One processor is randomly chosen from processor's pool
   7. Copy chosen processor to the child's gen

Figure 5.b. Mutation-Mapping part Algorithm

Inputs:

1. The same parent from mutation-mapping part

Output: A new chromosome in Scheduling part

For (all tasks in the task set**)**

2. Choose randomly a task T
3. Generate a new task orders by interchanging the task of T with a task in the same segment

Figure 5.c. Mutation-Scheduling part Algorithm

Figure 5. cross over operation and its mapping and cross over algorithms Shows the

3) *Selection*

The selection operator is based on NSGAII [17]. The pseudo-code of NSGAII is shown in Figure 6.

Input: N population size, M max number of generation

Output: Pareto frontier, non-dominated solutions in P_M

P_0 = GenerateInitialPopulation(); // Size N
$Q_0 = \emptyset$; // Start with children set empty
EvaluateObjectiveFunction(P_0); //calculate fitness
RankPopulation(P_0); //Done according to fitness values
For (i=0 to M-1) **do**
  // Create children population
  Q_i = SelectionCrossoverMutation(P_i);
  // Calculate children fitness
  EvaluateObjectiveFunction (Q_i);
  P_{i+1} = CombineParentsAndChildren(P_i, Q_i);
  RankPopulation(P_{i+1});
  //Elitism: Keep non-dominated:
  P_{i+1} = SelectNIndividuals(P_{i+1});
End for

Figure 6. Design Space Exploration based on NSGAII[17]

E. Fitness functions

In MOGATS, three different design parameters are taken into consideration while finding the best task scheduling; i.e. execution time, energy consumption, and the reliability. The fitness functions related to these design parameters used for evaluation of each task schedules are determined after specifying executing core and other related aspects of each task for a specific task scheduling [16].

1) *Execution Time*

The execution time of the critical path of a task scheduling (i.e. the path which includes the maximum executing tasks) is computed as follows:

$$\sum_{i\in V_{AP},\, i\in Path} ht_{iu}\, x_{iu} + \sum_{i\in V_{AP},\, i\in Path} st_{iv}\, x_{iv} + \sum_{j\in E_{AP},\, j\in Path} [lt_{kl} + (mt_{jw} + lt_{mn})\, x_{jw}]x_j \quad (1)$$

Where the first and the second parts of Eq. 1 respectively represent the effect of hardware processors and software processors on the execution time while the last part shows the effect of the links delay due to memory access time if the task communication j is assigned to a memory core. The parameter mt_{jw} illustrates the memory access time for $w \in \{1, \dots, M\}$ in which M is the maximum number of memories. Additionally, lt_{kl} illustrates the delay of communication link between processors k and l with $k, l \in \{1, \dots, |V_{AR}|\}$. Finally, the variable lt_{mn} represents the communication delay among two cores m and n where $m, n \in \{1, \dots, |V_{AR}|\}$.

In order to model which task is assigned to each processor, the decision variables are used. In more details, when task i is assigned to the hardware processor u or software processor v, decision variables x_{iu} and x_{iv} are set to 1, respectively; otherwise, they are set to 0. On the other hand, in order to show that a communication link j is linked to a memory core w, or a communication channel j is included within a core, decision variables x_{jw} and x_j are set to 1, respectively; otherwise, they are set to 0. Note that if two communicating tasks are assigned to the same processor x_j value will be zero.

2) *Energy consumption*

The fitness function used for computing energy consumption is as [13]:

$$min\left\{\sum_{i\in v_{AR}} t_p^e\, w_{pe} + \sum_{j\in E_{AR}} (\, t_l^c w_{le} + t_m\, w_{me})\right\} \quad (2)$$

Where t_p^e shows the execution time of the processor p and w_{pe} illustrates the power consumption per unit of this processor while t_i^c and t_m respectively determine the time elapsed on the communication link and memory. Moreover, w_{le} and w_{me} indicate power consumption by communication link and memory, respectively.

3) *Reliability*

The third fitness function is related to computing the reliability of the task scheduling. The system's reliability model introduced in [16] and [17] is adopted in MOGATS to compute the reliability of the task scheduling. This fitness function is based on Discrete-Time Markov Chain (DTMC) of the system architecture which are graphical models consisting of finite state graphs. A user-oriented reliability model has been developed to measure the reliability of service that a system provides to a user community. It has been observed that in many systems, especially software systems, reliable service can be provided to a user when it is known that errors exist, provided that the service requested dose not utilize the defective parts. The reliability of the system, therefore, depends both on the utilization of the components and the probabilistic distribution of the utilization of the components to provide the service. This user-oriented software reliability figure of merit is defined to measure the reliability of the system with respect to a user environment. A simple Markov model is formulated to determine the reliability of a software system based on the reliability of each individual module and the measured inter modular transition probabilistic as the use profile. The reliability of the program can be calculated from the following procedure in the critical path. Let $\{N_1, N_2, \dots, N_n\}$ be the set of nodes in the program graph with $N_1$the entry node and N_n the exit node. Let R_i be the reliability of node N_i and P_{ij} the transition probability of the branch (N_i, N_j). Let P_{ij}= 0 if the branch (N_i, N_j) does not exist. However In the critical Path all Tasks must be taken so the P_{ij}= 1. Let the transition matrix be P' where $P'(i,j)$ represents the probability of transition from state i to state j in the Markov process. For any positive integer n, let the nth power of P' be P'^n.

Evidently, $P'^{n}(i,j)$ is the probability that starting from state i, the chain enters the absorbing state $j \in \{C, F\}$ at or before the n[th] step. The reliability of the program R is the probability of reaching state C (correct termination) from the initial state N_1. Hence, we have

$$R = P'^{n}(N_1, C) \tag{3.a}$$

Let S be an n by n matrix such that:

$$S = 1 + Q + Q^2 + Q^3 + \cdots = \sum_{k=0}^{\infty} Q^k \tag{3.b}$$

If Q is finite, which is the case here, and we let $W = 1 - Q$, it can be shown that [20]

$$R = S(1,n) * R_n \tag{3.c}$$

where S is the base matrix of the DTMC, *S(i, j)* shows the expected number of times to visit state *j* from the base state *i*, and parameter *n* is the number of states in the model, and $R_n$indicates the reliability of last state *n*. Since a better task scheduling is the one which has less execution time, less energy consumption, and higher reliability, in order to align this fitness function with the previous ones, unreliability (*1-R*) is considered in MOGATS; i.e.

$$Unreliability = 1\text{-}R \tag{4}$$

F. Solving the multi-objective optimization problem

After defining fitness functions in the previous section, the general form of the multi-objective optimization problem which is the task scheduling problem here can be written as follows [17]:

$$\min_{x} z = f(x) = \left(f_1(x), f_2(x), f_3(x)\right)^T \quad s.t.\ x \in X \tag{5}$$

Where x shows a possible solution from the set of all feasible solutions *X*. In MOGATS, a possible solution is a task scheduling which is determined by the decision variables discussed before and completely depends on how application tasks are mapped and scheduling onto the processors of the architecture platform. For a specific solution (i.e. a scheduling of tasks), the fitness functions f_1, f_2, and f_3 calculate the value of design parameters from Eq. (1), (2), and (4). The overall fitness function of $z = f(x)$ translates a solution x from the decision space determined by the decision variables into a point in the objective space defined by the three objectives or fitness functions. However, in multi-objective optimization, there does not typically exist a feasible solution that minimizes all objective functions simultaneously as the design parameters are antagonistic [19][20]; i.e. finding a design parameter decreases at least one of the other design parameters. So, we are interested in finding so-called Pareto optimal solutions; i.e. the solutions which are non-dominated by any other solution point among all solutions from the feasible set [17].

In order to obtain Pareto optimal solutions, multi-objective evolutionary algorithms such as genetic algorithm can be used due to the capability of these algorithms to optimize several independent objective functions simultaneously. In this paper, NSGAII algorithm [17] has been used as it has shown that, in contrast to other evolutionary algorithms, NSGA II has several advantages such as ease of implementation and low computing power [17]. The pseudocode of this algorithm is shown in Fig. 6. NSGA II starts with a randomly generated population (i.e. different solutions for task scheduling) and then, iteratively generates new populations from the previous one using genetic operators (i.e. crossover and mutation). The aim is to use these operations to produce a more appropriate set of non-dominated results in each iteration. The algorithm terminates when the required number of iterations are passed. The final set of Pareto solutions are the Final solutions for task mapping and scheduling problem.

IV. EXPERIMENTAL RESULTS

The proposed mapping and scheduling method (MOGATS) is implemented in C programming language and run on a LINUX machine with Intel Core i3 with 2.30 GHz clock frequency and 2 GB RAM. For simulation, we use four test-cases, the first two test cases are in the domain of automotive software; i.e. Anti-lock Brake System (ABS) and Adapter Controller System (ACC) These two test cases and their failure rates are adopted from the study in [20]. The last two test cases are chosen from Embedded System Synthesis Benchmarks Suite (E3S) benchmark [15]. For E3S benchmarks, the failure rate is also adopted from the work done in [20]. The ABS and ACC are two services that are used in most of modern cars to enhance the safety as well as to assist convenient driving. The software architecture of the subsystems is depicted in [22], where includes 14 components (tasks) from (components 0–7) are mainly responsible for the ABS subsystem and (components 8–14) contribute to the ACC functionality. The two E3S benchmarks [15] Auto.indust-mocsyn consists of 21 components and consumer-mocsyn consists of 10 tasks. The hardware platform consists of 8 processors, four high-performance processors, and four energy-efficient processors the hardware detailed are adopted from [22]. The correlation between the energy consumption and reliability is also considered; i.e. the reliability of the processors with less energy consumption should be less than the ones with higher energy consumption. Some further information about task graphs and is presented in the following table.

Table 1- The number of task graphs, edges from each Bench mark has been shown.

Target Application	# of Nodes	# of Edges
ABS and ACC	14	33
E3S Benchmark-Auto. Indust	21	82
E3S Benchmark-mocsyn	10	21

The parameters of NSGA II are set as follows: the probability of crossover (P_C) has been set to 0.7 and the probability of mutation (P_m) has been set to 0.05. The number of iterations is 9 million times (the final Pareto set is achieve in 870 iterations but we continue the execution up to 9 million to make sure that, there would be no change in the final set) with population size 100.

In MOGATS, because reliability, performance, and energy consumption represent objective functions, the only constraints that we used in our problem formulation consist of the architecture platform begin given and the HW/SW partitioning of the given application. Specifically, in our case, we assume that the architecture platform has eight components in order to accommodate the largest application task graph that we investigated. It is assumed that the hardware architecture and software components (tasks graph) are given as inputs to the algorithm. In these experiments, the hardware platform includes four CPUs as software processing cores, four FPGAs as hardware components, and two communication units which are assumed to be handling through memory mapping; i.e. the source core writes into a memory component, and the destination core reads it from memory. The communication arcs in the task graph are handled in this manner. The energy consumption, reliability (in terms of failure rates) and execution time (in terms of Million Instructions per Second (MIPS)) of each processor has been adopted from [20].

A. Evaluation metrics

To compare the efficacy of MOGATS with other algorithms on standard graphs, two standard factors scheduling length Ratio (SLR) and Speedup are used.

- *Scheduling length Ratio(SLR):*

The length of the scheduled output graph is an essential factor in comparing different scheduling algorithms; we need to synchronize these graphs to get the shortest possible length in each graph. To this aim, we have used SLR, a suitable metric for comparing different algorithm which is computed as follows [23]-[31]:

$$SLR = \frac{makespan}{\sum_{T_i \in CP_{min}} min\, P_k \in p\ (W(T_i, P_k))} \quad (6)$$

Where $makespan$ or completion time is the total time taken to process a set of tasks for its complete execution, CP_{min} represents the critical path, and $W(T_i, P_k)$ indicates the time of executing task T_i on the P_k processor based on the output of the task scheduling. In fact, this criterion divides the achieved execution time ($makespan$), into the optimum execution time of the scheduling ($\sum_{T_i \in CP_{min}} min\, P_k \in p\ (W(T_i, P_k))$). To obtain a normal criterion SLR average on several graphs is considered as comparison criterion. SLR, The most important performance measurement criterion on graph scheduling algorithms is scheduling length resulted from output algorithm. Since a large number of graphs with different characteristics are used, it is necessary to normalize scheduling length to a low band for every graph so that we reach a criterion for total comparison called scheduling length ratio.

- *Speedup parameter*

Speedup metric is another standard performance metric which is obtained by dividing the time of executing all the tasks serially on the fastest processing element by the total completion time of tasks in the obtained scheduling solution; i.e. [23]-[26],[29]-[31]:

$$Speedup = \frac{min\, P_k \in p\ (\sum_{T_i \in T} W(T_i, P_k))}{makespan} \quad (7)$$

The problem of task scheduling, in the form of software scheduling programming on the hardware platform, is considered. Pareto solutions taking into account parameters in the form of unreliability, the energy consumption and the execution time in the objective space are considered. The experimental results obtained for the two test cases are shown in Fig. 7. X, Y, and Z axis of this figure respectively show the energy consumptions, execution time, and unreliability of the solutions (i.e. task scheduling). The initial populations and final set of Pareto optimal solutions have been shown in a circle (red) and triangle (blue), respectively. Comparing the first population and the final one, it can be seen that noticeable improvements in all three objectives are achieved as the final solutions have moved in the direction of the arrows shown in the figure; i.e. the better solution is the one with less execution time, less energy consumption, and less unreliability.

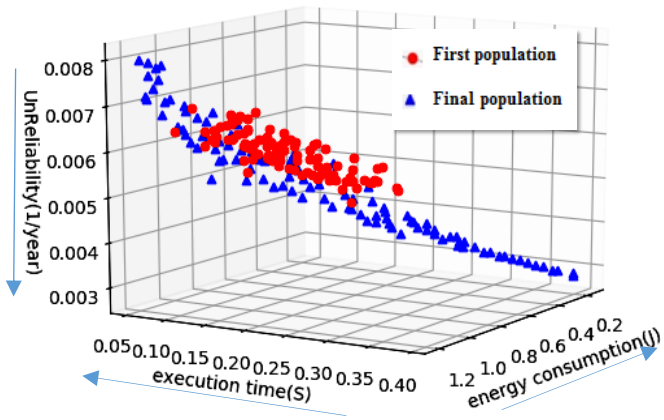


Figure 7. Comparison of the initial population (circle - red) and the

final population (triangle - blue).

In order to show the results more clearly, these results are re-displayed using two-dimensional graphs in Fig. 8. For example, Fig. 8.a compares the results of Energy consumption vs execution time in the first population and the final one. Similarly, Fig. 8.b and Fig. 8.c shows the comparison of the parameters of energy consumption vs unreliability and execution time vs unreliability, respectively. As can be seen in Fig. 8, the Pareto set of the final population in all cases dominates the first population, which means that MOGATS improve the initial task scheduling solutions considering all three fitness functions.

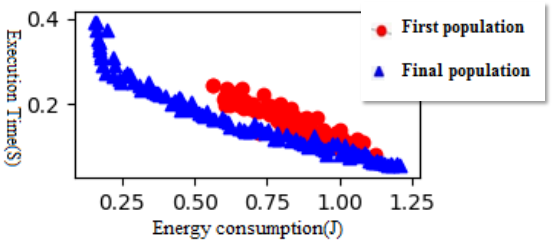


Figure 8.a. Energy consumption vs. Execution Time Graph, the comparison of initial population (circle - red) and final population (triangle - blue)

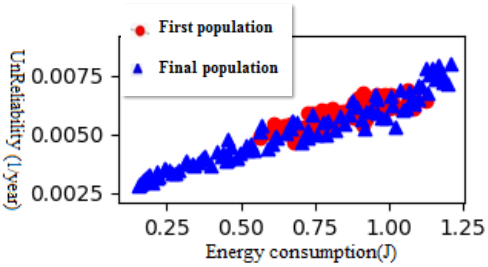


Figure 8.b. Energy consumption vs. Unreliability Graph, the comparison of initial population (circle - red) and final population (triangle - blue)

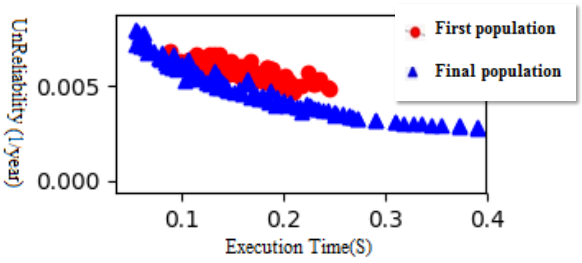


Figure 8.c. Execution Time vs. Unreliability Graph, the comparison of initial population (circle - red) and final population (triangle - blue)

Figure 8. Two-dimensional graphs for better results demonstration- the first population and the final one has been respectively shown in (circle - red) and (triangle - blue) respectfully. As it can be seen in all scenarios, the final population covers the wide range of answers.

The 3D view of E3S benchmark, Auto.indust-mocsyn, is shown in Fig. 9. For a better understanding as it can be seen in Fig. 10.a, the final population dominates the first population regarding energy consumption and execution time. Fig. 10.b/10.c shows the results regarding energy consumption/execution time vs unreliability. As power has been increased, the reliability improves the reason behind it is that in this scenario, we are using less power consumption cores. With less power consumption cores, the reliability of the system is less affected. In the negative side, the less power consumption cores usually have a longer time of execution as a result in figure 10.c as the time of execution increased the reliability of the system increased.

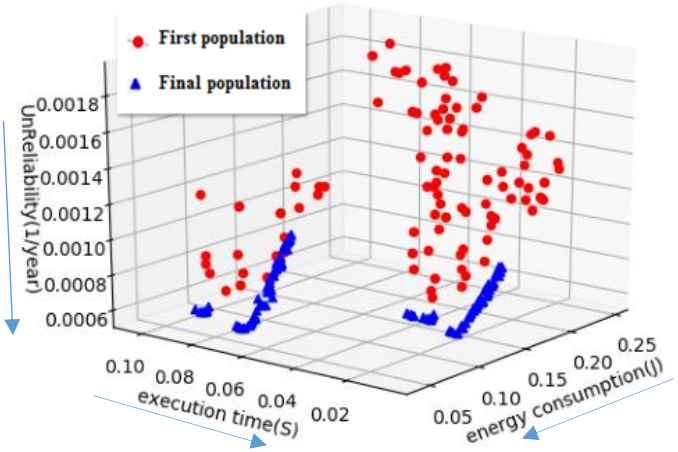


Figure9 Comparison of initialization Population (circle - red) and Final Population (triangle - blue) using MOGATS

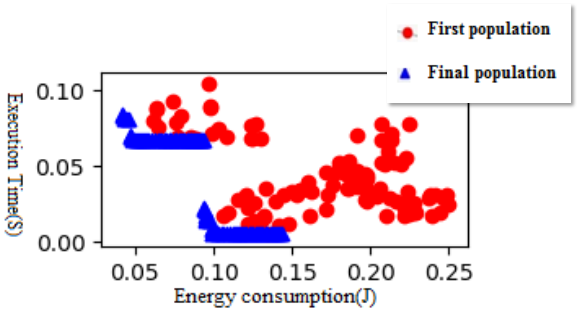


Figure 10.a. Energy consumption vs. Execution Time Graph, the comparison of initial population (circle - red) and final population (triangle - blue)

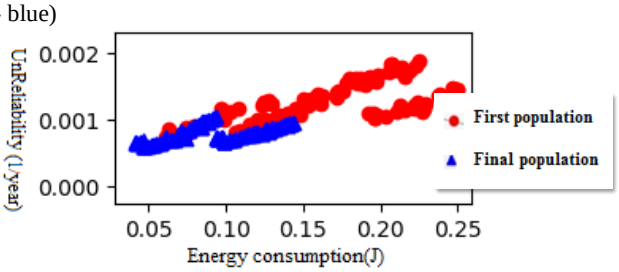


Figure 10.b. Execution Time vs. Unreliability Graph, the comparison of initial population (circle - red) and final population (triangle - blue).

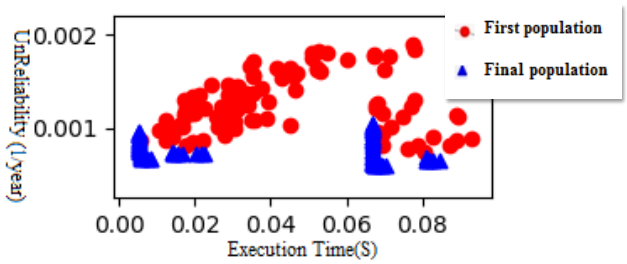


Figure 10.c. Execution Time vs. Unreliability Graph, the comparison of initialization Population (circle - red) and Final Population (triangle - blue)

Figure 10. Two-dimensional graphs for better results demonstration- the first and the final population.

The experimental results of the other E3S benchmark (i.e. consumer-mocsyn) are shown in Fig. 11 and 11. Fig. 11 shows the 3D view of the first and final population considering energy consumption, unreliability, and execution time while Fig. 12 shows the 2D view of the first and final populations. The point about this benchmark is that the overall Execution time of tasks on processors was insignificant despite the type of the processor. Thus the best and worse execution times are closed together as it can be seen in figure 12.a. Additionally, The effect of this can be seen on execution-unreliability and power-unreliability charts as well. The best and worse reliability values are extended from 0.991 to 0.999. As a result, these two charts are constant with unreliability values near zero.

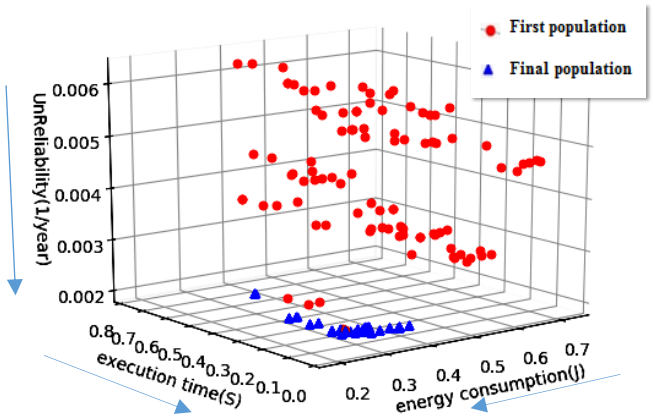


Figure 11. Comparison of initialization Population (circle - red) and Final Population (triangle - blue) using MOGATS.

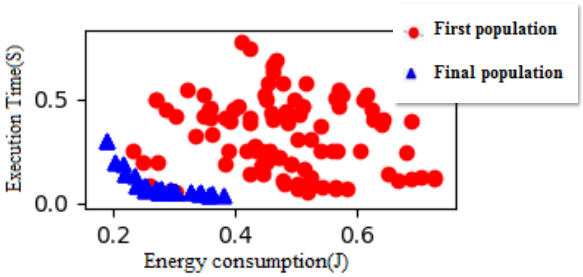


Figure12.a. Power – Execution Time Graph, the comparison of initial population (circle - red) and final population (triangle - blue)

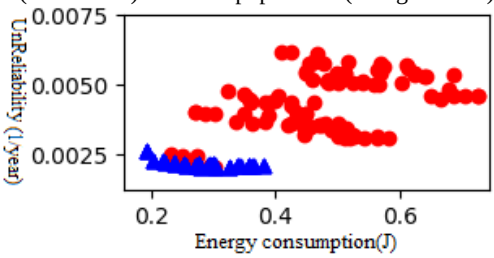


Figure12.b. Power– Unreliability Graph, the comparison of initial population (circle - red) and final population (triangle - blue)

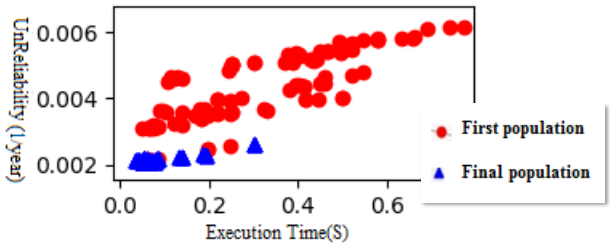


Figure 12.c. Execution Time – Unreliability Graph, the comparison of initial population (circle - red) and Final Population (triangle - blue)

Figure12. Two-dimensional graphs for better results demonstration- the first population and the final one.

In order to compare the efficacy of MOGATS with similar scheduling algorithms, 100 random graphs with size 20 nodes are created using an automatic graph generator tool that is implemented in our work. In random graphs, the costs of communications for all nodes are considered to be the same. Specially, we compare performance standards (i.e. SLR and Speedup) of MOGATS with well-known heuristic task scheduling algorithms. (i.e. EGA-TS [12], HEFT_T [29], and BGA [32]).

The SLR comparison results are presented in Figure 13. As shown in the figure, MOGATS provides the scheduling solution with an execution time equal to the optimum execution time of the scheduling according to Eq. (6) while the other scheduling algorithms achieve to SLR value of .5 by EGA-TS in the best case. Figure 14 shows the Speedup comparison results. As can be seen in the figure, MOGATS outperform other scheduling algorithms by achieving higher speedup value. The results show that, if all the tasks are serially executed on the fastest processing element, it takes six times longer than if the obtained scheduling solution is applied

for executing the task set. In comparison to the other methods, in the best case, EGA-TS provides scheduling solutions which are four times shorter than the case in which the tasks are serially executed on the fastest processing element.

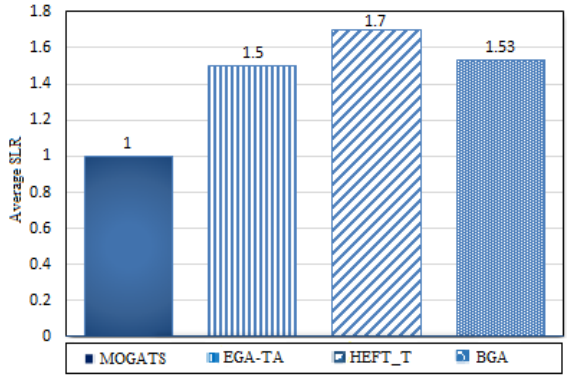


Figure 13. SLR Comparison Results between MOGATS and other task scheduling methods

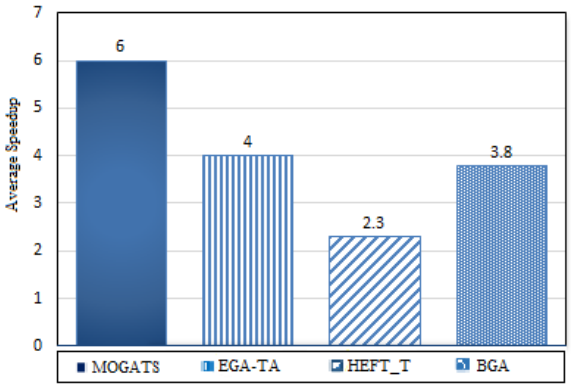


Figure 14. Speedup Comparison Results between MOGATS and other task scheduling methods

We also compare MOGATS with greedy-based task scheduling method to show its efficacy in finding the best solution in terms of each of the three objective functions (i.e. execution time, energy consumption, and reliability). To this end, we implement the greedy-based scheduling considering one the objective functions and then, the best tasks scheduling solution provided by MOGATS is selected for comparison each time. In this experiment, ABS and ACC benchmarks were used. Table 2.a, Table 2.b, and Table 2.c show that the numerical results of the obtained scheduling solution when the selected objective function in the greedy algorithm is respectively execution time, energy consumption, and unreliability. As shown in Table 1.a, MOGATS finds a scheduling solution which has similar execution time to the greedy-based method, which is even better in terms of energy consumption and unreliability. More interestingly, as shown in Table 2.b and 2.c, the task scheduling solutions found by MOGATS dominate the ones obtained by the greedy-based task scheduling approach when the greedy-based method is applied to optimize the tasks scheduling in terms of energy consumption and unreliability, respectively. The main reason for this superiority is that MOGATS takes advantages of genetic algorithm which explores the search space broader and more accurate.

Table 2-Comparison of MOGATS with greedy-based task scheduling method

a. Objective function: Execution time

	MOGATS	*GREEDY*
EXECUTION TIME (S)	0.0545	0.0545
ENERGY CONSUMPTION (J)	0.8087	0.8169
UNRELIABILITY (1/YEAR)	0.6581	0.7599

b- Objective function: Energy consumption

	MOGATS	*GREEDY*
EXECUTION TIME (S)	0.3909	0.2728
ENERGY CONSUMPTION (J)	0.1563	0.1764
UNRELIABILITY (1/YEAR)	0.4304	0.999

c- Objective function: Unreliability

	MOGATS	*GREEDY*
EXECUTION TIME (S)	0.1954	0.2727
ENERGY CONSUMPTION (J)	0.3909	0.2154
UNRELIABILITY (1/YEAR)	0.0140	0.0758

V. Conclusion

In this paper, a multi-objective genetic algorithm based task scheduling method called MOGATS is proposed for designing embedded systems considering the parameters of the execution time, energy consumption, and reliability. In MOGATS, tasks are modelled as a task graph, and according to the existing hardware architecture, a method for mapping and scheduling tasks on hardware architecture is proposed. The task mapping and scheduling problem are modelled in forms of chromosomes and using the genetic operators such as crossover and mutation and a selection approach based on NSGAII, the new populations (i.e. task scheduling solutions) are generated. The multi-objective task scheduling optimization provided by MOGATS is verified using real task graphs, and its efficacy is investigated by comparing to the well-known heuristic-based task scheduling algorithm. The use of a multi-objective optimization strategy allows the system designers to balance various design parameters (i.e. execution time, energy consumption, and reliability) according to the main constraints of their embedded system design.

Appendix A

In this part, it is explained how segmentation is implemented in mapping and scheduling parts. Figure 16.a and 16.b show how to recover the processor allocation (mapping part/scheduling part) from the task graph segmentation.

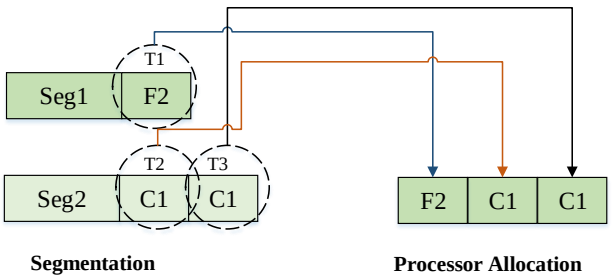


Figure16.a. Converting segmentation to Processor Allocation (the mapping part)

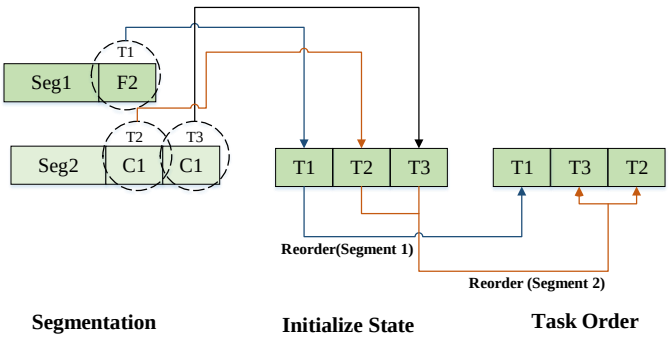


Figure16.b. Converting segmentation to Processor Allocation (the scheduling part)

Figure 16. (a) Converting segmentation to Processor Allocation (the mapping part, (b) Converting segmentation to Processor Allocation (the mapping part)

APPENDIX B

To prove that the genetic operators maintain the order of tasks and would not result in invalid orders, they have to fulfill the following condition [16]:

- **Correctness:** following the prioritized dependency of tasks.
- **Competence and Uniqueness:** non-repeated occurrence of all tasks in a given chromosome

Theorem 1: A task orders is an execution order if the tasks keep their dependencies orders. In this case if we remove T_i from the task orders, the remaining tasks still keep the topological order of tasks without violating precedence constrains (Correctness condition).

Proof 1: When T_i is removed from a topological order, the remaining tasks are actually a topological order of the new graph created by removing T_i from the original graph. Therefore, all priority constraints of the new graph are preserved in the remaining task orders.

Theorem 2: Task T_i can be inserted into any position among tasks with higher and lower priority then T_i which can provide a new task orders without violating priority constraints. (Competence and Uniqueness condition).

Proof 2: All tasks with the same higher and lower priority as T_i are independent of T_i. It means if we change the position of these tasks with each other there is no threat for priority constraints. Hence, the relative order between them in any task orders can be acceptable.

In the following, it is proved that the mutation operator maintains the order of tasks and will not result in invalid orders by fulfilling the necessary conditions [16]:

Theorem 3. For mutation, task orders can be replaced in any order without violating priority constraints (Correctness condition).

Proof 3. This is because of the fact that we only replace the tasks in the same segment and the tasks in a segment have the capacity of parallel execution.

APPENDIX C

In this appendix, we have performed a hyper-parameter exploration in the experiments. In the first set of these experiments, we have extended the population to 500 and 1000. This modification leads to acceleration of finding the solution with only 150 and 30 iterations respectively. However, memory usage increased to 50 and 60 percent respectively for experiments with 500 and 1000 populations. The obtained results are shown in Figures 17.a and 17.b for 500 and 1000 population sizes, respectively.

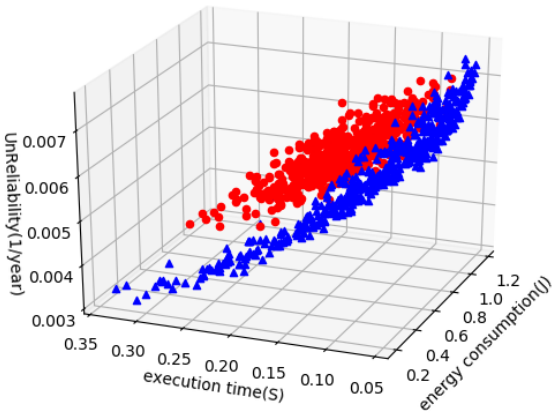


Figure17.a. The first (red circles) and final (blue triangles) results with population size of 500 after 150 iterations.

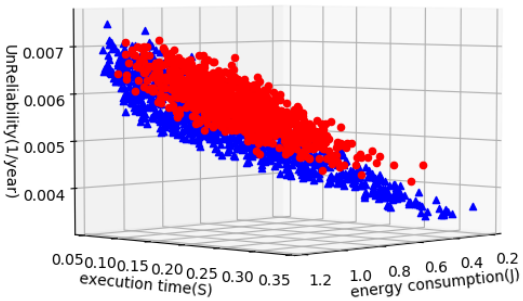


Figure17.b. The first (red circles) and final (blue triangles) results with population size of 1000 after 30 iterations.

In order to explore the impact of the mutation and crossover rates, we have made changes to the crossover and mutation probabilities in MOGATS. The obtained results are shown in Figure 18.a in which the mutation and crossover probabilities have been set to 0.6 and 0.03, respectively. The results show that, it takes about 1700 iterations to obtain Pareto set similar to the set in the experimental result section. The reason for that is that, the local exploration the design space is reduced by reducing the crossover probability. In another experiment, we have set crossover and mutation probabilities to 1 and 1 shown in Figure 18.b. In this case, all of the produced children are acceptable. In this experiment, the Pareto set similar to the one in the experimental result section have been obtained in 500 iterations while the memory usage has been increased about 40%.

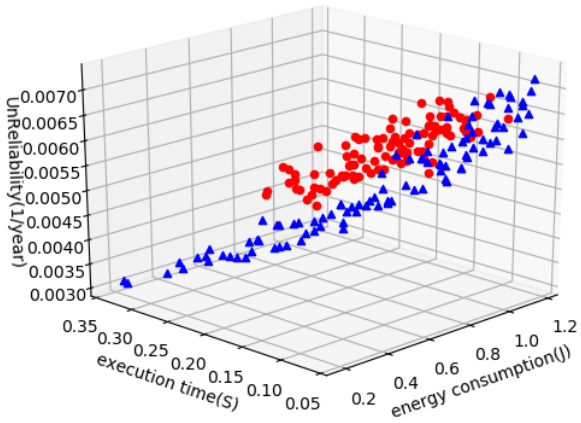


Figure18.a. The first (red circles) and final (blue triangles) results with crossover probability of 0.03 and mutation probability of 0.6.

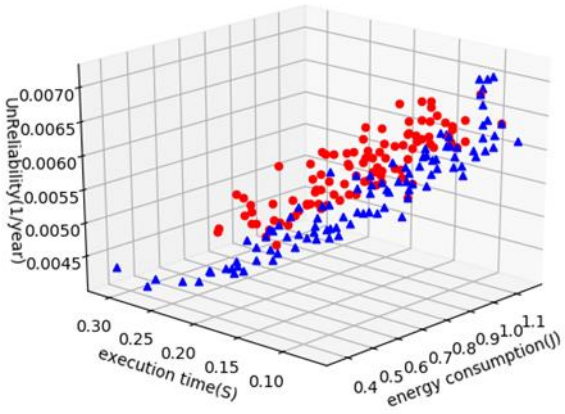


Figure18.b. The first (red circles) and final (blue triangles) results with crossover probability of 1 and mutation probability of 1.

REFERENCES

[1] Yuanyuan Fan; Qingzhong Liang; Yunsong Chen; Xuesong Yan," Executing time and cost-aware task scheduling in hybrid cloud using a modified DE algorithm," International Journal of Computeational Science and Engineering(IJCSE), 2019 Vol. 18 No. 3, pp: 217 – 226.

[2] Rachhpal Singh, "Hybrid genetic, variable neighbourhood search and particle swarm optimisation-based job scheduling for cloud computin," International Journal of Computeational Science and Engineering (IJCSE), 2018 Vol. 17 No. 2, pp: 184 – 191.

[3] YongXing Liu; Kenli Li; Zhuo Tang; Keqin Li, "Energy aware list-based scheduling for parallel applications in cloud," International Journal of Computeational Science and Engineering (IJCSE), 2018 Vol. 10 No. 5, pp: 345 – 355.

[4] Sambit Kumar Mishra; Md Akram Khan; Dampa Ashoo; Bibhudatta Sahoo," Allocation of energy-efficient task in cloud using DVFS," International Journal of Computeational Science and Engineering (IJCSE), 2019 Vol. 18 No. 2, pp: 154 – 163.

[5] Xiang Yu; Hui Wang; Hui Sun, " Decomposition-based multi-objective comprehensive learning pratocle swarm optimisation, " International Journal of Computeational Science and Engineering(IJCSE), 2019 Vol. 18 No. 4, pp: 349 – 360.

[6] Ullman, J. (1975). NP-complete scheduling problems. Journal of Computer and System Sciences, 10(3), pp.384-393.

[7] Salimi, Maghsood, et al. "Multi-objective Optimization of Real-Time Task Scheduling Problem for Distributed Environments." Proceedings of the 6th Conference on the Engineering of Computer Based Systems. ACM, 2019.

[8] Maid, Amin, et al. "NOMeS: Near-optimal metaheuristic scheduling for MPSoCs." 2017 19th International Symposium on Computer Architecture and Digital Systems (CADS). IEEE, 2017.

[9] S-H. Kang, H. Yang, L. Schor, I. Bacivarov, S. Ha, and L.Thiele, "Multi-objective mapping optimization via problem decomposition for many-core systems" *ESTlmedia*, 2012.

[10] Nikolov, H., Stefanov, T. and Deprettere, E. (2008). Systematic and Automated Multiprocessor System Design, Programming, and Implementation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(3), pp.542-555.

[11] Sigdel, K., Galuzzi, C., Bertels, K., Thompson, M. and Pimentel, A. (2012). Evaluation of Runtime Task Mapping Using the rSesame Framework. International Journal of Reconfigurable Computing, 2012, pp.1-17.

[12] Akbari, M., Rashidi, H. and Alizadeh, S. (2017). An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems. Engineering Applications of Artificial Intelligence, 61, pp.35-46.

[13] Pimentel, A., Erbas, C. and Polstra, S. (2006). A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Transactions on Computers, 55(2), pp.99-112.

[14] Ullman, J. (1975). NP-complete scheduling problems. Journal of Computer and System Sciences, 10(3), pp.384-393.

[15] Embedded System Synthesis Benchmark Suite (E3S), 2008, http://ziyang.eecs.umich.edu/dickrp/e3s/, (Accessed 23 August 2018).

[16] Xu, Y., Li, K., Hu, J. and Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Information Sciences, 270, pp.255-287.

[17] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiojective genetic algorithm: NSGA-II,"*IEEE Trans. On Evolutionary Computation, 2002.*

[18] A.Y. Yamamoto and C.Ababei, "Unified reliability estimation and management of NoC based chip multiprocessors". *Microprocessors and Microsystems, 2014.*

[19] Erbas, C. (2006). System-level modelling and design space exploration for multiprocessor embedded system-on-chip architectures. Amsterdam: Amsterdam University Press.

[20] I. Meedeniya, A. Aleti, and L. Grunske, "Architecture-driven reliability optimization with uncertain model parameters", *J.of Systems and Software, 2012*

[21] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in Proceedings of the Fifth International Conference on Genetic Algorithms, S.Forrest, Ed. San Mateo, CA: Morgan Kauffman, 1993, pp. 416–423

[22] K. Deb, Multiobjective Optimization Using Evolutionary Algorithms. Chichester, U.K.: Wiley, 2001

[23] Burkimsher, A., Bate, I., Indrusiak, L.S., 2013. A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times. Future Gener. Comput. Syst.29, 2009–2025.

[24] Dai, Y. and Zhang, X. (2014). A Synthesized Heuristic Task Scheduling Algorithm. The Scientific World Journal, 2014, pp.1-9.

[25] Ijaz S., Munir E., Anwar W., and Nasir W., "Efficient Scheduling Strategy for Task Graphs in Heterogeneous Computing Environment," The International Arab Journal of Information Technology, vol. 10, no. 5, pp. 486-492, 2013.

[26] Kumar, Vinay; Katti, C P, "A Scheduling Approach with Processor and Network Heterogeneity for Grid Environment,". International Journal on Computer Science and Engineering, Vol. 6(1), pp. 42-48, (Jan2014).

[27] Neubert, G., Savino, M.M., and Pedicini, C. (2010), "Simulation approach to optimize production costs through value stream mapping", International Journal of Operations and Quantitative Management, Vol. 16 (No. 1), pp. 1-21.

[28] Savino, M. and Mazza, A. (2015). Kanban-driven parts feeding within a semi-automated O-shaped assembly line: a case study in the automotive industry. Assembly Automation, 35(1), pp.3-15.

[29] Topcuoglu, H., Hariri, S. and Min-You Wu (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3), pp.260-274.

[30] Dick and Jha, "MOGAC: a multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems", Proceedings of IEEE International Conference on Computer Aided Design (ICCAD) ICCAD-97, 1997.

[31] Zhang, J., Wen Hu, W. and Ni Yang, M. (2014). A Heuristic Greedy Algorithm for Scheduling Out-Tree Task Graphs. TELKOMNIKA Indonesian Journal of Electrical Engineering, 12(6).

[32] Gupta, S., Kumar, V. and Agarwal, G. (2010). Task Scheduling in Multiprocessor System Using Genetic Algorithm. 2010 Second International Conference on Machine Learning and Computing

[33] García-Martínez, C., Lozano, M., Herrera, F., Molina, D. and Sánchez, A. (2008). Global and local real-coded genetic algorithms based on parent-centric crossover operators. European Journal of Operational Research, 185(3), pp.1088-1113.

[34] Sivanandam, S. and Deepa, S. (2008). Introduction to genetic algorithms. Berlin [etc.]: Springer.

[35] C. Erbas, S. Cerav-Erbas and A. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design", IEEE Transactions on Evolutionary Computation, vol. 10, no. 3, pp. 358-374, 2006.

[36] V. Kianzad and S. Bhattacharyya, "CHARMED: a multi-objective co-synthesis framework for multi-mode embedded systems", Proceedings. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004.

[37] N. Nedjah, M. da Silva and L. de Macedo Mourelle, "Customized computer-aided application mapping on NoC infrastructure using multi-objective optimization", Journal of Systems Architecture, vol. 57, no. 1, pp. 79-94, 2011.

View publication stats