The Journal of Supercomputing (2022) 78:279–314
https://doi.org/10.1007/s11227-021-03887-1

UMOTS: an uncertainty-aware multi-objective genetic algorithm-based static task scheduling for heterogeneous embedded systems

Mohsen Raji[1] · **Mohaddaseh Nikseresht**[1]

Accepted: 11 May 2021 / Published online: 25 May 2021

Abstract

Increasing manufacturing process variations due to aggressive technology scaling in addition to heterogeneity in design components are expected to cause serious challenges for future embedded system design steps including task scheduling. Process variation effects along with increased complexity in embedded applications result in design uncertainties, which in turn, reduce the accuracy and efficiency of traditional design approaches with deterministic values for the design component parameters. In this paper, a multi-objective task scheduling framework is proposed for embedded systems considering uncertainties in both hardware and software component parameters. The tasks which are modeled as a task graph are scheduled on a specific hardware platform consisting of processors and communication parts. Uncertainty is considered in both software (task parameters) and hardware (processor and communication parameters) of the embedded system. UMOTS takes advantages of a Monte-Carlo-based approach within a multi-objective genetic algorithm to handle the uncertainties in model parameters. The proposed approach finds the Pareto frontier, which is robust against uncertainties, in the objective space formed by performance, energy consumption, and reliability. The efficiency of UMOTS is investigated in the experimental results using real-application task graphs. In terms of Scheduling Length Ratio (SLR) and speedup, UMOTS provides 27.8% and 28.6% performance improvements in comparison to HSHD, one state-of-the-art task scheduling algorithm. Additionally, UMOTS, which is based on a multi-objective genetic optimization algorithms, finds robust Pareto frontier with 1%, 5% and 10% uncertainty in design indicators with respect to design limitations.

Keywords Embedded systems · Task scheduling · Multi-objective optimization · Genetic algorithm · Monte Carlo simulation · Uncertainty

✉ Mohsen Raji
mraji@shirazu.ac.ir

Extended author information available on the last page of the article

1 Introduction

An embedded system is a combination of computer hardware and software, designed for a dedicated function embedded within a larger system [1]. Embedded systems are widely used in commercial, industrial, and military applications; i.e., from small portable devices such as digital watches and cameras to large complex systems like autonomous vehicles, medical imaging systems, and avionics [2]. Embedded system design combines hardware (electronic, electrical, and electromechanical) and software (program) design. It composed of various steps including design specification extraction, modular design, application mapping, task scheduling, hardware/software implementation, and testing [1]. In modern embedded systems, designers take advantages of hardware platforms which are composed of different hardware processing elements (PEs) and communication links to provide better design solutions. However, the design process of such systems which are called *heterogeneous embedded systems* becomes more challenging [3].

Application mapping and task scheduling (briefly called task scheduling) is an important step in embedded system design [4]. The functionality of the systems is modeled as a set of tasks with a specific dependency shown in a graph called a task graph. Task scheduling is defined as the process of assigning the tasks on the heterogeneous hardware platforms (i.e., mapping) in addition to finding the order of executing them (i.e., scheduling). Since task scheduling problem is an NP-hard one [5], meta-heuristic algorithms are suitable to be used for finding the solutions which are as close as possible to the global optimum solution.

During task scheduling, the designer has to consider various design requirements to be satisfied including execution time, energy consumption, and reliability. These design parameters are conflicting [6]; i.e., improving each of them may degrade the other ones. For example, improving the reliability requires redundancy (either spatial or temporal), which adversely affects the execution time. Similarly, the execution time and the power consumption are also conflicting because downscaling the operating voltage and frequency of the PEs for reducing the power consumption increases the execution time. This property motivates the designers to apply multi-objective algorithms for task scheduling problem.

Recently, different sources of uncertainty have emerged and affected the task scheduling step of embedded systems. For example, manufacturing process variations originated from inexact hardware fabrication [7] have led to uncertainties in the characteristics of hardware platforms such as the delay and power consumption. On the other hand, the software components have not been developed completely in the task scheduling step and thus, there is some sort of uncertainties in terms of their execution time and energy consumption [7]. These features result in more uncertainties in design parameters; i.e., the design parameters would not be precisely and deterministically known anymore. Hence, the traditional approaches of design space explorations for embedded systems which are based on deterministic values for design parameters may lose their accuracy and efficacy in finding the true optimal solutions. So, it is required to develop multi-objective task scheduling algorithms capable of finding the design solutions which are robust against uncertainties with certain confidence [8].

This paper presents UMOTS which is an Uncertainty-aware Multi-Objective Task Scheduling algorithm for embedded system design. The tasks are modeled as a task graph and scheduled on a specific hardware platform consisting of processors and communication parts. In UMOTS, the task scheduling is modeled as a multi-objective genetic algorithm-based optimization problem. It takes advantages of a MC-based approach within the genetic algorithm to handle the uncertainties in both hardware and software components. The proposed approach finds the robust Pareto frontier against uncertainties in the objective space formed by performance, energy consumption, and reliability. The efficiency of UMOTS is investigated in the experimental results using real-application task graphs. In terms of Scheduling Length Ratio (SLR) and speedup, UMOTS provides 27.8% and 28.6% improvements in comparison to Heuristic Approach for Scheduling in Heterogeneous Distributed Embedded Systems (HSHD), one state-of-the-art task scheduling algorithm [3]. Additionally, we are able to find robust Pareto frontier with 1%, 5% and 10% uncertainty in design indicators with respect to design limitations.

The rest of the paper is organized as follows: In Section II 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 III, experimental results are presented and, finally, the paper is concluded in Section IV.

2 Related works

Design space exploration for embedded systems (specially the application mapping and task scheduling step) has been extensively investigated in the past. Some previous works proposed solutions for only the mapping problem in heterogeneous embedded systems [9–11]. In [12–14], task mapping has been formulated as multi-objective optimization in the system level abstraction. Some computer-aided design automation tools have been proposed for a multi-objective design optimization for embedded systems [15–18]. Multi-objective task scheduling has been also studied in some previous works [19] and [20]. However, these works have serious limitations; i.e., some of them only deal with the problem of scheduling tasks and did not consider the uncertainty in the important design parameters of embedded systems including the execution time, energy consumption, and reliability.

Some recent works have attempted to consider uncertainties in design space exploration of embedded systems. In [21], a task allocation and scheduling algorithm for Multiprocessor System-on-Chip (MPSoC) architectures has been presented considering the impact of manufacturing process variations in the hardware components. However, this algorithm is not generally applicable to all sources of uncertainty and other essential uncertainties such as voltage and temperature variations and the uncertainty in software components have not been considered. An uncertainty-aware optimization approach for the design phase of software-intensive systems considering the architecture-based reliability has been presented in [22].

The impact of design-time uncertainty evaluations for parameters (the failure rates of software and hardware components) was taken into consideration in an architecture optimization process. The uncertainties were handled by Monte-Carlo (MC) simulations combined with probabilistic-based evaluation methods and hence, this approach supported any type of probability distribution in the input parameters. A genetic algorithm-based metaheuristic approach is used for solution exploration of task mapping problem. The two disadvantages of this method are: (1) it only deals with the uncertainty in reliability parameter while other design parameters may also be affected by uncertainty and (2) this work focused only on the task mapping problem and neglected the problem of task scheduling. In [23], an uncertainty-aware mapping approach is presented for embedded systems. The problem of uncertainty-aware mapping for multicore embedded systems has been formulated and solved by integrating the uncertainty models with MC and genetic algorithm. This design methodology is developed as a tool to find the Pareto frontier which is robust against uncertainties in both hardware and software components. The objective space of the optimization is execution time, energy consumption, and reliability. The main limitation of this work is that its focus is only on mapping problem which is much simpler than the scheduling one; i.e., it only assigns the tasks to the PEs but does not determine the execution order of the tasks which may be assigned to the same PE. Also, this work uses a simple evaluation approach for the reliability parameter which does not cover all scenarios of hardware and hardware-originated software failures.

3 Preliminaries

A System model: software and hardware components.

For modeling the software component of the embedded system, the model introduced in [27] is used. This model is based on the Kahn Process Network (KPN), a very well-known computational model for modeling embedded systems [27]. KPN is a method for representing tasks in which tasks are indicated in the form of a directed graph. The KPN is represented by a directed graph $G_{SW}\left(V_{SW}, E_{SW}\right)$ in which a vertex $v_i \in V_{SW}$ for $i = 1, \ldots, \left|V_{SW}\right|$ 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 $E_i = \{e_j E_{SW}\}$. In a case where v_i is assigned to a hardware component in a scheduling, h_i represents the execution time on this hardware component. If it is possible for a task to be run on multiple hardware processing cores (e. FPGA or ASIC), its execution time is shown as the set of $h_i = \left\{h_{i1}, h_{i2}, \ldots, h_{iH}\right\}$.where H 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 PE (i.e., CPU or DSP) for scheduling, s_i represents the execution time on that software PE. When it is possible to run a task on multiple software elements, the execution time is shown as the set of $s_i = \left\{s_{i1}, s_{i2}, \ldots, s_{iS}\right\}$ where S represents the number of software PE. Each edge $e_j \in E_{SW}$ for $j = 1, \ldots, \left|E_{SW}\right|$ represents the relationship between two tasks of the task graph G_{SW}. If this link is allocated to a memory, m_j represents the memory access time. If ring scheduling, the edge can be allocated to multiple memories, the corresponding access time is represented as to the set of $m_j = \{m_{j1}, m_{j2,}, \dots, m_{jW}\}$ where W denotes the number of memories that the task can be assigned.

The component is modeled as a graph $G_{HW}(V_{HW}, E_{HW})$ in which the V_{HW} and E_{HW}, 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., $V_{HW} = PUM$. A delay between communication links is indicated by l_{pq}, where $p, q \in \{1 \dots |E_{HW}|\}$. The energy consumption related to running a 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. It is notable that, since the main goal of UMOTS is to solve the problem of task scheduling in embedded systems not hardware/software partitioning for finding the best hardware architecture, it is assumed that the hardware platform is known and specified as the input of the framework.

B Uncertainty modeling.

The uncertainty block labeled uncertainty in Fig. 1 shows the injection process of uncertainties into the component parameter. There are numerous previous studies on the uncertainty handling in different scientific fields [24–26]. Nevertheless, it is generally accepted that there is no single modeling approach for addressing the uncertainties. Hence, UMOTS takes advantages of MC simulation-based approach which is accounted as the most general approach for uncertainty handling; i.e., generalized

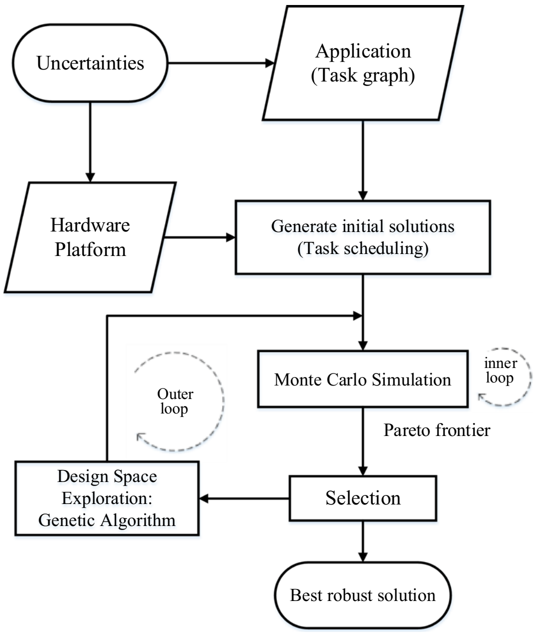


Fig. 1 Block diagram of UMOTS

(continuous and discrete) probability distributions can be considered for modeling the uncertainty-aware design parameters in UMOTS.

UMOTS supports the designer to inject uncertainty into the software component (task graph-related features) or/and the hardware architecture (processors, communication links). Moreover, UMOTS is able to inject different uncertainty degrees or amounts during the inner loop of the design space exploration. The uncertainty injection process is done during the MC simulation by generating random samples based on the pre-specified probability distributions. The uncertainty degree is considered in the sampling step as proposed in [23] based on the probability distribution parameters (i.e., mean and standard deviation) and the rules listed in Table 1.

The reasoning behind the rules can be justified considering the mathematical concept of mean and variance in the probability distribution functions. For a uniform distribution $U(a, b)$, in terms of mean μ and variance σ^2, the probability density $(f(x))$ may be written as:

$$f(x) = \begin{cases} \frac{1}{2\sigma\sqrt{3}} & -\sigma\sqrt{3} < x - \mu < \sigma\sqrt{3} \\ 0 & \text{otherwise} \end{cases}$$

Hence, for example for 1% uncertainty, the designer expect that, the amount of uncertainty with respect to the mean (i.e., 0.01μ) falls into the defined bound of the distribution (i.e., $\sigma\sqrt{3}$). So, we have:

$$0.01\mu = \sigma\sqrt{3} \rightarrow \sigma = 0.01\mu/\sqrt{3}$$

In the case of a Gaussian distribution $Gaussian(\mu, \sigma)$, randomly generated samples which fall inside the interval [μ-3σ, μ+3σ] are accepted, which represents 99.7% of all generated samples. Hence, for example for 1% uncertainty, the uncertainty amount (i.e., 0.01μ) should fall into the defined distribution bound (i.e., 3σ) So, we:

$$0.01\mu = 3\sigma \rightarrow \sigma = 0.01\mu/3$$

.

Similar reasoning is applicable for other distributions that are interested by the designer in modeling parameter uncertainties. In UMOTS, we use uniform and Gaussian distributions for modeling the execution time, energy consumption, and

Table 1 Rules for defining mean and variance of distributions from which sampling must be done to achieve a certain degree of uncertainty injection [23]

Probability distribution	Uncertainty level		
	1%	5%	10%
Uniform(μ, σ)	$\sigma = 0.01 \cdot \mu/\sqrt{3}$	$\sigma = 0.05 \cdot \mu/\sqrt{3}$	$\sigma = 0.01 \cdot \mu/\sqrt{3}$
Normal(μ, σ)	$\sigma = 0.01 \cdot \mu/\sqrt{3}$	$\sigma = 0.05 \cdot \mu/\sqrt{3}$	$\sigma = 0.01 \cdot \mu/\sqrt{3}$

temperature, and reliability of the embedded system. It is notable that, UMOTS is flexible and can be easily adopted to consider other probability distributions if the designers of embedded system find those distributions to fit better to their field data.

4 UMOTS: the uncertainty-aware multi-objective genetic algorithm-based task scheduling method

This section presents the proposed Uncertainty-aware Multi-Objective Genetic Algorithm-based Tasks Scheduling Approach called UMOTS. At first, the overall block diagram of UMOTS is presented. Then, the uncertainty modeling method is described. After that, we explain how the software and hardware components of the embedded system are modeled. Design space exploration methodology is provided in the next sub-section.

A Block diagram of UMOTS.

The block diagram of UMOTS which is adopted from [23] is shown in Fig. 1. The inputs are software and hardware components of the embedded systems. UMOTS finds the best task scheduling by exploring the design space formed by all possible solutions. This exploration is done by a two-nested-loop iterative process. The outer loop is related to the genetic optimization algorithm in which in each iteration, a population of candidate solutions (task scheduling results) is evolved toward better solutions (i.e., with better values of the objective functions). The inner loop is related to the MC simulation method applied for the uncertainty-aware estimation of the objective functions. The objectives considered in UMOTS are performance (measured as execution time), energy consumption, and reliability. In each iteration of MC simulation, the uncertainties are injected into the parameters of hardware and software components based on an uncertainty model. As the final output, UMOTS provides the designer with a set of solutions which are the Pareto frontier in the four dimensional objective space and robust against uncertainties. The details of UMOTS are provided in the next sections.

B Multi-objective genetic algorithm-based design space exploration.

Design space exploration here refers to finding the best solutions by exploring among all possible solutions to the task scheduling problem. Figure 1 shows an illustrative example of a solution found for a task scheduling problem given the software component (task graph) and hardware component of an embedded system. Since the task scheduling problem is an NP-hard one [1], it is required to use meta-heuristic algorithms to find the 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. In this subsection, we describe the design space exploration approach which is based on a multi-objective genetic optimization algorithm.

In the following, before going deep into modeling the problem with genetic algorithm, we describe a segmentation method presented in [28] to handle the dependency between the tasks in task scheduling problem. After that, the chromosome representation and genetic operators (including crossover, mutation, and selection) are presented. Finally, the objective function (i.e., execution time, energy consumption, temperature, and reliability) are modeled in terms of the parameters of hardware and software components.

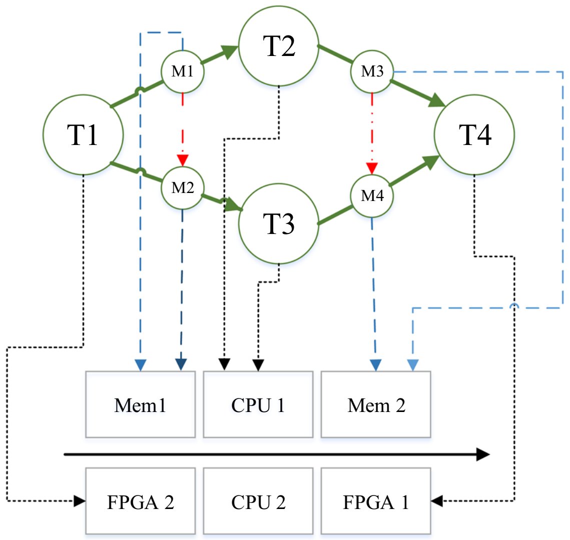
(1) Segmentation method.

The segmentation method described in [28] to eliminate the inter-task dependencies in the task graph. According to the dependency graph, separate segments are extracted such that each segment includes independent tasks that can be simultaneously executed on different processors. Each element of each segment shows the composition of the task and the processor assigned to it. As mentioned, the feature of the segmentation method is that the tasks within each segment are independent of each other, but the segments themselves must be executed sequentially. Figure 2a shows the mapping of task graphs T1 to T4 and communication between tasks M1 to M4 to the architecture platform. Figure 2b illustrates a scheduling for this mapping. Figure 2c shows the result of the segmentation procedure for this task graph in which the tasks are placed in three segments; i.e., segment 1 (Seg1) which consists of task T1, segment 2 (Seg2) which consists of task T2 and task T3 as they are independent and can be executed in parallel, and task T4 is placed in the third segment (Seg3). Figure 2d indicates the segmentation which a hardware is assigned to each task. For example, in segment 1 (Seg1), the task T1 runs on the F2 processor. After running the Seg1 task list, it is done. Seg2 and Seg3 can execute their to-do list in sequential order. Note that for a dependency graph, the order of the inter-segmentation tasks is assumed to be the same for all chromosomes, but the assigned processors (entries of each segment) are different. Each core is the output of the randomly select a processor from a set of available processors.

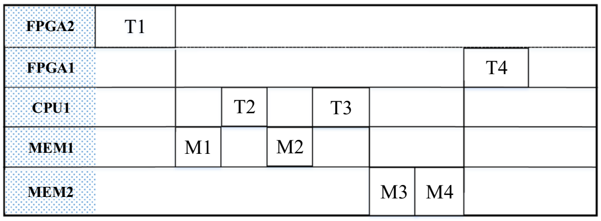
(2) Chromosome representation.

In a genetic algorithm which is trying to solve a task scheduling problem, each solution is represented by a chromosome which includes the assigned tasks/dependency and processors/communication links. For a task graph with a set of N tasks and M communication nodes, each chromosome is characterized by a set of $2\,\mathrm{M}+\mathrm{N}$ genes.

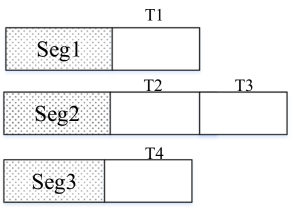
The structure of a chromosome in UMOTS is shown in Fig. 3a. Each chromosome consists of two main parts:



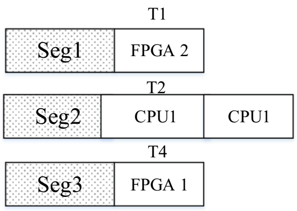
(a) An illustrative task graph and its mapping on a given architecture



(b) An example of task scheduling

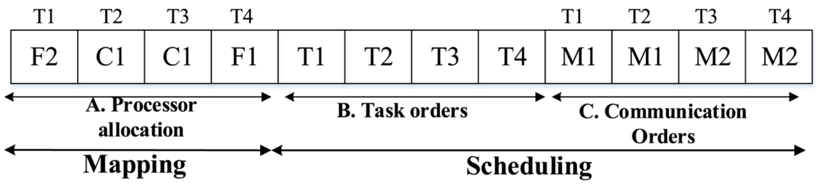


(c) The result of task segmentation procedure.

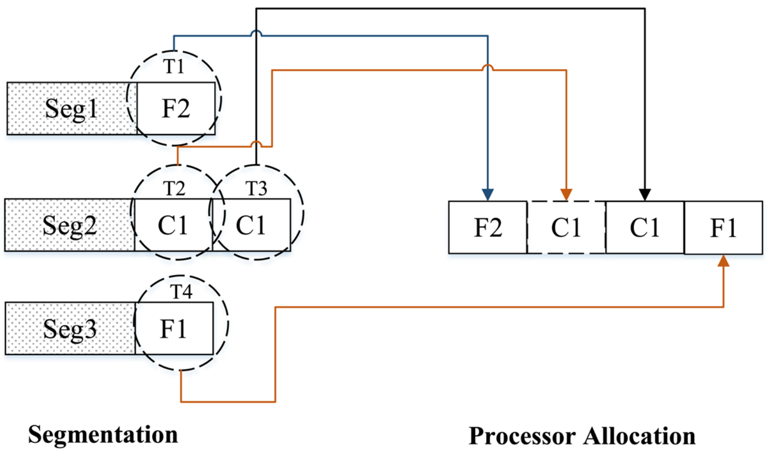


(d) Task segmentation used in the task scheduling.

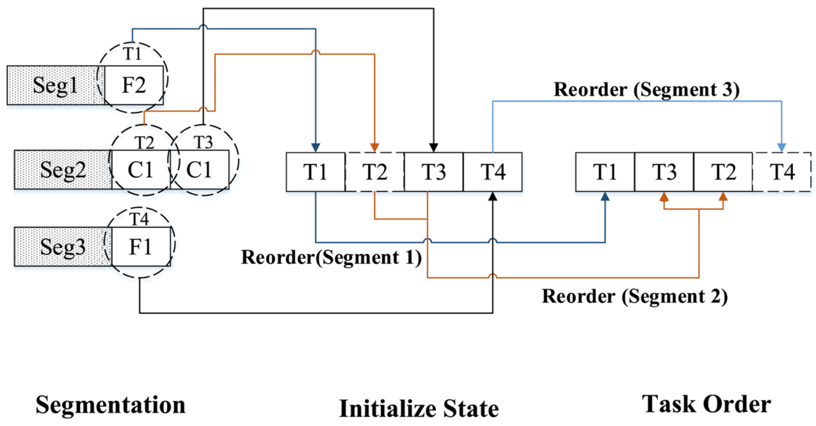
Fig. 2 The problem of mapping and scheduling in the design of Embedded Systems and an example of segmentation procedure results



(a) Chromosome representation for task graph shown in Figure 2(a)



(b) Converting segmentation to Processor Allocation (the mapping part)



(c) Converting segmentation to Processor Allocation (the scheduling part)

Fig. 3 Chromosome representation of UMOTS

(1) *Mapping part* In this part, the PE which is selected for each task is specified. In a chromosome, the first N genes are related to the mapping of all N tasks on P processors. For example, task T2 is mapped to processor C1 (Fig. 3a). In order to assign tasks to processors in constructing the initial population, a processor is randomly selected from the set of available processors. Figure 3b shows how the mapping part is constructed according to the task graph segmentation.

(2) *Scheduling part* In this part, the order of executing tasks on PEs and communications are specified. The first portion of this part (N genes) shows the order of executing tasks on the selected processors and the second portion (M genes) represents the order of data transmission over the communication links. In the first part, we assign the tasks according to the order assigned to the partitioning method (initial state) and then, randomly change the ordering of the same tasks by calling the reorder function to make sure all possible execution orders have been supported. Illustrates the process of this conversion in Fig. 3, Part c. The second part deals with data transfer using communication links, respectively. The process is similar to the procedure performed in the mapping section in which a memory is randomly selected from the set of available memory elements and assigned to the related task.

(3) Genetic operators.

Genetic algorithms use genetic operators to generate and maintain chromosome diversity from one generation to another. UMOTS takes advantages of the most common genetic operators; i.e., crossover and mutation. In the following, the details of these genetic operators are explained more.

(a) Crossover.

The crossover operator is used to combine the information of two individuals (called parents) and produce a new one (called child). In UMOTS, the local parent selection technique has been used in which chromosomes are classified based on the NSGA II Pareto set classification [29]. In this approach, the individuals are classified based on their appropriateness (i.e., fitness values) and then, the first parent is randomly chosen while the other one is randomly selected from the same class of the first one. For example, if the first parent be selected from the Pareto set class 1, the second Pareto will also be selected from the Pareto set class 1. After the parents' selection step, two children are generated as follows. This process is shown in Fig. 4a.

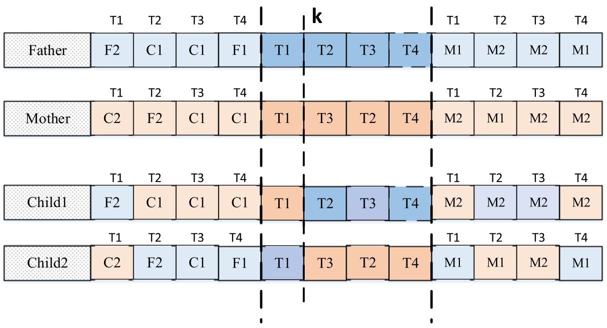
The pseudocode of the crossover for the mapping part is shown in Fig. 4b. In order to find the mapping part of the children chromosomes, the uniform crossover is used. According to this operator, for each chromosome gene, a number is generated between 0 and 1. If this number be less than P_C, the mother's gene will be selected to fill the mapping part of the child #1's chromosome and otherwise, the father's gene will be selected. For the second child, this process is reversed. In the scheduling part, the single point crossover algorithm is used. In this approach, a boundary point called k is used. This point is between 0 and $N-1$ where N

```
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 children
Repeat
    1. Generate a random number R∈ $[0 - 1]$
    2. If (R $\leq P_C$)
           3. Copy mother's gene to child #1's gene and
           Copy father's gene to child #2's gene
    4. Else if(R $> P_C$)
           5. Copy father's gene to child's gene and Copy
           mother's gene to child2's gene
Until Size of Tasks
```

(a) Crossover- mapping part

```
Inputs:
        1. The same parents from crossover-mapping part
Output: Generating two new children
Repeat
        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 child #1 and tasks
           from father [0 to $k$] are directly transmitted to
           child #2
        4. To fill [$k$+1 to $N$-1], father's chromosomes are
           scanned from beginning and each node that is
           not yet in the child is added to the next empty
           position of the child's chromosome. The same
           repeated with mother's chromosome for child #2
Until Size of Tasks
```

(b) Crossover- scheduling part



k
Father: F2 C1 C1 F1 | T1 T2 T3 T4 | M1 M2 M2 M1
Mother: C2 F2 C1 C1 | T1 T3 T2 T4 | M2 M1 M2 M2
Child1: F2 C1 C1 C1 | T1 T2 T3 T4 | M2 M2 M2 M2
Child2: C2 F2 C1 F1 | T1 T3 T2 T4 | M1 M1 M2 M1
T1 T2 T3 T4

(c) An illustrative example of Crossover operation

Fig. 4 Crossover operator

represents the total number of tasks. The parents' task orders are cut from point k and the first part of the mother's task orders is copied to the same part of the child #1's chromosome. The same will be repeated for child #2 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 the father's task order part which did not appear on the left side of child #1's task orders until filling it. The same will be done for the child #2 with the mother's task orders. The crossover operation applied for the communication links portion of the scheduling part is similar to the one explained for the mapping part. Figure 4c illustrates an example of applying the scheduling operator on two chromosomes.

The proof of the validity of these crossover approaches is provided in the "Appendix".

(b) Mutation.

A mutation operator or mutant is a genetic operator used to maintain genetic diversity from one generation of chromosomes to the next. These changes should not affect the priority of tasks. In UMOTS, one parent is randomly selected from the current population to implement this operator, and one child is produced from the selected parent. Since, the changes made by this operator should not violate the task dependency graph, changes in chromosomes will occur through segmentation.

The pseudocode of mutation for the mapping part of the child's chromosome is provided in Fig. 5a. In the mapping part of the child's chromosome, uniform mutation is applied in which the parent's gene will be copied to the child's one with probability P_m; i.e., a random number between 0 and 1 is generated and if the number be lower than P_m, the parent's gene will be copied to the child's one and otherwise, the gene will be changed such that another PE from the PE pool is chosen randomly.

Figure 5b shows the pseudocode of mutation for the scheduling part of the child's chromosome. For scheduling part, a task is randomly selected in each segment and is replaced with another task from the same segment; i.e., their execution order will be changed.

The mutation operation procedure used for the communication links in the scheduling part is similar to the mutation operator of the mapping part.

Figure 5c shows an example of applying the mutation operator on a chromosome; i.e., a random number in task number's range is generated and we find Task T segment and randomly replace the task T with a task in that specific segment.

The proof of the validity of these crossover approaches is provided in the "Appendix".

(4) Selection.

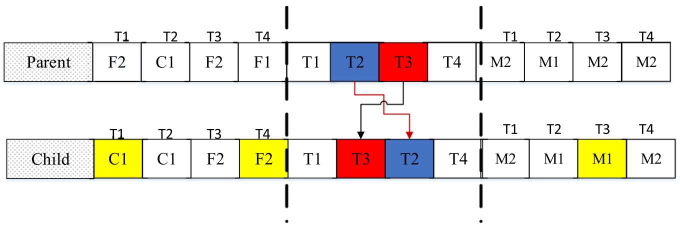
The selection operator that is used for choosing the desirable population for the next generation is based on NSGAII [29]. The pseudocode of NSGAII is shown in Fig. 6. Briefly, the first population is generated randomly and then, a procedure is repeated during several generations; i.e., the objective functions are evaluated

```
Inputs:
    1.  One parent is randomly chosen from current
        population
Output: Generating two new children (Part A)
Repeat
    2.  Generate a random number R∈ $[0 - 1]$
    3.  If(R ≤ $P_m$)
            4. Copy parent's gene to the child's one
    5. Else if(R > $P_m$)
            6. One PE is randomly chosen from the pool of
            the PE
            7. Copy the parent's gene to the child's one
Until Size of tasks
```

(a) Mutation operation on mapping part

```
Inputs:
    1.  The same parent from mutation-mapping part
Output: Generating two new children (Part B)
Repeat
    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
            Until Size of tasks
```

(b) Mutation operation on the scheduling part



(c) An illustrative example of Mutation operation

Fig. 5 Mutation operator

for each population member, a ranking is done on the members for each objective function, the non-dominated members for each objective is selected to construct the next generation. The pseudocode of this process is shown in Fig. 6 and complete details are found in [29].

(5) Objective functions.

Fig. 6 Design Space Exploration based on NSGAII [29]

```
**Input:** N population size, M max number of generation
**Output:** Pareto frontier, non-dominated solutions in $P_M$
$P_0$ = GenerateInitialPopulation(); // Size N
$Q_0$ = ∅; // 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**
```

UMOTS is a multi-objective function in which three objectives are considered during the optimization process; i.e., execution time, energy consumption, and reliability. In the following, the functions which are used for evaluating the solutions in terms of the three objectives are provided.

(a) Execution time.

In order to evaluate the execution time of the solutions (i.e., task scheduling), the information contained within the chromosomes is used to extract the characteristics of the processor and the task in question. Then, the execution time of each task is computed as:

$$\mathrm{Exe}_{\mathcal{T}_n} = \frac{\mathrm{WCET}\left(\mathcal{T}_n, P_{in}\right)}{\frac{F_{jn}}{f\,\mathrm{max}_{p_{in}}}} \tag{1}$$

where $\mathrm{Exe}_{\mathcal{T}_n}$ represents the execution time of task $\mathcal{T}_n$ in the task graph and $\mathrm{WCET}\left(\mathcal{T}_n, P_{in}\right)$ shows the worst case execution time of task $\mathcal{T}_n$ on processor P_{in}. F_{jn} and $f\,\mathrm{max}_{p_{in}}$ are the operating frequency and maximum frequency of the processor P_{in}, respectively.

(b) Energy consumption.

The energy consumption of a system is obtained by multiplying the power consumption to the operating time of the system [30]. The power consumption includes a static and a dynamic part [31]. Dynamic power depends on the frequency and operating voltage while the static power is mainly due to leakage and depends on the current and temperature of the system [32].The overall system-level power consumption (P_{system}) is computed as:

$$P_{\text{system}} = P_{\text{dynamic}} + P_{\text{static}} = C_{\text{eff}} \times V^2 \times f + \alpha \times T(t) + \beta \tag{2}$$

where P_{dynamic} and P_{static}, respectively, shows the dynamic and static power consumption of the system C_{eff} represents the capacitance of the switching, V and f represent the voltage and operating frequency of the system. $T(t)$ shows the total execution time for the task to be completed. The coefficients α and β are system-specific and architecture-dependent constants. The value of the constant value parameters of the above failure mechanism are based on [27, 28].

(iii) Reliability.

In UMOTS, it is assumed that embedded systems may fail due the permanent failure mechanisms. The main mechanisms which leads to permanent faults are as follows: Electromigration (EM), Stress Migration (SM), Time-dependent Dielectric failure (TDDB), Navigation Bias Temperature Instability (NBTI) and Thermal Cycling (TC) [25, 33, 34]. In order to compute the system reliability, Mean Time to Failure (MTTF) of the system considering these failure are obtained as described in details in [34].

To calculate MTTF for the whole processor, one must consider the impacts of all failure mechanisms over time. So, we use the approach of Summing Of the Failure Rates (SOFR model) [33] which is widely used in the industry. In order to apply this model, we need to consider two assumptions: (1) a failure in one component causes the whole system to fail, (2) the calculated failure rates are constant over time. Given these assumptions, we have:

$$\text{MTTF}_P = \frac{1}{\lambda_P} \tag{3}$$

where MTTF_P and λ_P, respectively, represents the total MTTF and failure rate of PE P. Given the failure rate of each PE, the system reliability for the time interval of [0,t] ($R_{\text{system}}(t)$) is calculated as:

$$R_{\text{system}}(t) = \prod_{i=0}^{n} R_i(t) \Rightarrow R_i(t) = e^{-\lambda_{P,i} t_i} \tag{4}$$

where n represents the number of tasks executed by the PE, $R_i(t)$ shows the reliability of PE i in time interval [0,t], t_i denotes the execution time of the task on the scheduled PE, and $\lambda_{p,i}$ denotes the error rate on the PE P when executing task i.

According to the equations mentioned for system reliability, the temperature is one of the most effective parameters in computing the lifetime reliability of the embedded system. System-level temperature modeling of a computing system such as a multiprocessor embedded system is generally performed according to the Fourier Heating Law using the thermal model equation [24, 27]. The thermal degree of each PE is mainly defined by the energy consumed, the instantaneous temperature, and the ambient temperature modeled as:

$$C\left(\frac{\mathrm{d}T(t)}{\mathrm{d}t}\right) + G\left(T(t) - T_{\mathrm{amb}}\right) = P(t) \tag{5}$$

where C, G, $T(t)$, and T_{amb} are, respectively, the capacitors, thermal conductivity and ambient temperature and P represents the power consumption of the system based on its dynamic and static power components as shown in Eq. (2). To improve the accuracy of the high-level relationship at the system level, there must be an interaction and spatial dependence between the internal heat and the PEs. Hence, the following two-dimensional heat transfer equation is considered for the heat transfer between the PEs and is added to the right-hand part of the equation [28, 31]

$$\text{Heat\_transfer}_{2D} = \sum_{c' \in \mathrm{nbr}(c)} G\left(c, c'\right)\left(T_c(t) - T_{c'}(t)\right) \tag{6}$$

where nbr(c) are all neighbors of the PE c adjacent to it and its temperature dependence, T_c and $T_{c'}$ are the process temperatures of c and c' $G(c, c')$ is the thermal conductivity between the processors that depends on the distance and the geometric properties of the chip. By solving the differential Eq. 4 and considering the effect of adjacent processors according to Eq. 5, the operating processor temperature of the system is estimated according to the following equation.

$$T_C(t) = \frac{b}{a} + \left(T\left(t_0\right) - \frac{b}{a}\right)e^{-a\left(t-t_0\right)} \tag{7}$$

While the values of a and b are calculated from the following equations

$$\begin{aligned} a &= \frac{G - \alpha + \sum_{c' \in \mathrm{nbr}(c)} G\left(c, c'\right)}{C} \\ b &= \frac{G \cdot T_{amb} + \sum_{c' \in \mathrm{nbr}(c)} T_{c'}(t) \cdot G\left(c, c'\right) + C_{\mathrm{eff}} \cdot V^2 \cdot f + \beta}{C} \end{aligned} \tag{8}$$

(6) Solving the multi-objective optimization problem.

After defining all three objective functions as described in the previous section, the optimization problem (i.e., finding the best task scheduling) can be written in a general form as [35]:

$$\begin{aligned} &\min_x z = f(x) = \left(f_1(x), f_2(x), f_3(x)\right)^T \\ &s.t.\ x \in X \end{aligned} \tag{9}$$

where x shows a definite solution and X represents the set of acceptable solutions. The three main functions shown above $f_1(x)$, $f_2(x)$ and $f_3(x)$ efficiently evaluate the timing and mapping problem of Eqs. 1, 2, and 4. The target function $z = f(x)$

converts a x response of the decision space to a point of the target space using the design definitions of the simplified definition. As stated earlier because multiple target spaces do not usually have a single optimal response, we are in fact looking for a set of dominant optimal responses. These optimally dominant responses are not defeated by other possible solutions in the search space. In the following, we examine how to incorporate uncertainty into its design and modeling indices.

III Monte Carlo simulation for uncertainty-aware estimation.

During the implementation of the NSGAII algorithm [29] under uncertainty, three objective functions (i.e., execution time, energy consumption, and system reliability) for each of the selected solutions must be evaluated. In the traditional optimization approach, the values of the objectives would be evaluated to find the deterministic values. However, in an uncertainty-aware approach, none of the objectives can be measured based on deterministic calculations. Analytical evaluation methods are also very costly and complex when various probability distributions are applied in the optimization problem. Hence, estimation techniques capable of modeling and simulating the uncertainty should be implemented. In such cases, a MC-based simulator is considered as the only promising and practical solution for supporting various probability distribution types [22]. Hence, UMOTS takes advantages of this approach to evaluate the objective function under uncertainty shows as the inner loop in Fig. 1 by the Monte-Carlo Block. More details of this block are provided in Fig. 7.

For example, in order to simulate the reliability design parameter using the Sartre MC simulation, the reliability and uncertainty-influenced variables are modeled as probability distributions. During the MC iterations, the probability distributions, which are used for modeling the uncertainties, are randomly sampled to produce the

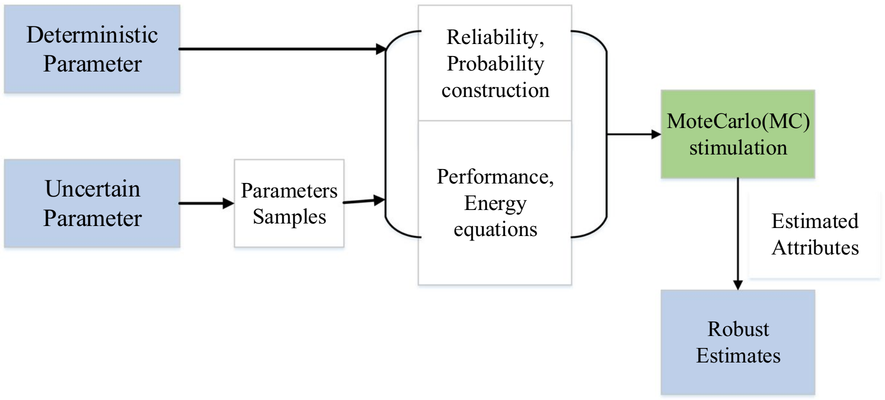


Fig. 7 The MC simulation approach for uncertainty-aware evaluation of execution time, energy consumption, and system reliability

instances that will be later employed as the numerical values to evaluate the objective functions using derived Eqs. 1, 2, and 3.

IV Robustness of the design solution points.

In the robust optimization approach with NSGA-II, we exploit the characteristic of evolutionary optimization as an accuracy improvement technique of the Monte-Carlo simulation. The output of Monte Carlo simulation is maintained in a data structure attached to the solution points. When an elite architecture candidate is re-evaluated in the next generation, the estimate is the union of new and previous results. As a result the accuracy of the estimate is improved. We use 95-percentile point estimate as the actual value used to generate and plot the robust Pareto frontier in the objective space. Working with percentage estimation as a tool has made it possible to identify and quantify robust solutions. As the indicator percentile grows the solutions are more robust against uncertainty. As a result after computing the values of the objective functions, we use a 95-percentile point estimate as the right value to produce and design the Pareto Frontiers in the target space.

To estimate the performance and energy consumption attributes, the MC simulation technique is similar. During multiple MC runs, parameters affected by uncertainties are also sampled from their respective probability distributions and used as numerical values inside Eqs. 3 to 8.

5 Experimental results

In this section, we conduct a set of experiment to show the efficacy of UMOTS.

A Experimental setup.

The proposed uncertainty-aware task scheduling method (UMOTS) is implemented in C+ +programming language and run on a LINUX machine with Intel processor Corei3 with 2.30 GHz clock frequency and 2 GB RAM.

For simulation, we have used 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)) and the other two test-cases are chosen from Embedded System Synthesis Benchmarks Suite (E3S) benchmarks [36].

In UMOTS, it is assumed that the hardware architecture and software components (tasks graph) are available as the inputs. Our experimental hardware platform contains six PEs with different levels of operational voltages and frequencies. Four PEs are considered between 300 and 900 MHz (similar to arm cortex design characterization) and two IBM PowerPC with 266 MHz clock frequency [37]. The communication arcs in the task graph are handled through communication units, which are assumed to be handled through memory mapping; i.e., the source PE writes into a memory component and then, the destination PE reads it from memory. Two communication units are considered in the following experiments.

The parameters of NSGA II are set as follows: the probability of crossover and mutation is set to 0.7 and 0.05 respectfully. The number of iterations is 9000 times with a population size of 100 [29].

To simulate the multi-objective problem and find the robust Pareto frontier, the objective functions of execution time, energy consumption, and reliability have been taken into consideration. We use UMOTS to identify robust Pareto frontier in the execution time vs. energy consumption versus (1-reliability) objective space. All attributes are assumed to be affected by uncertainty and thus, they are estimated using the MC simulation approach with 1000 iterations. Robust Pareto frontier has been obtained for different levels of uncertainty injections (i.e., 1%, 5%, and 10%). Different sets of experiments are conducted to evaluate the efficacy of UMOTS with 0%, 1%, 5%, and 10% of uncertainties. Moreover, UMOTS is compared to a greedy-based optimization approach and some state-of-the-art works. Finally, the computation complexity of UMOTS is investigated.

The efficacy of UMOTS to find the robust Pareto frontier in the presence of uncertainties is studied in the 2nd set of experiments. In these experiments, three levels of uncertainties (i.e., 1%, 5% and 10%) are injected in design parameters and UMOTS is used to find the Pareto frontiers to objective functions. UMOTS indicates the effect of uncertainty injection and gains the Robust Pareto frontiers in each level. (III) The third category compares the capability of UMOTS in finding the optimum solutions and forming the Pareto fronts with the greedy algorithm. For both deterministic and uncertainty injection implementations. (IV) The fourth class compares UMOTS to a related meta-heuristic-based method, EGA-TA [28], in terms of performance factors such as Scheduling length Ratio (SLR) and Speed up parameters which are explained in the end of the following section.

B The correctness and effectiveness of UMOTS.

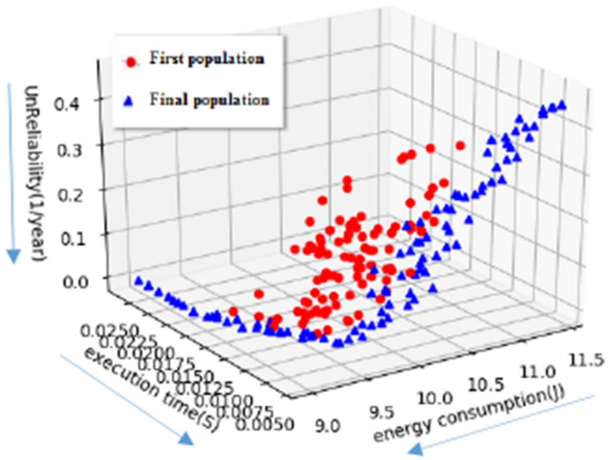


Fig. 8 Comparison of the Pareto frontiers of the first population (red circles) and final one (blue triangles) obtained by UMOTS for ABS and ACC benchmarks (color figure online)

In this section, the efficiency of the genetic optimization algorithm used in UMOTS for design space exploration is investigated. So, no uncertainty is injected in UMOTS and meanwhile, the Pareto frontier of the first and final solutions are compared.

Figure 8 shows the three-dimensional (3D) plot of the Pareto frontiers of ABS and ACC benchmarks in design space of execution time, energy consumption, and unreliability. X-, Y-, and Z-axis of this figure, respectively, show the energy consumptions, execution time, and unreliability of the solutions (i.e., task scheduling). Comparing the Pareto frontiers of the first population (shown in red circles) and the final one (demonstrated in blue triangles), it can be observed that, significant improvements in all three objectives are achieved; the solutions have moved in the

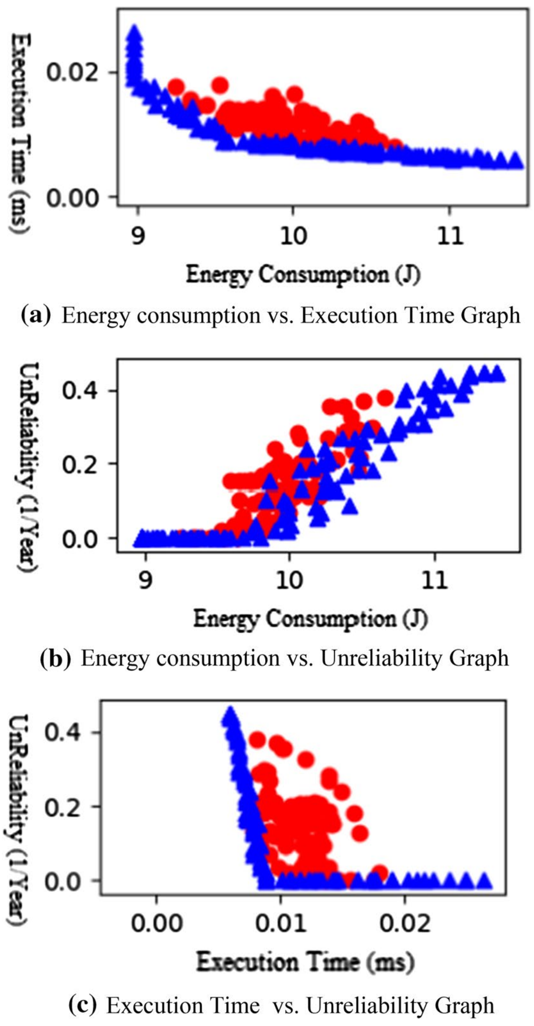


Fig. 9 Two-dimensional graphs of the Pareto frontiers of the first population and the final one for ABS and ACC benchmarks

direction of the arrows shown in the figure to less execution time, less energy consumption, and less unreliability.

The results are re-displayed using two-dimensional (2D) graphs in Fig. 9 to show them more clearly. For example, Fig. 9a compares the results of energy consumption vs. execution time in the first population and the final one. Similarly, Fig. 9b, c show the comparison of the parameters of energy consumption vs. unreliability and execution time vs. unreliability, respectively. As can be seen in Fig. 9, the Pareto set of the final population in all cases dominates the initial population, which means that UMOTS improves the initial task scheduling solutions considering all three fitness functions. For example, in Fig. 9a, in the solutions with less energy consumptions, the execution time is higher because in this case, PEs with less energy consumption are selected in the obtained task scheduling and hence, the execution time of designed system becomes high.

UMOTS is also used for design space exploration for an E3S benchmark, auto. indust-mocsyn, with no uncertainty injected. Figures 10 and 11, respectively, show the 3D and 2D plots of the Pareto frontiers obtained for this benchmark. As can be seen in figures, the Pareto frontiers of the final solutions dominate the first one regarding the three objectives in the design.

Figures 12 and 13 show the 3D and 2D view of the Pareto frontiers of the first and final population for consumer-mocsyn benchmark from E3S suit. It is notable that, since the total execution time of the tasks on the PEs is negligible regardless of the PE type, the best and worst execution times are close together. This property shows its effect in other parameters and plots as the duration time of executing a task on a PE affects the unreliability and energy consumption of the system.

III. Design space exploration considering different levels of uncertainty.

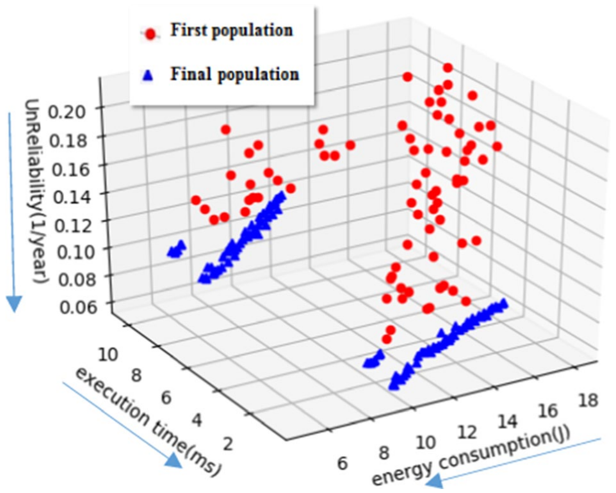
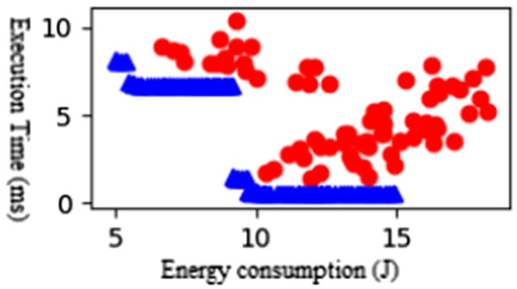
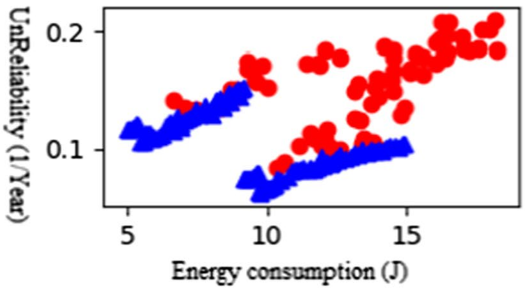


Fig. 10 Comparison of the Pareto frontiers of the first population (red circles) and final one (blue triangles) obtained by UMOTS for auto.indust-mocsyn benchmark (color figure online)

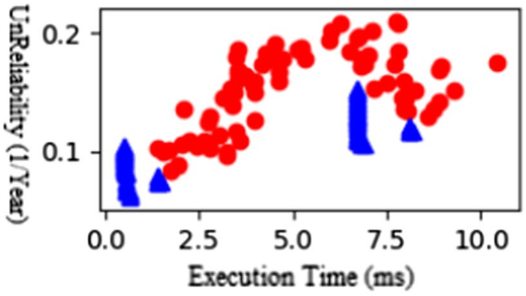
Fig. 11 Two-dimensional graphs of the Pareto frontiers of the first population and the final one for auto.indust-mocsyn benchmark



(a) Energy consumption vs. Execution Time Graph



(b) Energy consumption vs. Unreliability Graph



(c) Execution Time vs. Unreliability Graph

The ability to generate these 3D Pareto frontiers compromised of robust solution points (robust in the sense described in section II.F) represents on the main contributions of UMOTS. The efficiency of UMOTS for uncertainty-aware design space exploration is investigated considering different uncertainty levels. These experiments can help to study the impacts of the uncertainties imposed by the current or future technology nodes to the embedded system design. For example, suppose that the uncertainty of a given technology be 5%, but this value is not completely guaranteed, too. UMOTS is capable of studying how the solutions of deterministic approach would be changed when the uncertainty affects the design. Moreover, it can answer that, how the final uncertainty-aware solution for uncertainty 5% would

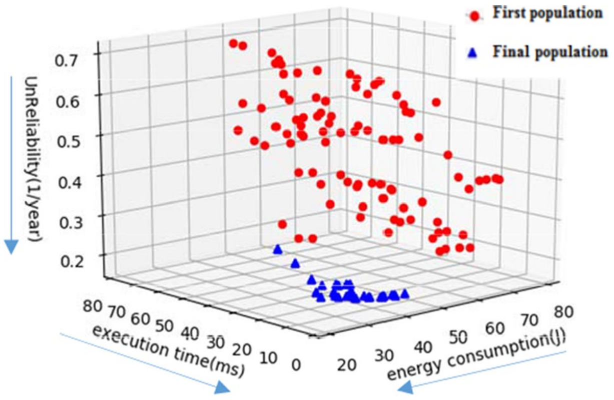


Fig. 12 Comparison of the Pareto frontiers of the first population (red circles) and final one (blue triangles) obtained by UMOTS for consumer-mocsyn benchmark (color figure online)

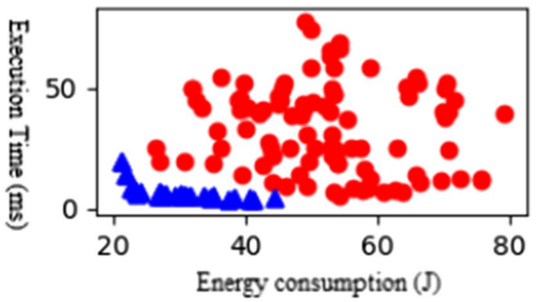
be affected if the given uncertainty amount itself is varied. These types of results can help the designer to investigate how the solution moves in the 3D design space and whether it meets the desired design constraint for the application at hand. Hence, UMOTS is applied to find the best task scheduling while uncertainties at 1%, 5%, and 10% levels are injected to the design parameters of different benchmarks. In order to compare the uncertainty-aware solutions with the deterministic approach, we also add the Pareto frontiers of the final solutions with no uncertainty injection (0% uncertainty) to the plots.

Considering the deterministic case as the baseline, by injecting uncertainty, the deterministic values for the parameters are replaced with the generated samples from the probability distributions considered to model the uncertainty. The distributions are characterized by a certain mean along with a standard deviation which is computed based on the uncertainty level determined by the designer (Table 1). Hence, the deterministic solution is changed into a set of solutions modeled as a probability distribution, whose 95 percentile estimate shows the robust solution which is used for finding the robust Pareto frontier. The location of this solution in design space is probably changed compared to the location of the deterministic design solution. The extent of this difference is related to the level of injected uncertainty. In the following, this property is investigated in task scheduling of different applications.

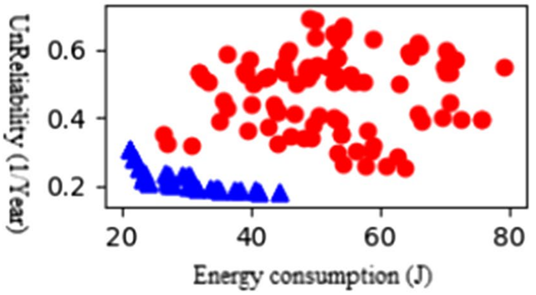
Figure 14 shows the 3D plot of the Pareto frontiers of different applications (i.e., ABS and ACC benchmarks, auto.indust-mocsyn, consumer-mocsyn from E3S benchmarks) with different levels of uncertainty. X-, Y-, and Z-axis, respectively, show the energy consumptions, unreliability, and execution time of the obtained solutions (i.e., task scheduling). As can be observed, the solutions obtained by deterministic approach (0% uncertainty) are off from the robust design solution points found by UMOTS for a given amount of uncertainty. It is notable that, UMOTS can identify this change and quantify each of the solutions in terms of the triple objectives per given level of uncertainty.

IV. Comparison of UMOTS with greedy-based approach.

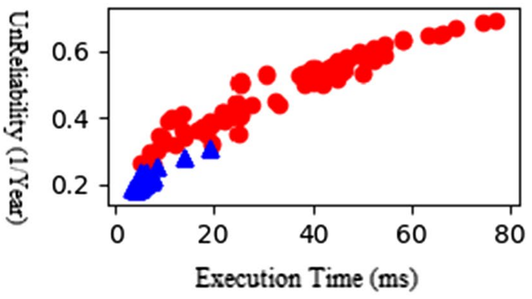
Fig. 13 Two-dimensional graphs of the Pareto frontiers of the first population and the final one for consumer-mocsyn benchmark



(a) Energy consumption– Execution Time Graph

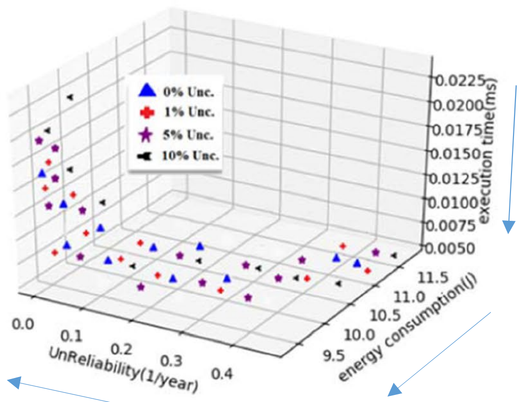


(b) Energy consumption – Unreliability Graph

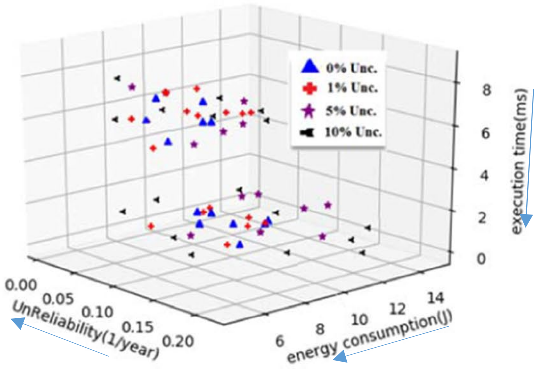


(c) Execution Time – Unreliability Graph

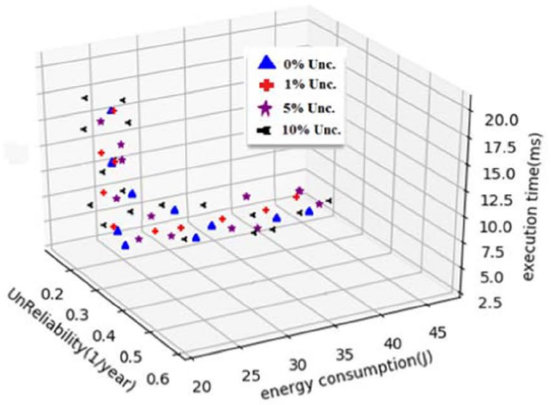
We compare the efficiency of UMOTS in finding the optimal solutions under uncertainty with an uncertainty-aware greedy-based approach. To this end, for each optimization objective (i.e., execution time, energy consumption, and reliability), an uncertainty-aware task scheduling based on greedy optimization algorithm is developed; i.e., each time, an objective is considered as the objective function in the greedy-based task scheduling and then, in each one, the task scheduling decisions are made based on a greedy approach in which, a MC simulator with 1000 iterations is employed in the objective evaluation step of the algorithm. To find a deterministic output from the objective evaluation, 95 percentile point of the distribution of the output is found and used in the greedy-based task scheduling algorithm. The greedy algorithm is implemented for all four levels of uncertainty (i.e., 0%, 1%, 5%,



(a) Comparison of different level of uncertainty in ABS/ACC benchmark



(b) Comparison of different level of uncertainty in E3S benchmark – Auto indust-mocsyn.



(c) Comparison of different level of uncertainty in E3S benchmark – Auto indust-mocsyn.

Fig. 14 Comparison of different level of uncertainty blue-triangle is 0% uncertainty injection (deterministic approach), 1% green-plus sign, 5% purple-star sign and 10% black-triangle left (robust approach) (color figure online)

Table 2 Comparison of UMOTS with uncertainty-aware greedy-based task scheduling method in 0% uncertainty injection

	UMOTS	Greedy
a		
Energy consumption (J)	11.244454	12.4410
Execution time (ms)	0.005889	0.00667
Unreliability (1/year)	0.448521	0.517695
b		
Energy consumption (J)	8.977705	9.774681
Execution time (ms)	0.026333	0.02
Unreliability (1/year)	0.000010	0.000011
c		
Energy consumption (J)	8.977741	9.774681
Execution time (ms)	0.023667	0.02
Unreliability (1/year)	0.000010	0.000011

Table 3 Comparison of UMOTS with uncertainty greedy-based task scheduling method in 1% uncertainty injection

	UMOTS	GREEDY
a		
Energy consumption (J)	11.392077	12.4992
Execution time (ms)	0.005762	0.006594
Unreliability (1/year)	0.453741	0.52621
b		
Energy consumption (J)	8.711	9.690569
Execution time (ms)	0.026520	0.02390
Unreliability (1/year)	0.18430	0.21321
c		
Energy consumption (J)	8.730	9.690569
Execution time (ms)	0.023633	0.02390
Unreliability (1/year)	0.18430	0.21321

and 10%) for this comparative experiment on ABS and ACC benchmarks. The final output of the uncertainty-aware greedy-based task scheduling is compared with the best solution among the Pareto frontier of each objective obtained from UMOTS. Tables 2, 3, 4, and 5 show the numerical results of the scheduling solution for 0% (deterministic case), 1%, 5%, and 10% uncertainty injection in the ABS/ACC benchmark, respectively. Furthermore, for each uncertainty level, there are three tables regarding the three objective functions; e.g., Table 2 includes parts a, b and c representing the obtained scheduling solutions when the objective function in the greedy algorithm is, respectively, execution time, energy consumption, and unreliability.

Table 2 shows the deterministic case in which the no uncertainty injection is performed on the greedy-based scheduling and UMOTS. The results in Table 2a show that, the best execution time obtained from the greedy algorithm is about 12.4 ms while it is about 11.2 ms in UMOTS. Additionally, Table 2b shows that the energy consumption of the schedule with the lowest energy obtained from the greedy algorithm is about 9.7 j while UMOTS algorithm has been able to obtain the schedule and mapping with the energy consumption of 8.9 j. The main reason is that, UMOTS takes advantages of a genetic algorithm-based optimization in which better solutions may be found due to its intrinsic diversity and randomness.

The results of injecting higher uncertainty amounts reported in the following tables indicate that the range of changes is proportional to the amount of uncertainty injected. For instance, the execution time is in the wider range from (8.71 ms–11.39 ms) in comparison to no uncertainty injection case when the execution time is in the range of (11.24 ms–8.98 ms) and this range will get wider as we increase the uncertainty level. It happens since by increasing the uncertainty we explore more area and possibilities in the solution space. This manner is same for energy consumption and reliability.

E Comparison of UMOTS with other heuristic-based task scheduling algorithms.

In this section, UMOTS is compared with the well-known task scheduling algorithms used heuristic optimization algorithms (i.e., HSHD [3], EGA-TS [28], HEFT_T [38], and BGA [39]). In order to have a fair comparison, uncertainty-aware versions of these task scheduling algorithms are developed by embedding MC simulation-based into their parameter evaluation steps. To this end, 100 random graphs with size 20 nodes are created using an automatic graph generator tool (i.e., TGFF tool [28]). In random graphs, the characteristics of nodes and arcs (such as we create 15 nodes with 19 edge connecting them or 21 nodes with 30 connections) are randomly generated. As the comparison criteria, two standard criteria called scheduling length Ratio (SLR) and Speedup are used:

- Scheduling length ratio (SLR).

The length of the scheduled output graph is an important factor in comparing different scheduling algorithms. SLR is known as a suitable metric for comparing different algorithm in terms of the length of the scheduled graph which is computed as follows [40–42]:

$$\mathrm{SLR} = \frac{\text{makespan}}{\sum_{T_i \in CP_{\min}} \min p_k \in p\left(W\left(T_i, P_k\right)\right)} \tag{10}$$

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 (i.e., the path with the most number of nodes and links in the graph), and $W(T_i, P_k)$ indicates the time of executing the task T_i on the P_k processor based on the output of the task scheduling. This criterion divides the achieved execution time (makespan), into the optimum execution time of the scheduling ($\sum_{T_i \in CP_{\min}} \min p_k \in p\left(W\left(T_i, P_k\right)\right)$). The average of SLR computed on several graphs is considered as comparison criterion to prevent the comparison dependency to the structure of the graph.

Table 4 Comparison of UMOTS with uncertainty greedy-based task scheduling method in 5% uncertainty injection

	UMOTS	GREEDY
a		
Energy consumption (J)	11.525093	12.732227
Execution time (ms)	0.00525	0.006041
Unreliability (1/year)	0.4721	0.55120
b		
Energy consumption (J)	8.4553	9.6515
Execution time (ms)	0.027333	0.02550
Unreliability (1/year)	0.15321	0.19574
c		
Energy consumption (J)	8.490	9.6515
Execution time (ms)	0.025100	0.02550
Unreliability (1/year)	0.15220	0.19574

Table 5 Comparison of UMOTS with uncertainty greedy-based task scheduling method in 10% uncertainty injection

	UMOTS	GREEDY
a		
Energy consumption (J)	11.804616	13.023619
Execution time (ms)	0.00480	0.005713
Unreliability (1/year)	0.5312	0.61031
b		
Energy consumption (J)	7.9531	10.0412
Execution time (ms)	0.032620	0.028124
Unreliability (1/year)	0.11820	0.13190
c		
Energy consumption (J)	7.9820	10.0412
Execution time (ms)	0.030190	0.028124
Unreliability (1/year)	0.11640	0.13190

- Speedup.

Speedup metric is computed from dividing serial execution of tasks on the fastest processor by total completion time of tasks in parallel form; i.e., [41, 42]:

$$\text{Speedup} = \frac{\min P_k \in p\left(\sum_{T_i \in T} W\left(T_i, P_k\right)\right)}{\text{Makespan}} \tag{11}$$

Speedup answers to this question that how much improve we gain in parallel execution in comparison to serial execution of tasks on the fastest processor or comparison, the average of Speedup computed on several graphs is reported.

Figure 15 shows the SLR comparison results which are normalized to the SLR of UMOTS. UMOTS provides the scheduling solution with an execution time equal to the optimum execution time of the scheduling. Moreover, it performs similar to EGA-TS which is a single objective task scheduling that optimizes the execution time in its task scheduling.

The obtained Speedup values for different algorithms are shown in Fig. 16. As can be seen, UMOTS also outperforms as well a single objective heuristic method EGA-TS. The results show that, if all the tasks are serially executed on the fastest PE, it takes 2.89 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, BGA

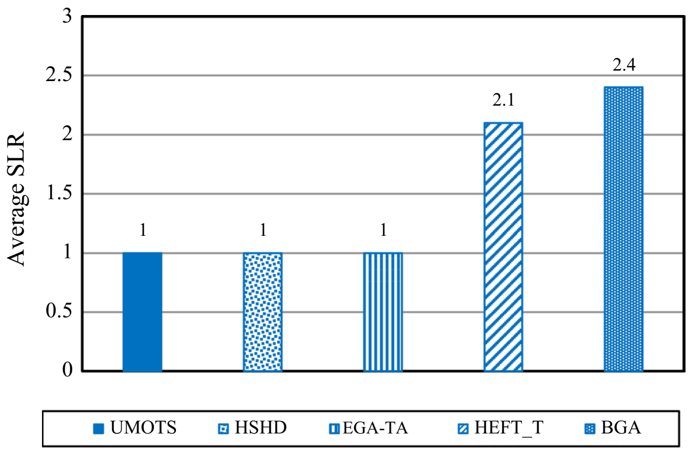


Fig. 15 Comparison between UMOTS and other task scheduling methods in SLR metric

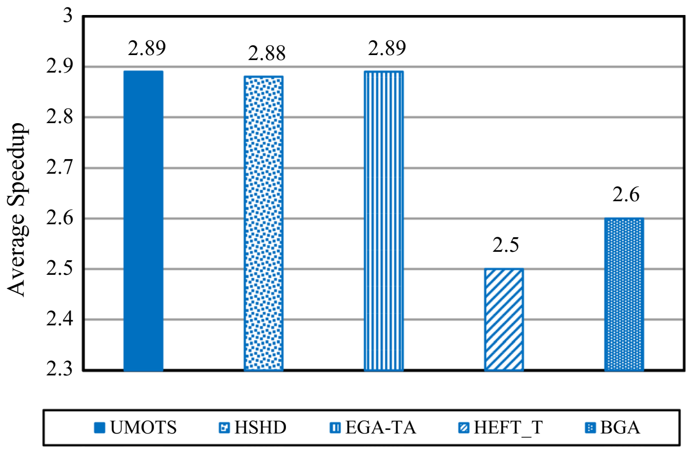


Fig. 16 Comparison between UMOTS and other task scheduling methods in Speedup metric

provides scheduling solutions, which are 2.6 times shorter than the case in which the tasks are serially executed on the fastest PE.

These two comparisons show that, UMOTS provides similar results to the state-of-the art single objective heuristic method-based tasks scheduling algorithm while it covers other objectives (energy consumptions and reliability) and more importantly, is capable of considering uncertainty in parameters of the embedded system components.

F Computational complexity and convergence of UMOTS.

For a given task graph size, the computational complexity of UMOTS primarily depends on the two factors: (1) the number of iterations of the outer loop which represents the number of solutions created by the genetic algorithm, and (2) the number of inner loop iterations which corresponds to the number of MC simulations. In order to investigate the runtime complexity of the outer loop and its scalability, the runtime of UMOTS vs. the number of iterations of the outer loop is shown in Fig. 17. In each iteration, the number of iterations of the inner loop (MC simulation) is set to 1000 which is found to be an acceptable threshold for convergence. The results show the runtime scalability of UMOTS as the runtime is growing linearly.

It is also interesting to know about the least number of iterations of MC simulation required to obtain a reasonable convergence in uncertainty estimation. To answer this, we have studied the impact of the number of MC iterations on the calculation of objective functions; i.e., we have changed the number of iterations of MC simulation in calculation of execution time, energy consumption, and unreliability and plot the obtained results, respectively, in Fig. 18a–c. As can be observed, in lower number of iteration of MC simulations, the variability of the estimation results are significant while after 1000 iterations of MC simulations, the estimation results converges to a stable value.

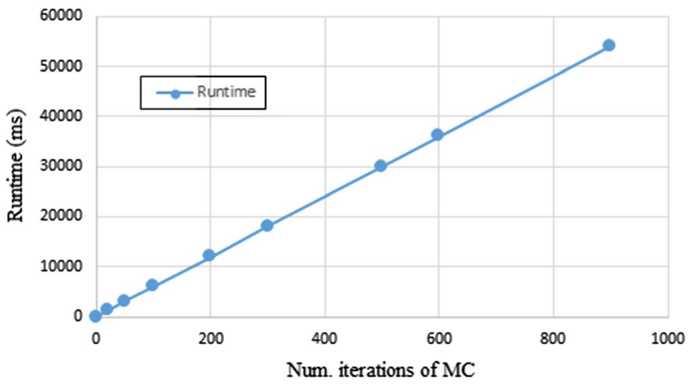
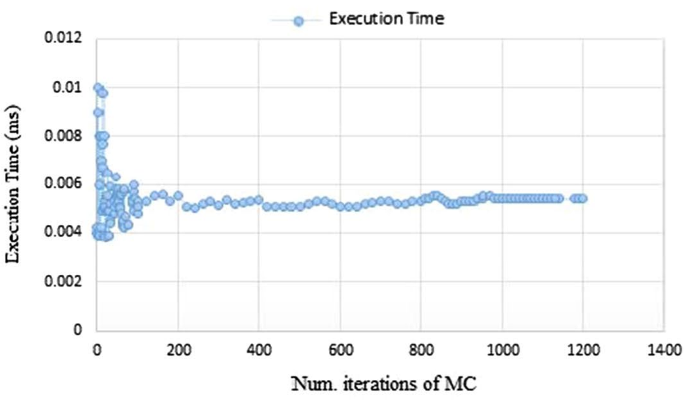
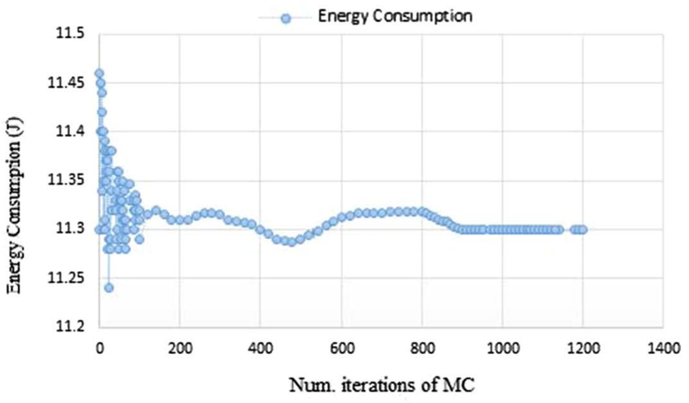


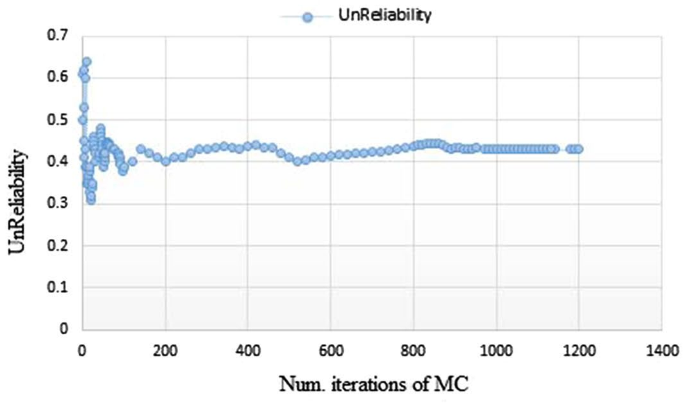
Fig. 17 Computational runtime of UMOTS versus the number of iterations of the NSGA-II genetic Algorithm



(a) Execution time



(b) Energy consumption



(c) Unreliability

Fig. 18 MC simulation convergence in uncertainty-aware estimation of different objective functions

6 Conclusion

Due to the increased manufacturing process variations and heterogeneity of applications, the embedded system designs encounter considerable levels of uncertainties which may reduce the accuracy and efficiency of traditional design approaches with deterministic values for the design component parameters. This paper presents UMOTS, a task scheduling framework for embedded systems. The main features of UMOTS are: (1) providing task scheduling solutions which are robust against the uncertainty, and (2) a multi-objective optimization for task scheduling considering reliability in addition to the execution time and energy consumption. In UMOTS, the uncertainty is considered in both software (task parameters) and hardware (processor and communication parameters) of the embedded system and handled using a Monte-Carlo-based approach. Moreover, UMOTS takes advantages of a genetic algorithm-based multi-objective optimization approach to find the Pareto frontier in the objective space formed by performance, energy consumption, and reliability. The experimental results based on real-application task graphs show the efficiency of UMOTS in terms of Scheduling Length Ratio (SLR) and speedup. UMOTS provides 27.8% better SLR and 28.6% improved speedup in comparison to HSHD, one state-of-the-art task scheduling algorithm. Moreover, UMOTS finds the robust Pareto frontier with 1%, 5% and 10% uncertainty in design indicators with respect to design limitations. UMOTS is able to help the designer to choose the best solution based on the given embedded system; e.g., for an energy-constrained system the designer can choose the solution which are optimized in terms of energy consumption while considering the constraints the impact of the chosen solution on the execution time and reliability of the system.

Appendix

To prove that the genetic operators maintain the order of tasks and would not result in invalid orders, they have to fulfill the following conditions [42]:

- *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; i.e., competence and uniqueness conditions.*

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 [42].

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.

References

1 Kristina B, Sanja C, Alen J (2021) Systematic review of methodologies for the development of embedded systems. Int J Adv Comput Sci Appl (IJACSA). https://doi.org/10.14569/IJACSA.2021.0120149
2. Blašković K, Čandrlić S (2018) DEM4RTS: software development methodology for special case of real-time closed-loop control systems. Ann DAAAM & Proc 29
3. Prongnuch S, Sitjongsataporn S, Wiangtong T (2020) A heuristic approach for scheduling in heterogenous distributed embedded systems. Int J Intell Eng Syst 13(1):135–145
4. Deng Z, Cao D, Shen H et al (2021) Reliability-aware task scheduling for energy efficiency on heterogeneous multiprocessor systems. J Supercomput. https://doi.org/10.1007/s11227-021-03764-x
5. Ullman J (1975) NP-complete scheduling problems. J Comput Syst Sci 10(3):384–393
6. Abdi A, Girault A, Zarandi H (2019) (2019) ERPOT: a quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores. IEEE Trans Parallel Distrib Syst 30(10):2193–2210
7. Lombardi M, Milano M, Benini L (2013) Robust scheduling of task graphs under execution time uncertainty. IEEE Trans Comput 62(1):98–111
8. Yeh D, Peh L, Borkar S, Darringer J, Agarwal A, Hwu W (2008) Thousand-core chips [roundtable]. IEEE Des Test Comput 25(3):272–278
9. Yamamoto A, Ababei C (2104) Unified reliability estimation and management of NoC based chip multiprocessors. Microprocessors Microsyst 38(1):53–63

10. Erbas C, Cerav-Erbas S, Pimentel A (2006) Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. IEEE Trans Evol Comput 10(3):358–374
11. Kianzad V, Bhattacharyya S (2004) CHARMED: a multi-objective co-synthesis framework for multi-mode embedded systems. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors. https://doi.org/10.1109/ASAP.2004.1342456
12. Nedjah N, da Silva M, de Macedo ML (2011) Customized computer-aided application mapping on NoC infrastructure using multi-objective optimization. J Syst Architect 57(1):79–94
13. Kang S-H, Yang H, Schor L, Bacivarov I, Ha S, Thiele L (2012) Multi-objective mapping optimization vi problem decomposition for many-core systems. 2012 IEEE 10th Symposium on Embedded Systems for Real-time Multimedia, pp 28–37
14. Ascia G, Catania V, Palesi M (2004) A GA-based design space exploration framework for parameterized system-on-a-chip plathforms. IEEE Trans Evol Comput 8(4):329–346
15. Balarin F, Watanabe Y, Hsieh H, Lavagno L, Passerone C, Sangiovanni-Vincentelli A (2003) Metropolis: an integrated electronic system design environment". IEEE Comput 36(4):45–52
16. Cassidy A, Paul J, Thomas D (2003) Layered, multi-threaded, high-level performace design. 2003 Design Automation and Test in Europe Conference and Exhibition.https://doi.org/10.1109/DATE.2003.1253728
17. Domer R, Gerstlauer A, Peng J, Shin D, Cai L, Yu H, Abdi S, Gajski D (2008) System-on-chip environment: a Spec C-based framework for heterogeneous MPSo design. EURASIP J Embed Syst. https://doi.org/10.1155/2008/647953
18. Nikolov H, Stefanov T, Deprettere E (2008) Systematic and automated multiprocessor system design, programming, and implementation. IEEE Trans Comput Aided Des Integr Circuits Syst 27(3):542–555
19. Pillai A, Singh K, Saravanan V, Anpalagan A, Woungang I, Barolli L (2017) A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems. Soft Comput 22(10):3271–3285
20. Zhang J, Zhou D, Yang Y, Lai R, Gao X (2010) Energy optimization of NoC based on voltage-frequency islands under processor reliability constraints. J Electron Inf Technol 33(9):2205–2211
21. Zhou J, Wei T, Chen M, Hu XS, Ma Y, Zhang G, Yan J (2018) Variation-aware task allocation and scheduling for improving reliability of real-time MPSoCs. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp 171–176). IEEE
22. Meedeniya I, Aleti A, Grunske L (2012) Architecture-driven reliability optimization with uncertain model parameters. J Syst Softw 85(10):2340–2355
23. Guan W, Moghaddam M, Ababei C (2018) Uncertainty aware mapping of embedded systems for reliability, performance, and energy. 2018 19th International Symposium on Quality Electronic Design (ISQED). https://doi.org/10.1109/ISQED.2018.8357284
24. Bandyszak T, Daun M, Tenbergen B, Kuhs P, Wolf S, Weyer T (2020) Orthogonal uncertainty modeling in the engineering of cyber-physical systems. IEEE Trans Autom Sci Eng 17(3):1250–1265
25. Li F, Liao TW, Cai W, Zhang L (2020) Multitask scheduling in consideration of fuzzy uncertainty of multiple criteria in service-oriented manufacturing. IEEE Trans Fuzzy Syst 28(11):2759–2771
26. Muhuri PK, Nath R, Shukla AK (2020) Energy efficient task scheduling for real-time embedded systems in a fuzzy uncertain environment. IEEE Trans Fuzzy Syst
27. Pimentel A, Erbas C, Polstra S (2006) A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Trans Comput. https://doi.org/10.1109/TC.2006.16
28. Akbari M, Rashidi H, Alizadeh S (2017) An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems. Eng Appl Artif Intell 61:35–46
29. Deb K, Agrawal S, Pratap A, Meyarivan T (2002) A fast and elitist multiojective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
30. Das A, Al-Hashimi BM, Merrett GV (2016) Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems. ACM Trans Embed Comput Syst (TECS). Embed Comput Syst 15(2):24–34
31. Sheikh HF, Ahmad I (2016) Sixteen heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-cores. ACM Trans Parallel Comput (TOPC). https://doi.org/10.1145/2948973

32. Abdi A, Zarandi HR (2018) Hystery: a hybrid scheduling and mapping approach to optimize temperature, energy consumption and lifetime reliability of heterogeneous multiprocessor systems. J Supercomput. https://doi.org/10.1007/s11227-018-2248-2
33. Srinivasan J, Adve S, Bose P, Rivers J (2005) Lifetime reliability: toward an architectural solution. IEEE Micro 25(3):70–80
34. J. E. D. E. Council, failure mechanisms and models for semiconductor devices, Tech. Rep. JEP122H. 2016, https://www.jedec.org/
35. Erbas C (2006) System-level modeling and design space exploration for multiprocessor embedded systems-on-chip architectures. Ph.D. thesis, Faculty of Science, Amsterdam University
36. Embedded System Synthesis Benchmark Suite (E3S) (2008) http://ziyang.eecs.umich.edu/dickrp/e3s/. Accessed 23 Aug 2018
37. Das A, Kumar A, Veeravalli B, Bolchini C, Miele A, (2014) Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in mpsocs. Proceedings of the Conference on Design, Automation & Test in Europe, pp 61–69
38. Topcuoglu H, Hariri S, Wu MY (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
39. Gupta S, Kumar V, Agarwal G (2010) Task scheduling in multiprocessor system using genetic algorithm. 2010 Second International Conference on Machine Learning and Computing. https://doi.org/10.1109/ICMLC.2010.50
40. Burkimsher A, Bate I, Indrusiak LS (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
41. Ijaz S, Munir E, Anwar W, Nasir W (2013) Efficient scheduling strategy for task graphs in heterogeneous computing environment. Int Arab J Inf Technol 10(5):486–492
42. Kumar V, Katti CP (2014) A scheduling approach with processor and network heterogeneity for grid environment. Int J Comput Sci Eng 6(1):42–48

Authors and Affiliations

Mohsen Raji[1] · Mohaddaseh Nikseresht[1]

Mohaddaseh Nikseresht
m.nikseresht@cse.shirazu.ac.ir

[1] School of Electrical and Computer Engineering, Shiraz University, Shiraz, Iran