Near-Optimal Energy-Efficient Partial-Duplication Task Mapping of Real-Time Parallel Applications

Minyu Cui, Angeliki Kritikakou, Lei Mo, Emmanuel Casseau

Noname manuscript No.
(will be inserted by the editor)

Near-optimal Energy-Efficient Partial-Duplication Task Mapping of Real-Time Parallel Applications

Minyu Cui, Angeliki Kritikakou, Lei Mo, Emmanuel Casseau

Abstract Minimizing energy consumption, as well as meeting real-time and reliability constraints, are major goals during system deployment. When complex platforms, such as multicore architectures with DVFS, and parallel applications are considered, these goals are significantly impacted by task mapping. To minimize energy consumption, while meeting real-time and reliability constraints, this work proposes a task mapping approach to jointly solve the problem of task allocation, task scheduling, frequency assignment, and task duplication. A novel heuristic algorithm is proposed to cope with this NP-hard problem, consisting of a pruning phase, which maintains only the task configurations that satisfy reliability constraints, and a mapping phase, which minimizes total energy consumption under real-time and precedence constraints. The obtained results show that the proposed heuristic obtains near-optimal results, with low computation time, compared to optimal solvers, while it achieves better energy consumption and finds slightly more solutions compared to other heuristic approaches.

Keywords Fault tolerant · Task mapping · DVFS · Real-time execution · Reliability · Energy minimization

Acknowledgements This work is funded by China Scholarship Council (CSC).

Minyu Cui
Univ Rennes, INRIA, CNRS, IRISA, France, E-mail: minyu.cui@irisa.fr

Angeliki Kritikakou
Univ Rennes, INRIA, CNRS, IRISA, France, E-mail: angeliki.kritikakou@irisa.fr

Lei Mo
School of Automation, Southeast University, China, E-mail: lmo@seu.edu.cn

Emmanuel Casseau
Univ Rennes, INRIA, CNRS, IRISA, France, E-mail: emmanuel.casseau@irisa.fr

1 Introduction

1.1 Context

The execution of embedded real-time parallel applications on multicore platforms require guarantees for real-time execution [1–3] and reliability [1,4]. Due to technology size reduction, multicore platforms are susceptible to transient faults [1]. Reliability is defined as the probability of executing a task without faults. Typical techniques to increase reliability are the execution of tasks with high frequency [5] and task replication [1, 4]. However, both techniques can lead to large energy consumption. Dynamic Voltage and Frequency Scaling (DVFS) is a well-known energy management technique, which optimizes energy consumption by scaling down the processor supply voltage and frequency [1, 4]. However, this has a negative impact on reliability, since more transient faults occur at a low voltage and frequency level [1, 4]. Therefore, during the task mapping not only task allocation and scheduling, but also frequency assignment and task replication should be incorporated into the optimization process to achieve energy efficiency, while meeting real-time and reliability constraints.

1.2 Related work and motivation

Table 1 summarises representative State-of-the-Art (SoA) task mapping approaches that minimize energy consumption considering DVFS, under Real-Time (RT.) and Reliability (R.) constraints. Tasks are Independent (I.) or Dependent (D.). The platform consists of Homogeneous processors(HO.), Heterogeneous processors (HE.), or Single (S.) processor. The applied fault tolerance policy can be task Recovery (Rec.) or task Replication (Rep.). A task is executed successfully, if at least one replica is executed without faults [1]. Last, the proposed solving method can be Optimal (O.) or Heuristic-based (H.).

Approaches exist without applying a fault tolerance policy, e.g., the task mapping problem is decomposed into a sub-problem that satisfies the reliability constraint and another that minimizes resources [5] and a whale optimization algorithm is proposed [6]. In [7], the scheduling algorithm for dependent multi-version tasks is studied based on Forward List Scheduling with the goal of minimizing the total energy consumption under real-time constraint. Approaches execute a recovery task, e.g., individual [8] or shared [2, 9], or both [10], with maximum frequency, exploring available time slack to meet the reliability constraint. Last, approaches apply task replication. Some works decide the required number of replicas per task always to meet the reliability constraint, e.g., for independent tasks on homogeneous platform [1] and dependent tasks on heterogeneous platforms [4]. In [1], all replicas of a task are executed at the same frequency, whereas in [4], no real-time constraints are taken into account. In [11], the number of replicas per task is given. In the first phase, half-plus-one copies per task are executed. If a fault occurs,

Table 1: Comparison with representative SoA approaches

Ref.	Task model		Fault tol.		Platform			Constraints		Solution	
	I.	D.	Rec.	Rep.	HO.	HE.	S.	RT.	R.	O.	H.
[1]	√			√	√			√	√		√
[4]		√		√		√			√		√
[5]		√				√			√		√
[6]		√				√		√	√		√
[7]		√				√		√			√
[2, 8, 10]		√	√			√		√	√		√
[9]		√	√				√	√	√		√
[3]		√	√	√	√			√			√
[13, 15]	√			√	√			√	√	√	
[12]		√		√	√			(√)	(√)		√
[11]		√		√	√			√	√		√
[14]		√		√	√			√	√	√	
Prop.		√		√	√			√	√		√

the second phase is applied to execute the remaining number of copies. Other works decide among different reliability mechanisms for each task. In [12], a heuristic explores three reliability mechanisms in sequence to decide single task execution, task duplication, and task triplication, without the requirements of always meeting the real-time and reliability constraints. In [3], a heuristic determines which tasks to be duplicated, removing the need for a recovery task for all tasks. An optimal approach that decides which tasks to duplicate, taking into account reliability and real-time constraints is proposed in [13–15]. The main difference between [13, 15] and [14] is the task model, i.e., independent tasks are considered in [13, 15] while tasks with dependencies are considered in [14].

Our work (Prop.) studies the same problem as in [14], i.e., partial duplication task mapping of tasks with dependencies and task-level DVFS. However, an optimal algorithm is presented in [14], while a heuristic algorithm that provides near-optimal, but less time-consuming solutions, is proposed in this paper. The experimental section provides a comparison between the optimal and the heuristic approaches.

Overall, optimal approaches are applicable only for small problem sizes and existing heuristics, either do not fully guarantee real-time or reliability constraints, or use a high number of replicas, leading to large energy consumption with a negative impact on execution time, and thus, potentially no feasible solution.

1.3 Contributions

This work addresses the task mapping problem of parallel applications on homogeneous multicore platforms with DVFS, with the goal of minimizing total energy consumption, under real-time and reliability constraints. An effective heuristic is proposed, considering multiple non-functional properties

of execution time, energy, and reliability, that combine task allocation, task scheduling, frequency assignment, and selective task duplication. Unlike the majority of SoA techniques, original and duplicated tasks can have different operating frequencies, being more suitable for real-world systems [16]. A pruning phase maintains only the task configurations that satisfy reliability constraints. A mapping phase minimizes total energy consumption under real-time and precedence constraints. Last but not least, the proposed method is evaluated both with task graphs from real applications and randomly generated graphs. The experiment results show that our approach provides near-optimal energy savings, with low computation time compared to optimal solvers. At the same time, it has better energy consumption and feasibility compared to other heuristics.

The rest of the paper is organized as follows. Section 2 introduces the system model. Section 3 presents the formulation of the proposed approaches. Section 4 presents the evaluation results. Finally, Section 5 concludes this study.

2 System Model

Table 2 shows the main notations. For the sake of paper presentation, when original and duplicated tasks must be distinguished in mathematical formulations, the superscript $k \in \{o, d\}$ indicates the original task (^{o}) or the duplicated task (^{d}). If no superscript exists, the mathematical formulation is valid for both.

2.1 Task Model

This work considers a real-time application modelled as a Directed Acyclic Graph (DAG) $G(\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ denotes the set of N frame-based, non-preemptive, dependent tasks $\boldsymbol{N} = \{\tau_0^o, \ldots, \tau_{N-1}^o\}$, while $\mathbb{E}$ represents the edges, corresponding to the tasks' precedence constraints. Tasks are released at time 0 and have a global deadline D, given by the application frame. The release period of the task set is assumed to be longer than its global deadline. We focus on a single frame, which implies only one job of the same task can be active simultaneously. If a task has no predecessors (successors), it is an entry task τ_{entry} (exit task τ_{exit}). If a graph has multiple entry or/and exit tasks, then a dummy entry or/and exit task can be added into the graph to meet our model. A task is ready for execution when all its predecessors have been completed. Each task τ_i is described by a tuple $\{W_i, R_i^{th}\}$, where W_i is the Worst Case Execution Cycles (WCEC) and R_i^{th} is its reliability threshold which is given as input and depicts the reliability requirement of the task. Each task has its own reliability constraint, since functions of an application exhibit distinct vulnerabilities, due to variations in the spatial and temporal vulnerabilities of different instructions [12].

Notations	Definitions
τ_i^o/τ_i^d	the original/duplicated copy of task τ_i
(v_l, f_l)	the l^{th} voltage/frequency level
W_i	WCEC of task τ_i
D	the global deadline
R_i^{th}	reliability threshold of task τ_i
PL	priority list of task set
SL	schedule length of DAG G
EST_i	earliest start time of task τ_i
LFT_i	latest finish time of task τ_i
st_i	actual start time of task τ_i
ft_i	actual finish time of task τ_i
et_i	execution time of task τ_i
$slack_i$	time slack of task τ_i
$Pred\{\tau_i\}$	all immediate predecessors of task τ_i
$Succ\{\tau_i\}$	all immediate successors of task τ_i
$Bf\{\tau_i\}$	task set that is executed before task τ_i on the same processor
$Af\{\tau_i\}$	task set that is executed after task τ_i on the same processor
$avail[m]$	earliest available time of processor θ_m when it is ready to execute a task
SC_i	Scheduled Configuration of task τ_i in current task mapping
NC_i	New (checked) Configuration of task τ_i to do relaxation

Table 2: Main Notations and their definitions

2.2 Platform Model

The target platform has a shared-memory multicore architecture with M homogeneous processors $\boldsymbol{M} = \{\theta_0, \ldots, \theta_{M-1}\}$. Each processor supports DVFS and has L pairs of frequency/voltage level $\{(f_0, v_0), \ldots, (f_{L-1}, v_{L-1})\}$. As the relationship of voltage and frequency is almost linear [4,9,11], we use the term frequency scaling to express the simultaneous change of voltage and frequency. We consider intra-task DVFS. When task τ_i is executed at frequency f_l, its execution time is $\frac{W_i}{f_l}$. The power consumption is modeled as the sum of static power P_l^{sta} and dynamic power P_l^{dyn} considering a voltage/frequency level (v_l, f_l) [1,4]. Specifically,

$$P_l = P_l^{sta} + P_l^{dyn} = P_l^{sta} + C_{eff} v_l^2 f_l, \tag{1}$$

where C_{eff} is the effective switching capacitance.

2.3 Fault Model and Reliability

This work addresses transient faults, having a higher occurrence than permanent faults during the useful lifetime of the system [5]. During this period, the fault model, where the fault occurrence is given by a Poisson distribution with an average fault rate $\lambda(f_l)$ at frequency f_l, is typically used [4,5,9,11,17]. The failure rate per time unit of a processor at frequency f_l is given by

$$\lambda(f_l) = \lambda_0 \times 10^{d_0 \frac{f_{\max} - f_l}{f_{\max} - f_{\min}}}, \tag{2}$$

where $f_{\max} = \max_{\forall l}\{f_l\}$, $f_{\min} = \min_{\forall l}\{f_l\}$, λ_0 is the average failure rate of the processor corresponding to $f_{\max}$, and d_0 is a positive constant, indicating the sensitivity of failure rates to voltage scaling [4].

The reliability of task τ_i is defined as the probability of executing τ_i without any fault. Based on the exponential model in [1, 9], the reliability of a task executed at f_l is calculated as

$$R_i(f_l) = e^{-\varphi_i(f_l)}, \tag{3}$$

where $\varphi_i(f_l) = \lambda(f_l) \times et_{i,l}$, and $et_{i,l}$ is the execution time of task τ_i at frequency f_l. If the reliability of original task τ_i^o is larger than its reliability threshold R_i^{th}, the execution is considered as reliable [5] and the task reliability is given by $R_i = R_i^o$. Otherwise, the task τ_i^o is duplicated and executed on a different processor, since it is unlikely that the execution of both original and duplicated tasks on different processors fails [1, 3, 4]. The duplicated task τ_i^d has the same characteristics with the original task τ_i^o. When the duplicated task τ_i^d is executed, its reliability is R_i^d and depends on the frequency it is executed. Then, the task reliability, after duplication, is

$$R_i = 1 - (1 - R_i^o)(1 - R_i^d). \tag{4}$$

As our approach finds an offline heuristic-based task mapping solution, with the goal of minimizing energy consumption, we consider only task duplication, i.e., a single replica per task, in order not to unnecessarily increase the number of replicas, in case no faults occur. An online mechanism can be applied to further improve the energy consumption of our solution (by not executing the duplicated task, when the first execution is correct), and to deal with the low probability cases, where both original and duplicated tasks are faulty. In this work, cores are assumed to operate below a temperature threshold, where temperature impact on reliability is low [18].

2.4 Problem under study

Given a DAG task graph G and M processors, the goal is to minimize the total energy consumption by deciding the: 1) task duplication, 2) assignment of frequencies to tasks, 3) allocation of tasks to processors, 4) start time of tasks, subject to reliability, real-time and task precedence constraints.

3 Proposed Approach

To solve the problem under study, we propose a *Heuristic for Reliability-aware Fault-tolerant Task Mapping (H_RAFTM)*, described in Algorithm 1. The algorithm consists of two phases:

1. `Phase A` obtains, per task, the set of possible configurations that meet the reliability constraint, ordered in decreasing energy consumption.

2. `Phase B` obtains the application mapping, by allocating tasks to processors using the least total energy consumption, under task precedence and real-time constraints.

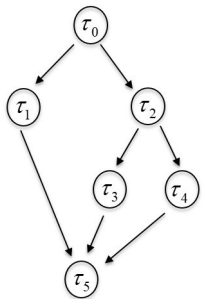
H_RAFTM is based on the following definitions:

Definition 1 (Configuration). *A configuration j of a task τ_i is denoted as $C_i^j = \{f_i^o, f_i^d, et_i^o, et_i^d, E_i^o, E_i^d, R_i\}$, where f_i^o (f_i^d) is the assigned frequency, et_i^o (et_i^d) is the required execution time, E_i^o (E_i^d) is the energy consumption of the original (duplicated) task, and R_i is the reliability of the task (taking into account task duplication). If the task is not duplicated, we have $f_i^d = et_i^d = E_i^d = 0$ which is considered as a dummy execution.*

Definition 2 (Task Mapping). *A mapping of a task τ_i, under the task configuration C_i^j, is denoted as $TM_i^{C_i^j} = \{\theta_i^o, \theta_i^d, st_i^o, st_i^d\}$, where θ_i^o (θ_i^d) is the allocated processor, and st_i^o (st_i^d) is the start time of the original (duplicated) task. If a task is not duplicated, then $f_i^d = 0$, and the duplicated task takes no execution time, i.e., its start time is equal with its finish time, $st_i^d = ft_i^d$.*

Definition 3 (Application Mapping). *The mapping of the application (AM) is given by the set of mappings of N original tasks and $S \subseteq N$ duplicated tasks. The mapping is valid if task precedence and real-time constraints are satisfied.*

The following paragraphs describe in detail the two phases of the proposed approach and illustrate them using the application DAG example of Fig. 1.



Part of RTE_5* with $W_5 = 1.4728 \times 10^8$ and $R_5^{th} = 0.9994$
*Processor model based on Table 3

C_5^j	f_5^o (GHz)	f_5^d (GHz)	et_5^o (msec)	et_5^d (msec)	E_5^o (mJ)	E_5^d (mJ)	R_5
C_5^0	0.801	-	18.387	-	0.77944	-	0.9908486
C_5^1	0.8291	-	17.764	-	1.02746	-	0.9966568
C_5^2	0.8553	-	17.220	-	1.36084	-	0.9986935
C_5^3	0.8797	-	16.742	-	1.81375	-	0.9994552
C_5^4	0.9027	-	16.315	-	2.43532	-	0.9997610
C_5^5	1.0	-	14.728	-	3.29633	-	0.9999926
C_5^6	0.801	0.801	18.387	18.387	0.77944	0.77944	0.9999163
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
C_5^{10}	0.801	1.0	18.387	14.728	0.77944	3.29633	~ 1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
C_5^{26}	1.0	1.0	14.728	14.728	3.29633	3.29633	~ 1

Fig. 1: Illustration example: Application DAG ($N = 6$) and part of RTE_5.

3.1 Phase A: Task configurations, under reliability constraint.

`Phase A` (Lines 1-9) is applied per task. For each task, a Reliability, execution Time, and Energy consumption (RTE) table is created based on all possible

Algorithm 1: Proposed H_RAFTM algorithm.

Require: Task graph (G) and set of processors ($\boldsymbol{M}$).
Ensure: Application mapping (AM).

`// Phase A`
1: **for** each task τ_i in $\boldsymbol{N}$ **do**
2: $RTE_i = \{C_i^j$: C_i^j is a configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each configuration bc in BC_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0 \wedge \min\{et_i^o, et_i^d\} \geq et^{bc} \wedge \sum\{E_i^o, E_i^d\} > E^{bc}\}$;
7: **end for**
8: $rPC_i = \{PC_i$: PC_i decreasing energy consumption$\}$;
9: **end for**
`// Phase B`
10: **for** each task τ_i in $\boldsymbol{N}$ **do**
11: Compute $rank_i$ (Eq. 9);
12: **end for**
13: $PL = \{\boldsymbol{N}$: ordered in decreasing $rank_i\}$;
14: **for** each task τ_i in PL **do**
15: $SC_i = rPC_i[0]$;
16: Select the processor and set $st_i = EST_i$ (Eq. 6 and 8), compute $TM_i^{SC_i}$ in Definition 2;
17: **end for**
18: $AM_0 = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
19: Compute SL_{AM_0} (Eq. 7);
20: **if** $SL_{AM_0} > D$ **then**
21: Infeasible problem, algorithm stops.
22: **else**
23: AM relaxation (Algorithm 2);
24: **end if**

configurations (Line 2). For instance, part of the RTE of task τ_5 is depicted in Fig 1. A pruning step removes the task configurations that do not satisfy the reliability constraint (Line 3).

Algorithm 2: Mapping Relaxation Algorithm.

$AM = AM_0$, $SL = SL_0$;
while $|rPC_i| > 1 (\exists \tau_i)$ **do**
  $flag = 0$
  for each task τ_i in $\boldsymbol{N}$ **do**
    $NC_i^A = rPC_i[0]$;
    $NC_i^B = rPC_i[j]$ with $\max \left\{ \sum_{k \in \{o,d\}} (ES^j_{\tau_i^k} / TI^j_{\tau_i^k}) \right\}$;
    if $E^{NC_i^A} \leq E^{NC_i^B}$ **then**
      $NC_i = NC_i^A$;
    else
      $NC_i = NC_i^B$;
    end if
    for each task τ_j in PL **do**
      if $\tau_j \neq \tau_i$ **then**
        Compute $TM_j^{SC_j}$ with configuration SC_j;
      else
        Compute $TM_j^{NC_j}$ with new checked configuration NC_j;
      end if
    end for
    Compute $AM_i = \{TM_i^{NC_i(/SC_i)}, i \in \boldsymbol{N}\}$;
    Compute SL_{AM_i} (Eq. 7) and $SLI_i = SL_{AM_i} - SL$;
    if $SLI_i \neq 0$ **then**
      Compute $Gain_i = \frac{\sum_{k \in \{o,d\}} (ES^{NC_i}_{\tau_i^k} / TI^{NC_i}_{\tau_i^k}))}{SLI_i}$;
    else
      Mark task τ_i as with "highest" gain;
    end if
    Compute $slack_i$ (Eqs. 8, 10, 11);
    if $SL_{AM_i} > D$ or $TI_i^{NC_i} > slack_i$ **then**
      $flag += 1$;
      Discard current task/application mapping when τ_i is considered;
    end if
  end for
  if $flag = N$ **then**
    break
  else
    if tasks with "highest" gain exist **then**
      Select the task with higher energy saving as τ_{rel} to do relaxation;
    else
      Select $\tau_{rel} = \tau_i$ to do relaxation, with i corresponding to task τ_i with $max(Gain_i), i \in \boldsymbol{N}$;
    end if
    Update $SC_{\tau_{rel}} = NC_{\tau_{rel}}$;
    Update $AM = AM_{\tau_{rel}}$ and $SL = SL_{AM_{\tau_{rel}}}$;
    for each configuration pc in $rPC_{\tau_{rel}}$ **do**
      $rPC_{\tau_{rel}} = rPC_{\tau_{rel}}$ - $\{rPC_{\tau_{rel}}$: $E^{pc} \geq E^{SC_{\tau_{rel}}}\}$;
    end for
  end if
end while

i) Reliability Constraint: A task must be executed meeting its reliability requirement, i.e., $R_i \geq R_i^{th}$.

For instance, assuming that the reliability of τ_5 is $R_5^{th} = 0.9994$, the configurations C_5^0, C_5^1 and C_5^2 are pruned in this step. The result is the Feasible Configurations (FC) of the task. FC_i considering only the original task τ_i^o (when $f_i^d = 0$ no duplicated task exists) serve as Baseline Configurations (BC) (Line 4). In our illustration example, the BC of τ_5 are C_5^3, C_5^4 and C_5^5. The next step prunes any feasible configuration with duplicated tasks, if both the energy consumption and the execution time are larger than any defined BC_i (Lines 5-7), e.g., C_5^{10} and C_5^{26} in our illustration example of Fig 1, because they are less efficient than the BC. The result is the Possible Configurations (PC). The PC are ranked based on decreasing energy consumption (rPC_i) (Line 8) where higher energy consumption generally implies that a higher frequency is assigned, and thus, it takes a shorter execution time to execute the task.

3.2 Phase B: Application mapping, under precedence and real-time constraints.

`Phase B` uses `Phase A` task configurations and performs the application mapping, subject to the following constraints:
ii) Precedence constraints: Based on the dependencies defined by the task graph, a task τ_i can start execution only when all its predecessors are completed. Task predecessors also include the duplicated tasks, since our approach is applied offline, and thus, strict scheduling is considered where a task can be executed after all its predecessors are finished. Then, the Earliest Start Time (EST) of τ_i on a processor is given by

$$EST_i = \begin{cases} 0, & if\ \tau_i = \tau_{entry} \\ \max\limits_{\tau_j \in Pred\{\tau_i\}} \{EFT_j\}, & else \end{cases} \tag{5}$$

where $Pred\{\tau_i\}$ is the set of τ_i's predecessors, and $EFT_j = EST_j + et_j$ is the Earliest Finish Time (EFT) of task τ_j.
iii) Deadline constraint: Due to precedence constraints, the actual start time of a task is $st_i \geq EST_i$. Our goal is to exploit the available time slack to save energy, thus, we initially consider that tasks start execution as soon as possible,

$$st_i = EST_i, \forall \tau_i \tag{6}$$

Therefore, the actual finish time of task τ_i is $ft_i = st_i + et_i$. The application must be finished before the deadline D. The Schedule Length of task graph G, under a given application mapping AM, denoted as SL_{AM}, is determined by the actual finish time of exit task τ_{exit}. The deadline constraint is as follows:

$$SL_{AM} = ft_{\tau_{exit}} \leq D. \tag{7}$$

iv) Non-overlapping constraint: Only a single task should be executed on a processor at a given time instance. Taking into account the task dependency and non-overlapping constraints, the EST_i on each processor in Equation 5 is modified as follows:

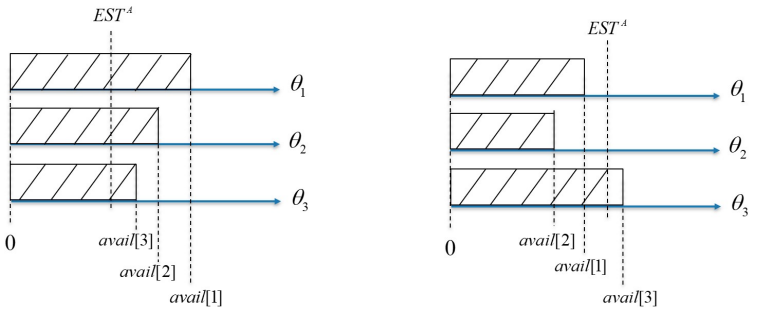
$$EST_i = \begin{cases} 0, if\ \tau_i = \tau_{entry} \\ \max \left\{ \begin{array}{l} \max\limits_{\tau_j \in Pred\{\tau_i\}} \{EFT_j\}, \\ \max\limits_{\tau_p \in Bf\{\tau_i\}} \{EFT_p\}, \end{array} \right\}, else \end{cases} \tag{8}$$

where $Bf\{\tau_i\}$ is the set of tasks that is executed before task τ_i on the same processor as task τ_i. $Bf\{\tau_i\}$ is subject to the task priority assignment explained in detail later in this section.

The method to choose a processor to execute a task is the following. From Eq. 8, except for the entry task, the earliest start time is the maximum of two components. For convenience, we define $EST_i^A = \max_{\tau_j \in Pred\{\tau_i\}}\{EFT_j\}$, due to the task dependency constraint, and $EST_i^B = \max_{\tau_p \in Bf\{\tau_i\}}\{EFT_p\}$ which corresponds to the earliest available time of each processor when it is ready to execute a task, due to non-overlapping constraint. For the currently scheduled task τ_i, EST_i^A has only one value, while EST_i^B may have different values on different processors. Because we target real-time applications, the processor that has the smallest EST_i for task τ_i is chosen to execute this task, i.e., $\{\theta_m : arg\ min_{m \in \boldsymbol{M}}\{EST_i\}\}$ for task τ_i. If there are more than one processors that have the same smallest EST_i for task τ_i, the first processor in the list of available processors (the one with smallest index) is chosen to execute this task.

Fig. 2 illustrates this selection process through an example, where the horizontal axis is related to time. We assume that the currently scheduled task is τ_i (and thus, not the entry task) and the mapping of tasks before τ_i in the Priority List (PL) is done. There are two possible cases: 1) in Fig. 2a, processors θ_1, θ_2 and θ_3 are available later than EST_i^A. The value for EST_i^B can be obtained as $EST_i^B = avail[1]$, if τ_i is executed on processor θ_1, $EST_i^B = avail[2]$, if τ_i is executed on processor θ_2, and $EST_i^B = avail[3]$, if τ_i is executed on processor θ_3. According to Eq. 8, the earliest start times EST_i for task τ_i on the three processors are $avail[1]$ (on θ_1), $avail[2]$ (on θ_2) and $avail[3]$ (on θ_3), respectively. The smallest Earliest Start Time is $EST_i = avail[3]$. Therefore, processor θ_3 is finally chosen to execute task τ_i and sets $st_i = EST_i = avail[3]$; and 2) in Fig. 2b processors θ_1 and θ_2 are available before EST_i^A, and processor θ_3 is available later than EST_i^A. According to Eq. 8, the Earliest Start Time for task τ_i on three processors are $EST_i = EST_i^A$ (on θ_1), $EST_i = EST_i^A$ (on θ_2) and $EST_i = avail[3]$ (on θ_3). Both processors θ_1 and θ_2 achieve the smallest EST_i for task τ_i, which is EST_i^A. In this case, the first processor in the list of available processors, i.e., θ_1, is finally chosen to execute task τ_i and sets $st_i = EST_i = EST_i^A$.

The text paragraphs describe the three steps of `Phase B` (Lines 10-26):



(a) case A (b) case B

Fig. 2: Illustration example of how to select the processor to execute a task.

Step 1 (Lines 10-13): Priorities are given to tasks for task allocation based on the upward rank value [4], $rank_i$, which is common for original and duplicated tasks (Lines 10-12):

$$rank_i = \begin{cases} \overline{et}_i, & if\ \tau_i = \tau_{exit} \\ \overline{et}_i + \max\limits_{\tau_j \in Succ\{\tau_i\}} \{rank_j\}, & else \end{cases} \quad (9)$$

where $\overline{et}_i = (\sum_{l=1}^{L} W_i/f_l)/L$ is the average computation time of τ_i and $Succ\{\tau_i\}$ the immediate successors of τ_i. Then, the Priority List (PL) is ordered in decreasing rank value (Line 13).

Step 2 (Lines 14-21): The initial application mapping AM_0 is generated to check if the problem is feasible and whether time slack exists. For all tasks, AM_0 uses the first configuration in rPC as the Scheduled Configuration SC_i (Line 15). When allocating a task, a processor is chosen as explained previously and sets $st_i = EST_i$ according to Eq. 6 and 8 to execute the current task. We remind that Bf in equation 8 is constructed according to the allocation performed in Line 16 of Algorithm 2. We thus obtain the task mapping ($TM_i^{SC_i}$) per task (Line 16). The set of all task mappings provides the AM_0 (Line 18). Then, its schedule length SL_{AM_0} is obtained (Line 19). If it is higher than the deadline, the problem is infeasible (Lines 20-21), and the algorithm stops.

Step 3: (Lines 22-24) If the initial schedule length is equal or less than the deadline (Line 22), time slack exists. Overall, different task configurations and different tasks can be relaxed to obtain energy savings (Line 23). Algorithm 2 decides the task to be relaxed and its configuration. Initially, the current mapping (schedule length) is initialized with the initial mapping (schedule length). Algorithm 2 is applied iteratively due to the while loop (Lines 2-46), until all tasks reach their last configuration $|rPC_i| = 1 (\forall \tau_i)$ (Line 2) with the least energy consumption, or the flag, which depicts whether no valid relaxation exists, is met. As explained later, when none of the tasks can be selected for relaxation, $flag = N$ and algorithm 2 stops. Firstly, the flag is initialised to zero

(Line 3). In each round of relaxation of the while loop, the algorithm initially enters into a for loop (Lines 4-31), which traverses all tasks for potential relaxation. In each iteration, one task is selected to do a "virtual" relaxation with a New Configuration NC, while the remaining tasks keep the Scheduled Configuration SC (Lines 12-18). Note that, this is not the final decision for relaxation. To decide which New Configuration should be selected for a task (NC_i), we combine two criteria. Let assume the currently selected task to do such a "virtual" relaxation is task τ_i. First, a local search decides a New Configuration NC_i^A (Line 5) by exploring rPC_i sequentially, i.e., selecting always the first configuration. Second, NC_i^B (Line 6) selects the j configuration in rPC_i with the highest value $(ES^j_{\tau_i^o}/TI^j_{\tau_i^o}) + (ES^j_{\tau_i^d}/TI^j_{\tau_i^d})$, where $ES^j_{\tau_i^o}$ ($ES^j_{\tau_i^d}$) is the energy savings and $TI^j_{\tau_i^o}$ ($TI^j_{\tau_i^d}$) is the time increase of task τ_i in configuration j, compared to the current Scheduled Configuration SC_i. The final new configuration NC_i for τ_i is the one with the minimum energy consumption (Lines 7-11). After selecting this new configuration, the relaxation information is updated accordingly, which includes the new application mapping AM_i (Line 19) and its schedule length SL_{AM_i} (Line 20) are obtained. The difference of SL_{AM_i} with the schedule length of the current mapping SL provides the Schedule Length Increase (SLI_i), when task τ_i changes its current configuration SC_i to NC_i (Line 20). With this, the overall gain ($Gain_i$), considering both energy and time, that this configuration modification will bring to the overall mapping, is computed (Lines 21-25). It is possible that in some relaxations the new application schedule length is equal to the schedule length of the current mapping, i.e., $SL_{AM_i} = SL$. In this case $SLI_i = 0$, i.e. relaxation does not increase schedule length. The task is thus marked as with "highest" gain since the $Gain_i$ should be infinite according to equation Line 22 (in this case $Gain_i$ is denoted as Inf in Fig 4.) and this task is selected for relaxation. The time slack $slack_i$ is computed (Line 26). In Line 27 when task τ_i is selected to do relaxation, it is required to compare whether the new SL_{AM_i} exceeds the global deadline. As all the EST values of task τ_i's successors will be accordingly changed due to the additional execution time consumed by task τ_i selecting a new configuration, we then also check whether the time increase TI_i exceeds the available time slack $slack_i$ in order to generate a valid relaxation. In case the new SL_{AM_i} exceeds the global deadline or the TI_i exceeds the available time slack, the flag is incremented (Line 28), and this "virtual" relaxation is considered as invalid, thus, it is discarded (Line 29). This task will not be selected for relaxation in this relaxation round. After we obtain the relaxation information for all tasks iteratively, we check if $flag = N$, which means no valid relaxation exists and none of the tasks can be selected for relaxation, then the algorithm 2 stops (Lines 32-33). This typically happens when the time slack is not enough to enable a relaxation for any task.

The time slack of each task in the current mapping ($slack_i$) is based on task mobility through Eqs. 8, 10, 11 :

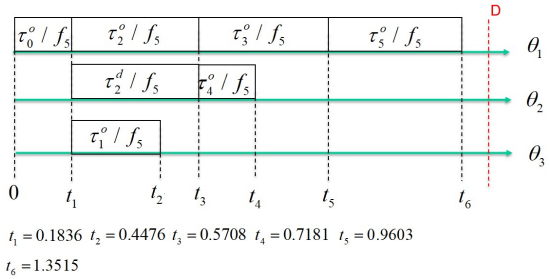
$$slack_i = LFT_i - EST_i - et_i. \tag{10}$$

The Latest Finish Time (LFT) of τ_i is:

$$LFT_i = \begin{cases} D, if\ \tau_i = \tau_{exit} \\ \min \left\{ \begin{matrix} \min\limits_{\tau_p \in Af\{\tau_i\}} \{LST_p\}, \\ \min\limits_{\tau_j \in Succ\{\tau_i\}} \{LST_j\}, \end{matrix} \right\}, else \end{cases} \tag{11}$$

where $LST_i = LFT_i - et_i$ is the Latest Start Time (LST) of task τ_i. $Af\{\tau_i\}$ is the set of tasks executed after task τ_i on the same processor as task τ_i. $Af\{\tau_i\}$ is subject to the task priority in PL. If $flag = N$ does not hold, there exists a valid relaxation, and thus, the global decision takes place (Lines 35-44). First if there exist "highest" tasks, the task with higher energy saving is selected to do relaxation, since it does not increase the schedule length while saves energy consumption (Line 36). Otherwise, the task (and its corresponding configuration) to be relaxed (τ_{rel}) is the task with the maximum overall gain (Line 38), whose time increase in this New Configuration is not larger than its available slack in the current mapping checked in Line 27. Last, the Scheduled Configuration (SC) for the relaxed task, the application mapping and its schedule length are updated (Lines 40-41), and all the configurations that have a higher energy consumption than the selected one are removed from $rPC_{\tau_{rel}}$ (Lines 42-44).

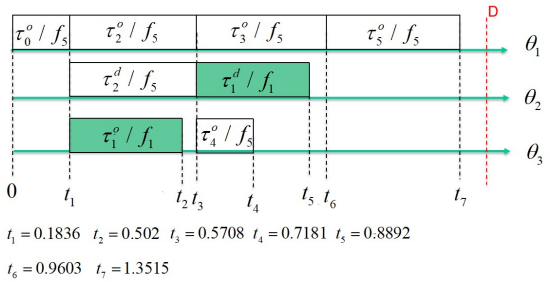
Fig. 3, Fig. 4 and Fig. 5 illustrate the application mapping process for the DAG in Fig 1, when $M = 3$ processors and $D = 1.4$ sec. After obtaining the priority list of the tasks in step 1 (Lines 10-13) in Algorithm 1, Fig 3 shows on the left the initial application mapping AM_0 (according to Lines 14-19 in Algorithm 1) and on the right the initial Scheduled Configuration for each task. The total energy consumption of AM_0 is $E_{AM_0} = 48.12$ mJ and schedule length is $SL_{AM_0} = 1.3515$ sec (Line 19 in Algorithm 1). Since the schedule length is lower than the deadline (Line 22 in Algorithm 1), there exists some time slack, so the relaxation algorithm is applied to save energy consumption (Line 23 in Algorithm 1). During relaxation, task τ_1 is firstly selected, along with the New Configuration where both original and duplication copies are executed with frequency f_1 (Lines 35-40 in Algorithm 2), since this task and its New Configuration provide valid relaxation with the highest gain (Lines 5-30 in Algorithm 2). The energy consumption is reduced without increasing the schedule length (Inf means positive infinity in Fig. 4 and the one with most energy saving is selected if there are multiple Inf cases). The updated application mapping is depicted in Fig. 4. The algorithm continues to apply the relaxation, by selecting the next valid task and the corresponding New Configuration with the highest gain, until the end conditions are met. The final application mapping is depicted in Fig 5, with an energy consumption equal to $E = 41.67$ mJ and a schedule length of $SL = 1.3971$ sec.



*Scheduled configuration

C_i^j	f_i^o (GHz)	f_i^d (GHz)	SLI_i (sec)	$Gain_i$
C_0^0	1.0	-	-	-
C_1^0	1.0	-	-	-
C_2^0	1.0	1.0	-	-
C_3^0	1.0	-	-	-
C_4^0	1.0	-	-	-
C_5^0	1.0	-	-	-

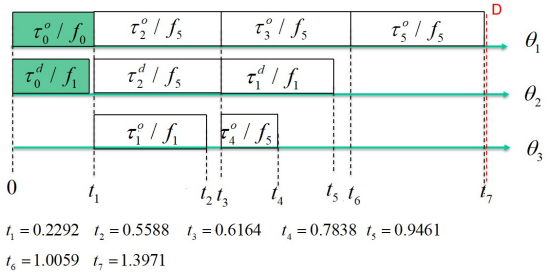
Fig. 3: Illustration example (Fig. 1): Initial application mapping AM_0.



*Scheduled configuration

C_i^j	f_i^o (GHz)	f_i^d (GHz)	SLI_i (sec)	$Gain_i$
C_0^0	1.0	-	0.031	2,246.9
C_1^0	0.8291	0.8291	0	Inf
C_2^0	1.0	1.0	0.066	2,309.2
C_3^0	1.0	-	0.08	858.9
C_4^0	1.0	-	0	Inf
C_5^0	1.0	-	0.07	1,054.3

Fig. 4: Illustration example (Fig. 1): AM_1 with task τ_1 relaxed.



*Scheduled configuration

C_i^j	f_i^o (GHz)	f_i^d (GHz)	SLI_i (sec)	$Gain_i$
C_0^0	0.801	0.8291	0.008	13013.0
C_1^0	0.801	0.801	-	-
C_2^0	1.0	1.0	0.066	2309.2
C_3^0	1.0	-	0.41	168.2
C_4^0	0.8797	-	0.12	472.7
C_5^0	1.0	-	0.066	1054.3

Fig. 5: Illustration example (Fig. 1): Final application mapping.

3.3 Time complexity

In this section, we compute on the dominating time complexity. In the proposed heuristic approach, the maximum number of possible configurations for each task is $L + \binom{2}{L}$ (denoted as K). Phase A takes $O(NK)$ time to obtain the possible configuration space for all tasks. Phase B firstly requires $O(NlogN)$ time to create the priority list (PL). Then, it takes $O(N)$ time to calculate the EST of all tasks in the worst case. For each single task mapping, considering the worst case when selecting a processor to execute the task, all processors needs to be traversed. Therefore, it requires $O(MN)$ time to proceed the task.

Table 3: Processor characteristics

v_l (V)	0.85	0.90	0.95	1.00	1.05	1.1
f_l (GHz)	0.801	0.8291	0.8553	0.8797	0.9027	1.0
C_{eff}	7.3249	8.6126	10.238	12.315	14.998	18.497

The application mapping, including N task mappings, takes $O(MN^2)$ time. In relaxation algorithm 2, we consider the worst case when the deadline is very relaxed and each configuration (K configurations per task) is searched, for all N tasks. The for loop (Lines 4-23) in Algorithm 2 traverses all N tasks and selects one to do a "virtual" relaxation for the application. It thus takes $O(MN^3)$ time. The step to remove configurations for the selected task in Lines 29-31 in Algorithm 2 takes $O(K)$ time considering the worst case, which is not the dominated part of the algorithm. Hence, the total complexity of the relaxation algorithm is $O(MKN^3)$. Finally, overall, the dominating time complexity of the proposed heuristic is $O(MKN^3)$.

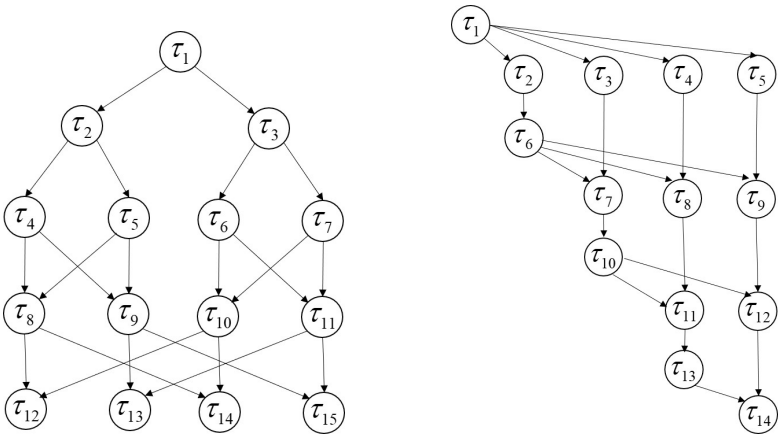
4 Experimental Results

This section evaluates the proposed heuristic (H_RAFTM) with i) the optimal solver using Gurobi 9.0.2 (O_RAFTM) as in [14], and ii) two SoA heuristics a) the Reliability-Aware Mapping (H_RAM), that meets the reliability constraint without task duplication, similar to [5] and ESRG algorithm in [4], and b) the Task Duplication Mapping (H_TDM), applying task duplication for all tasks, similar to [1,4], when the number of replicas is two, or to [19], with 100% task duplication. To evaluate the approaches, the following metrics are used:

1. Feasibility, i.e., the number of experiments where a solution is found out of the total number of experiments (NE).
2. Energy Consumption (EC) of proposed heuristic approach, compared to the energy consumption of the optimal approach O_RAFTM and two SoA heuristic approaches H_RAM and H_TDM.
3. Reliability Improvement (RI) ($RI = R_i - R_i^{th}$), i.e., the task reliability *above* the task reliability threshold.
4. Computation time (CT), i.e., the time required for each approach to find a solution.

Note that an approach may fail to find a solution, especially under strict deadlines. To fairly compare the energy consumption, reliability improvement, and computation time, we present the average values of the experiments where both compared approaches (i.e., O_RAFTM vs H_RAFTM, H_RAFTM vs H_RAM, or H_RAFTM vs H_TDM) were able to find a solution.

The model of the processor is based on a 32-bit RISC-V Instruction Set Architecture (ISA) with a 5-stage pipeline [20]. Regarding DVFS, $L = 6$ voltage/frequency levels are used [21] (Table 3), considering 64 nm technology. The number of processors in the experiments is $M = 2, 4, 6$. The task graphs are



(a) Fast Fourier Transform (FFT) (b) Gaussian Elimination (GE)

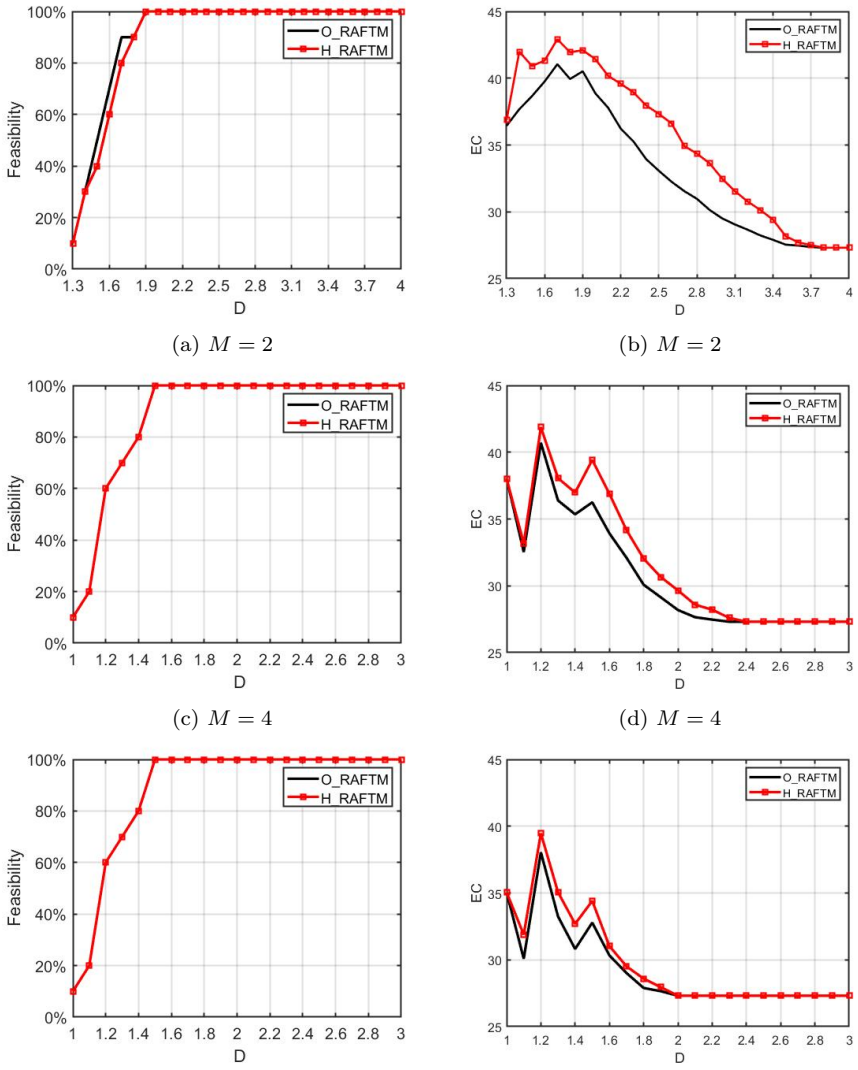
Fig. 6: Real-world application DAG.

obtained both from real-world applications and randomly generated tasks. Regarding real-word applications, two commonly used parallel applications with different shapes of parallelism have been considered, i.e., Fast Fourier Transformation (FFT) ($N = 15$) and Gaussian Elimination (GE) ($N = 14$) [4,5]. Fig 6 depicts the shape of parallelism obtained by FFT and GE DAGs. Random generation is used to compare the optimal approach ($N = 10$) and evaluate the scalability of the heuristics ($N = 100$). To obtain realistic values for the W_i of a task, common benchmarks from the MiBench suite were executed on the processor. The execution cycles and memory accesses are counted and WCEC is computed, assuming that all cores may conflict during memory access. The obtained range is $[1 \times 10^8, 4 \times 10^8]$, and it is used to randomly select the W_i of a task. The reliability threshold R_i^{th} of a task is selected within the range $[0.9990, 0.9995]$, considering a typical magnitude 10^{-3} for reliability target [1]. For each application task graph, several experiments (denoted as NE) are performed, each time with different task characteristics (W_i and R_i^{th}). For each experiment, D is tuned from strict to relaxed deadlines, starting from SL_{AM_0} of the initial application mapping and increased with a step of 0.1 seconds for small randomly generated DAGs and real-world DAGs and 0.5 seconds for large randomly generated DAG.

4.1 Comparison with optimal approach

Fig. 7 and 8 compare the quality of solutions obtained by O_RAFTM and H_RAFTM approaches. We present the results of $NE = 10$ considering random graphs with $N = 10$ tasks, $M = 2$, $M = 4$ and $M = 6$ processors.

Regarding feasibility, when $M = 2$, the H_RAFTM feasibility is very close to the optimal feasibility, as shown in Fig. 7a, except for a few cases when the



(a) $M = 2$ (b) $M = 2$

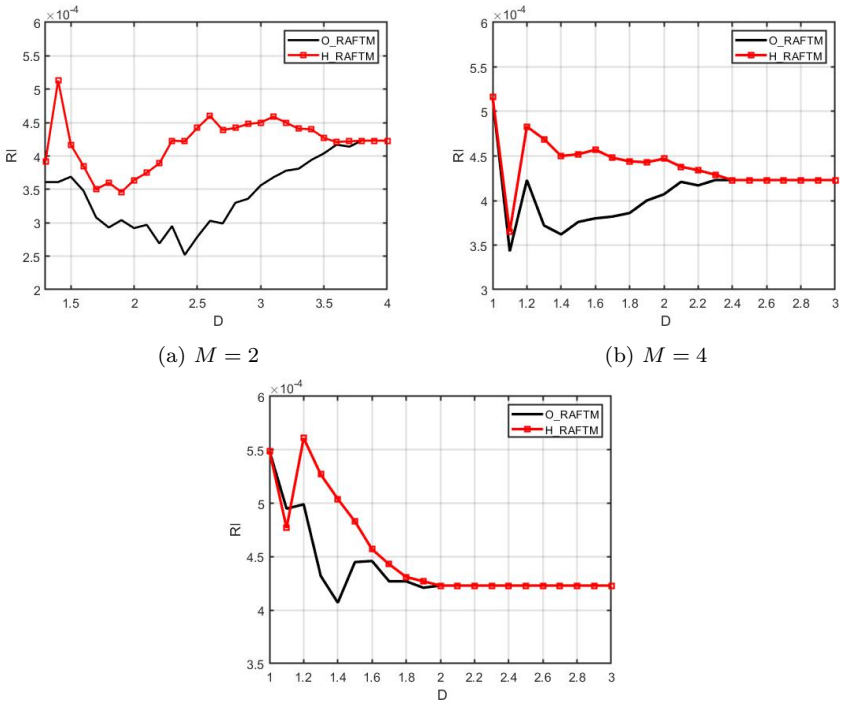
(c) $M = 4$ (d) $M = 4$

(e) $M = 6$ (f) $M = 6$

Fig. 7: Feasibility and Energy Consumption (EC in mJ) of optimal and heuristic approached for $M = 2$, $M = 4$ and $M = 6$ ($N = 10$)

deadline is strict. The average difference before achieving 100% feasibility is 3.3%. With the number of processors increasing to $M = 4$ (Fig. 7c) and $M = 6$ (Fig. 7e), H_RAFTM and O_RAFTM achieve the same feasibility, since more resources are available to execute the original and potentially duplicated tasks.

Regarding energy consumption in Fig. 7b, Fig 7d, and Fig 7f, H_RAFTM generally consumes slightly more energy than O_RAFTM. When the deadline



(a) $M = 2$ (b) $M = 4$

(c) $M = 6$

Fig. 8: Reliability Improvement (RI) of optimal and heuristic approached for $M = 2$, $M = 4$ and $M = 6$ ($N = 10$)

is relaxed, H_RAFTM and O_RAFTM obtain solutions with the same energy consumption. H_RAFTM consumes on average 7.3% ($M = 2$), 4.4% ($M = 4$) and 3.38% ($M = 6$) more energy than the optimal solutions. With the processor number increasing, the energy consumption of both the proposed heuristic and optimal solution flattens at earlier deadlines, since there are more processors resource available to perform the task mapping, and thus, more options to start the tasks earlier.

Regarding reliability improvement, H_RAFTM provides more reliability improvement than optimal solutions at the price of consuming slightly more energy under the same deadlines, as depicted in Fig. 8a, Fig 8b, and Fig 8c.

The average computation time of O_RAFTM and H_RAFTM is computed over the number of experiments to find a feasible solution. Table 4, Table 5 and Table 6 show the results in seconds per deadline D. It can be observed that although few tasks and processors are used, the time to obtain the optimal solution is very long, on average 10^4 more than the proposed H_RAFTM.

Overall, the obtained results show that i) H_RAFTM provides near-optimal solutions, and the solutions tend to converge to the optimal ones with the

number of processors increasing, ii) as expected, H_RAFTM takes significantly less time to obtain the results compared to the optimal approaches.

4.2 Comparison with other heuristics

This section evaluates the behavior of the proposed H_RAFTM heuristic to H_RAM and H_TDM heuristics, considering real-world application DAGs and large random generated DAGs.

4.2.1 Real-word DAG

The quality of solutions obtained by H_RAFTM, H_RAM and H_TDM heuristics for the FFT ($N = 15$) are compared in Fig. 9 and Fig. 11, and for the GE ($N = 14$) in Fig. 10 and Fig. 12, considering $M = 2$, $M = 4$ and $M = 6$ processors, and $NE = 20$ experiments.

The feasibility of the three heuristics for FFT and GE benchmarks is depicted in the left column of Fig. 9 and Fig. 10. Compared to H_TDM, the proposed H_RAFTM can find solutions in significantly more experiments than H_TDM, especially when the deadline is not fully relaxed or the number of cores is reduced. When tasks meet their reliability constraint, H_RAFTM does not need to duplicate these tasks. However, H_TDM duplicates all the tasks, and thus, it can find solutions only when the deadline is relatively relaxed, or several processors exist to run the tasks in parallel. Before obtaining 100% feasibility for both approaches, on average, H_RAFTM finds a solution in more experiments than H_TDM, i.e., 70.2% for FFT and 59.4% for GE ($M = 2$), 47.5% for FFT and 14.5% for GE ($M = 4$) and 47.5% for FFT and 2.2% for GE ($M = 6$). Note that, H_RAFTM and H_RAM have the same feasibility. This behavior is explained as follows: when H_RAM finds a solution, the reliability constraint of all tasks can be met by executing only the original task with a high frequency. In this case, H_RAFTM is also able to find this solution.

The energy consumption obtained by the solutions of the three heuristics for FFT and GE is depicted in the right column of Fig. 9 and Fig. 10. Comparing H_RAFTM and H_RAM, we observe that they consume similar energy

Table 4: Computation time (seconds) for $N = 10$, $M = 2$

D	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**
O_RAFTM	424.8	523.4	537.1	275.5	432.2	739.7	832.7	1,645.6	2,349.0
H_RAFTM	0.15	0.13	0.14	0.17	0.17	0.20	0.22	0.23	0.29
D	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**
O_RAFTM	3,358.1	5,368.3	4,543.9	9,335.6	11,974.2	13,038.4	19,364.5	27,312.9	19,378.6
H_RAFTM	0.32	0.31	0.36	0.33	0.36	0.35	0.41	0.42	0.42
D	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8-4**	
O_RAFTM	13,155.9	21,493.1	24,228.4	77,477.5	5,472.0	4,868.6	4,096.3	2.5	
H_RAFTM	0.49	0.52	0.47	0.52	0.56	0.58	0.59	0.63	

Table 5: Computation time (seconds) for $N = 10$, $M = 4$

D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**
O_RAFTM	14.9	61.0	74.9	138.2	202.1	256.9	327.3	219.0	1039.3	1054.7	126.8
H_RAFTM	0.19	0.26	0.27	0.30	0.35	0.27	0.33	0.43	0.47	0.50	0.57
D	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**	-
O_RAFTM	58.7	123.1	4.9	1.9	1.4	0.6	1.8	1.0	1.8	2.3	-
H_RAFTM	0.59	0.66	0.64	0.70	0.68	0.67	0.68	0.69	0.66	0.78	-

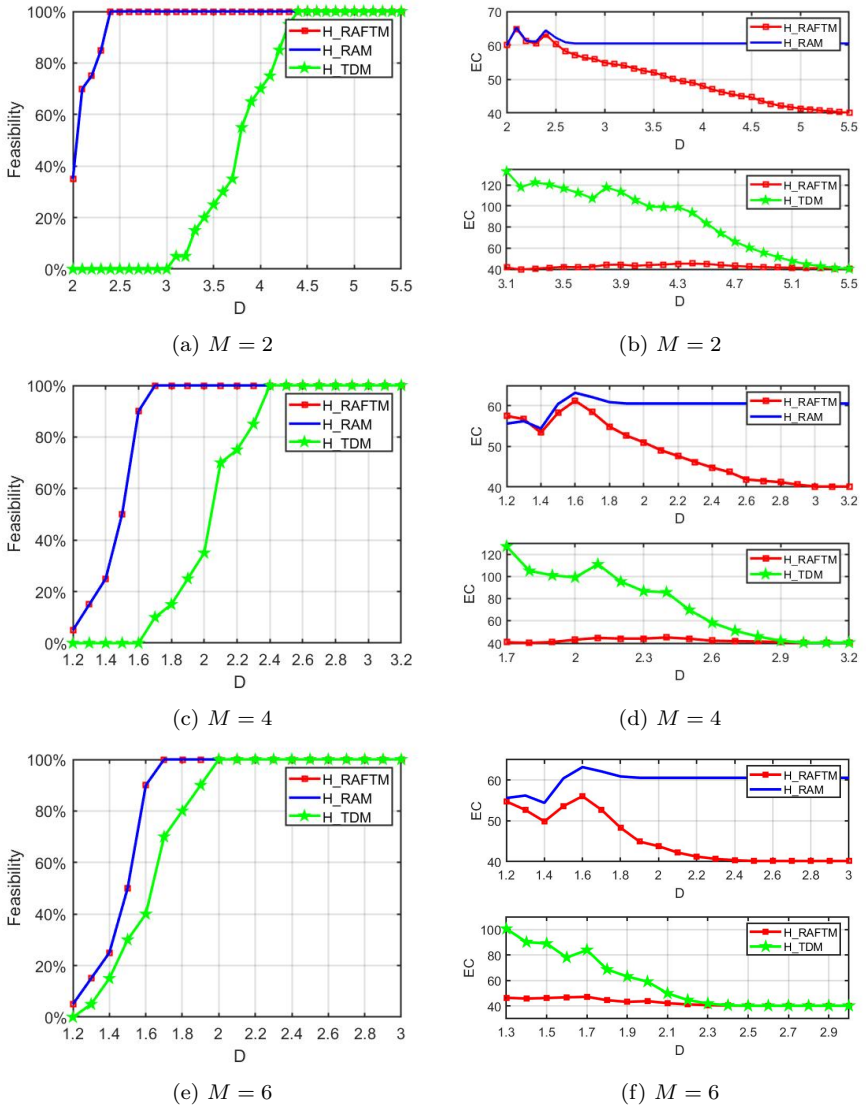
Table 6: Computation time (seconds) for $N = 10$, $M = 6$

D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**
O_RAFTM	0.83	58.10	26.95	89.89	81.87	109.82	58.60	62.77	59.77	14.69	2.49
H_RAFTM	0.32	0.27	0.33	0.48	0.54	0.50	0.58	0.68	0.69	0.70	0.67
D	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**	-
O_RAFTM	0.74	0.65	0.63	0.66	0.66	0.64	0.64	0.63	0.60	0.60	-
H_RAFTM	0.76	0.74	0.76	0.80	0.84	0.77	0.76	0.76	0.71	0.72	-

at very strict deadlines, when the number of processors is small. In this case, H_RAFTM behaves similarly to H_RAM, i.e., mainly executing the original tasks with the frequency required to achieve the reliability constraint. With the deadline relaxing, H_RAFTM starts to consume less energy than H_RAM. H_RAFTM achieves this gain by exploring the available time slack to duplicate tasks to save energy, e.g., up to ∼50.9% for FFT at relaxed deadlines. Similarly, when more processors are available, H_RAFTM can take advantage of these resources and execute duplicated tasks in parallel. Comparing H_RAFTM and H_TDM, as H_TDM applies task duplication for every task, it cannot find solutions under very strict deadlines. H_RAFTM consumes significantly less energy than H_TDM. H_RAFTM selects the task configuration, if it exists, with only the original task, meeting the reliability constraint and consuming less energy than configurations with duplicated tasks. Since H_TDM duplicates all the tasks, its energy consumption can be significant, when it finds a solution. In relaxed deadlines, H_RAFTM behaves similar to H_TDM, i.e., duplicates the tasks when less energy is consumed.

The average reliability improvement obtained by the solutions of the three heuristics is depicted in the left column of Fig. 11 and Fig. 12. H_RAFTM achieves higher reliability than H_RAM, except for very strict deadlines. As explained above, when there is no available time slack to perform duplication, H_RAFTM behaves similarly to H_RAM as most of the tasks are executed with only their original copy. Compared to H_TDM, H_RAFTM provides lower reliability for tight deadlines, as it duplicates only a part of the task set. The same reliability improvement can be achieved in relaxed deadlines, since both H_RAFTM and H_TDM duplicate tasks similarly.

The computation time of H_RAFTM, H_RAM and H_TDM heuristics is depicted in the right column of Fig. 11 and Fig. 12. Overall, when the deadline



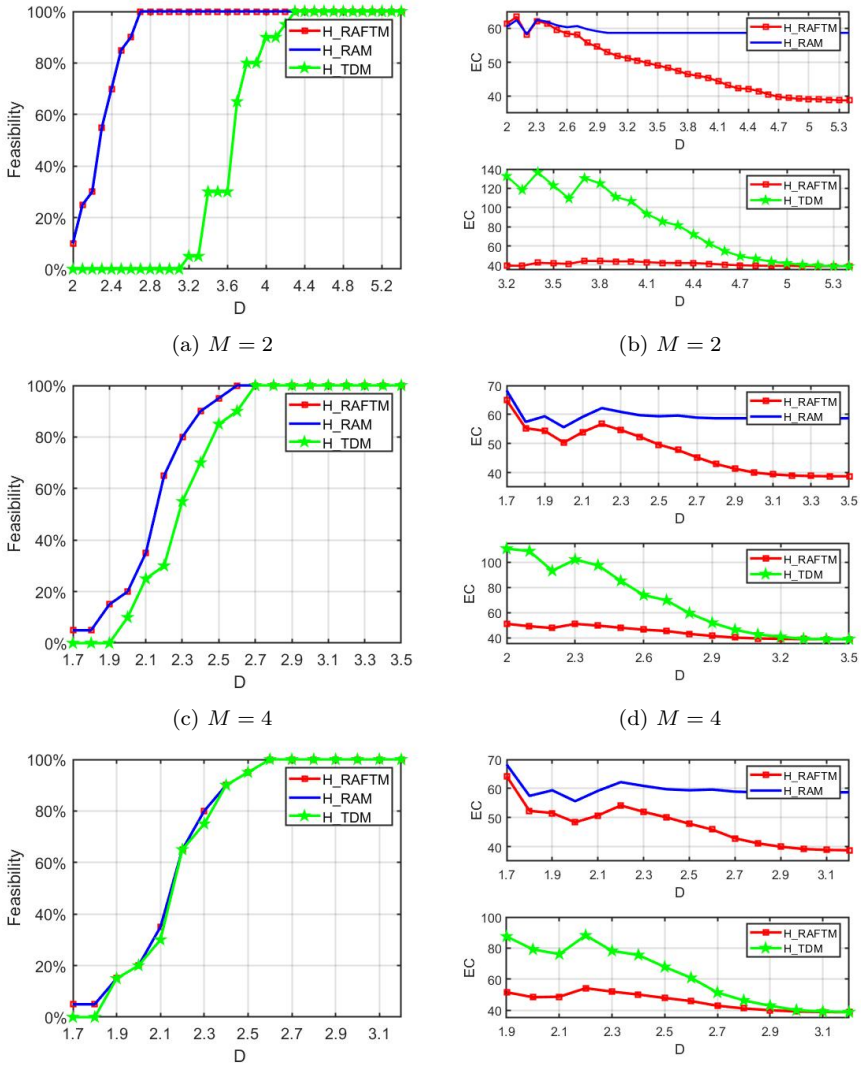
(a) $M = 2$ (b) $M = 2$

(c) $M = 4$ (d) $M = 4$

(e) $M = 6$ (f) $M = 6$

Fig. 9: Feasibility and energy consumption (EC in mJ) of FFT ($N = 15$) for $M = 2$, $M = 4$ and $M = 6$.

increases, the trend of computation time for H_RAM is to remain stable, for H_RAFTM to slightly increase and for H_TDM to increase with a higher factor. The computation time to obtain a feasible solution increases with deadline relaxation because the proposed heuristic explores the PC space for each task, based on the deadline constraints. Therefore, the more relaxed the deadline is, the larger the PC space to be explored per task, and thus, more time is needed.



(a) $M = 2$ (b) $M = 2$

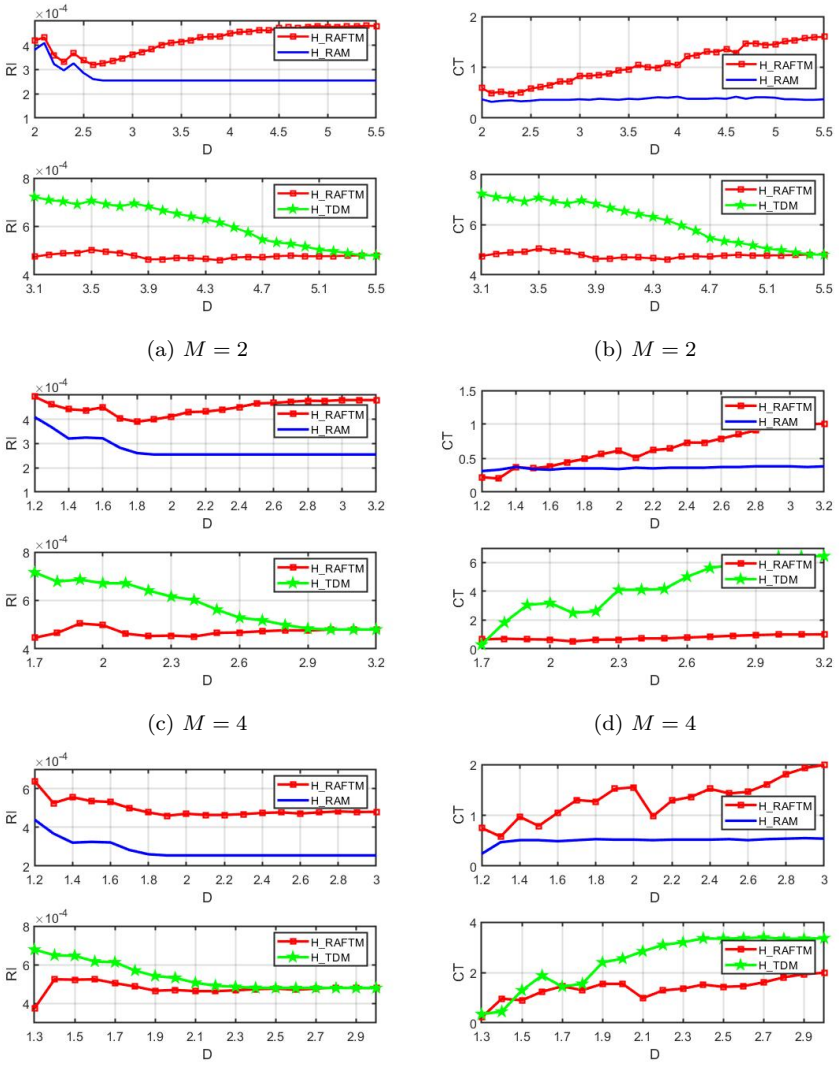
(c) $M = 4$ (d) $M = 4$

(e) $M = 6$ (f) $M = 6$

Fig. 10: Feasibility and Energy Consumption (EC in mJ) of GE ($N = 14$) for $M = 2$, $M = 4$ and $M = 6$.

Note that, H_TDM is the most expensive approach in terms of computation time. This behavior is because all tasks are duplicated, which increases the total number of tasks to be scheduled and the number of PCs in each task PC space, and thus, the time to find a solution. For H_RAM, it only executes original tasks. Therefore, it has a reduced number of PCs in the PC space,

taking the least time to obtain a solution. However, it provides less energy savings as explained above, especially at relaxed deadlines.

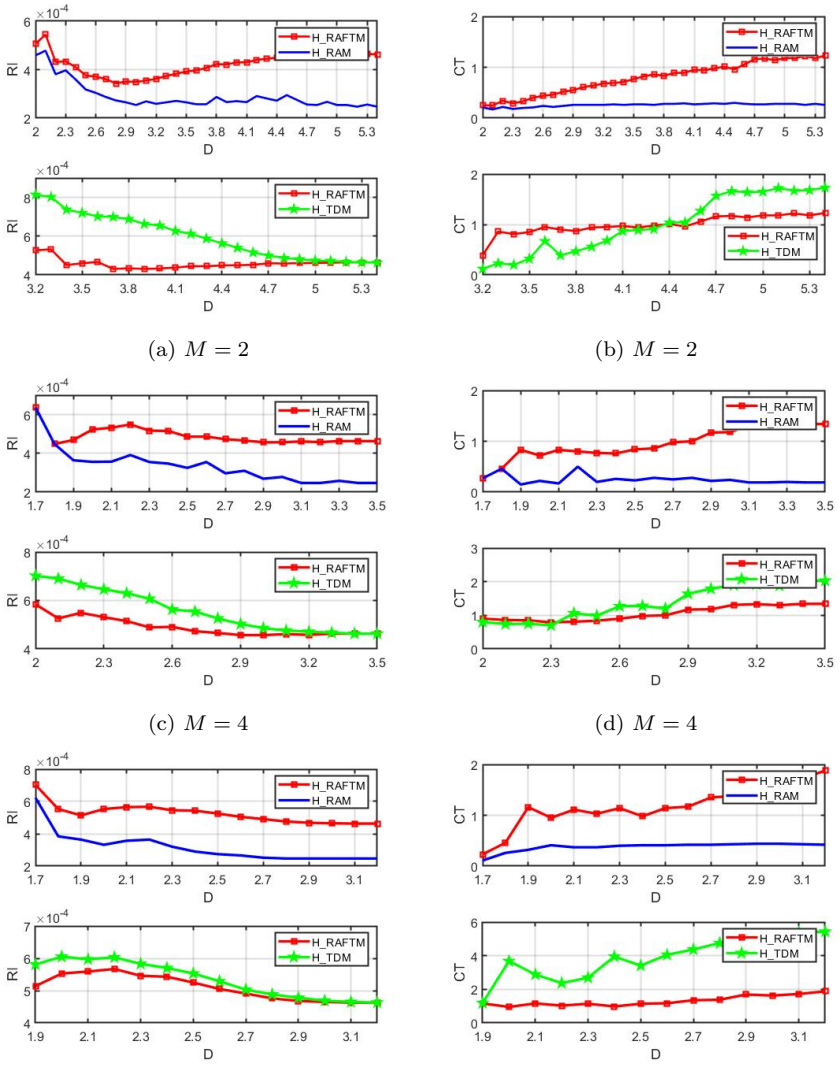


(a) $M = 2$ (b) $M = 2$

(c) $M = 4$ (d) $M = 4$

(e) $M = 6$ (f) $M = 6$

Fig. 11: Reliability improvement (RI) and Computation Time (CT in seconds) of FFT ($N = 15$) for $M = 2$, $M = 4$ and $M = 6$.



(a) $M = 2$ (b) $M = 2$

(c) $M = 4$ (d) $M = 4$

(e) $M = 6$ (f) $M = 6$

Fig. 12: Reliability improvement (RI) and Computation Time (CT in seconds) of GE ($N = 14$) for $M = 2$, $M = 4$ and $M = 6$.

4.2.2 Large random generated DAGs

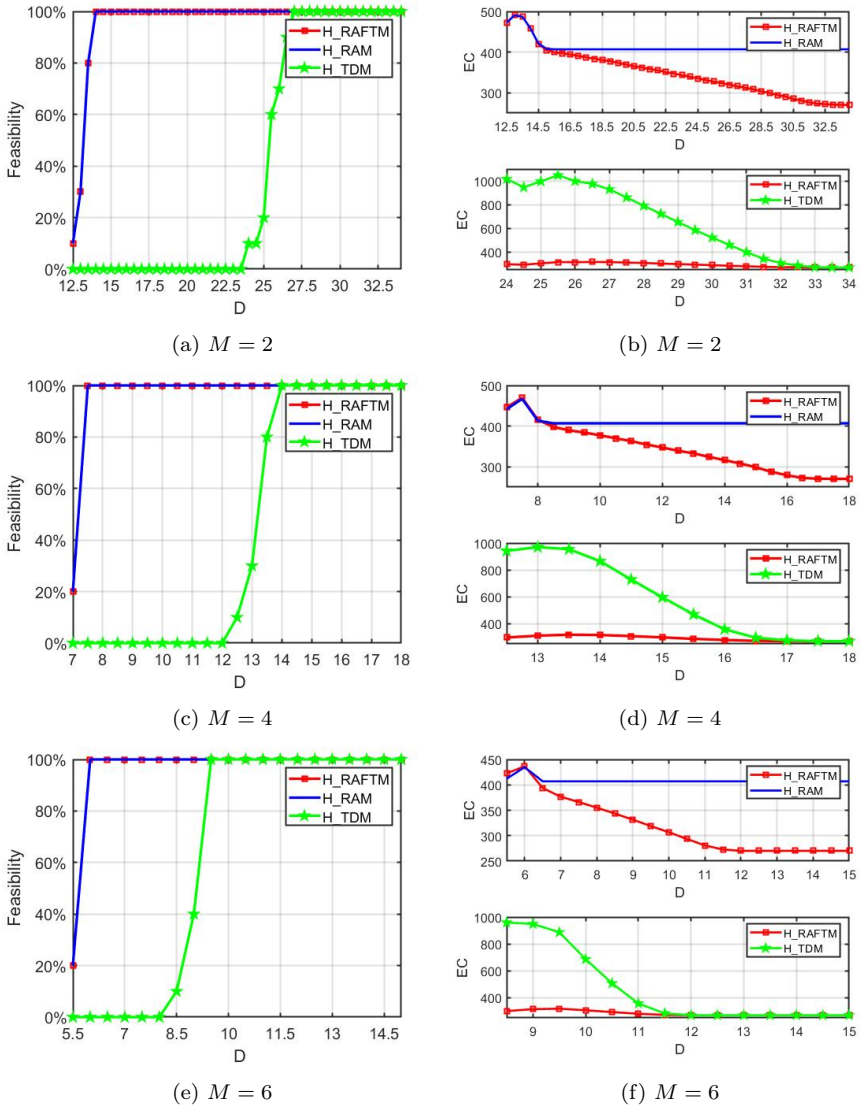
Fig. 13 and Fig. 14 compare the quality of solutions obtained by H_RAFTM, H_RAM and H_TDM heuristics for a large randomly generated task graph with $N = 100$, $M = 2$, $M = 4$ and $M = 6$ processors, and $NE = 10$ experiments. Previous observations regarding feasibility, energy consumption, reliability im-

provement, and computation time for the three heuristics are also verified by the experiments with the large randomly generated DAG. For instance, as the deadline is not fully relaxed or the number of cores is reduced, H_RAFTM has increased feasibility compared to H_TDM, i.e., 84.8% for $M = 2$, 85.7% for $M = 4$ and 83.8% for $M = 6$. Moreover, as the number of processors increases, energy consumption is reduced as lower frequencies can be selected, and partial task duplication can be applied by H_RAFTM, as long as the reliability constraint is met.

Regarding computation time, it is increased when the number of tasks increases as expected, but still remains low compared to the prohibited computation time required for the optimal approach. For a small randomly generated task with $N = 10$ (Table 5 when $M = 4$ and Table 6 when $M = 6$), the proposed heuristic takes less than 1 sec to find a solution for all experiments whereas for a large randomly generated task with $N = 100$, the average computation time (considering all experiments and deadlines) is 144 sec ($M = 2$), 159 sec ($M = 4$) and 212 sec ($M = 6$). Comparing the computation time of the three heuristics for large DAGs $N = 100$, the average computation time for H_RAM is 52 sec ($M = 2$), 66 sec ($M = 4$) and 75 sec ($M = 6$). For H_TDM, the average computation time is 488 ($M = 2$), 501 sec ($M = 4$) and 718 sec ($M = 6$). Overall, we observe that H_TDM is the more time-consuming approach, and H_RAM is the least time-consuming approach. However, H_TDM cannot always find solutions, whereas H_RAFTM finds solutions with the same or less energy consumption than H_RAM and H_TDM.

5 Conclusion

A heuristic algorithm is proposed to obtain the task mapping of parallel applications, under real-time, reliability, and task dependency constraints. The goal is to minimize total energy consumption by deciding frequency assignment, task allocation, task scheduling, and task duplication. Based on the experimental results from real-world applications and randomly generated task graphs, the proposed heuristic achieves near-optimal solutions, with low computation time. Compared to similar heuristics, it achieves better energy consumption and is able to find solutions even when the other approaches fail. In future work, we will propose similar heuristics for other DVFS schemes provided by platforms, such as assigning a common frequency at cluster, processor, and platform levels.

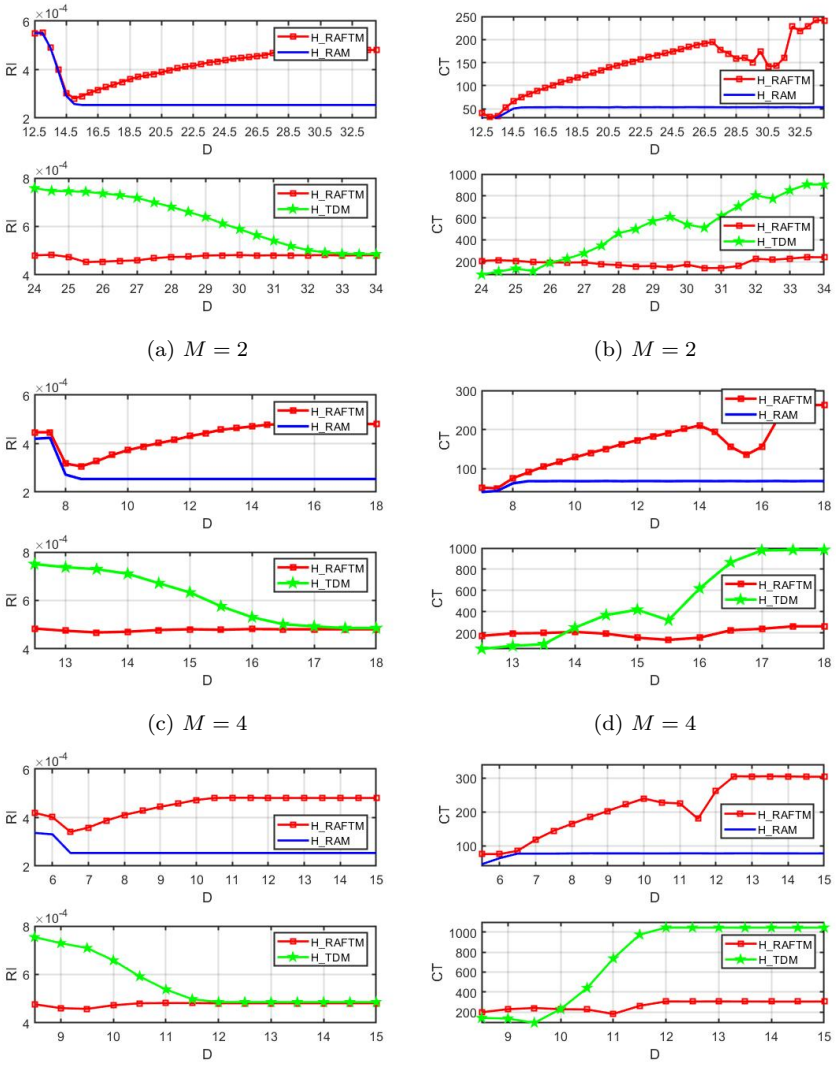


(a) $M = 2$ (b) $M = 2$

(c) $M = 4$ (d) $M = 4$

(e) $M = 6$ (f) $M = 6$

Fig. 13: Feasibility and Energy Consumption (EC in mJ) of large randomly generated DAG ($N = 100$) for $M = 2$, $M = 4$ and $M = 6$.



(a) $M = 2$ (b) $M = 2$

(c) $M = 4$ (d) $M = 4$

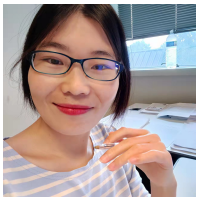
(e) $M = 6$ (f) $M = 6$

Fig. 14: Reliability Improvement (RI) and Computation Time (CT in seconds) of large randomly generated DAG ($N = 100$) for $M = 2$, $M = 4$ and $M = 6$.

References

1. M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Trans. Parallel and Distributed Systems*, vol. 28, no. 3, pp. 813–825, 2017.
2. K. Huang, X. Jiang, X. Zhang *et al.*, "Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems," *IEEE Access*, vol. 6, pp. 57 614–57 630, 2018.
3. C. Gou, A. Benoit, M. Chen *et al.*, "Reliability-aware energy optimization for throughput-constrained applications on MPSoC," in *IEEE International Conference on Parallel and Distributed Systems*, 2018, pp. 1–10.
4. G. Xie, Y. Chen, X. Xiao *et al.*, "Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Sustainable Computing*, vol. 3, no. 3, pp. 167–181, 2018.
5. G. Xie, Y. Chen, Y. Liu *et al.*, "Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems," *IEEE Trans. Industrial Informatics*, vol. 13, no. 4, pp. 1629–1640, 2017.
6. Z. Deng, D. Cao, H. Shen *et al.*, "Reliability-aware task scheduling for energy efciency on heterogeneous multiprocessor systems," *The Journal of Super computing*, 2021.
7. J.Roeder, B. Rouxel, S. Altmeyer *et al.*, "Energy-aware scheduling of multi-version tasks on heterogeneous real-time systems," in *Proceeding of the 35rh Annual ACM Symposium on Applied Computing*, 2020, pp. 501–510.
8. L. Zhang, K. Li, K. Li *et al.*, "Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems," *Int. Journal of Electrical Power Energy Systems*, vol. 78, pp. 499–512, 2016.
9. B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Trans. Design Automation of Electronic Systems*, vol. 18, no. 2, pp. 1–21, 2013.
10. Y. Guo, D. Zhu, and H. Aydin, "Reliability-aware power management for parallel real-time applications with precedence constraints," in *IEEE Int. Green Computing Conf.*, 2011, pp. 1–8.
11. M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-phase low-energy n-modular redundancy for hard real-time multi-core systems," *IEEE Trans. Distributed Systems*, vol. 27, no. 5, pp. 1497–1510, 2016.
12. M. Salehi, M. K. Tavana, S. Rehman *et al.*, "DRVS: Power-efficient reliability management through dynamic redundancy and voltage scaling under variations," in *IEEE/ACM Int. Symp. Low Power Electronics and Design*, 2015, pp. 225–230.
13. M. Cui, L. Mo, A. Kritikakou, and E. Casseau, "Energy-aware partial-duplication task mapping under real-time and reliability constraints," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, vol. 12471. Springer, 2020.
14. M. Cui, A. Kritikakou, L. Mo, and E. Casseau, "Fault-tolerant mapping of real-time parallel applications under multiple DVFS schemes," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 387–399.
15. M. Cui, A. Kritikakou, L. Mo, and E. Casseau, "Energy-efficient partial-duplication task mapping under multiple dvfs schemes," *International Journal of Parallel Programming(IJPP), Springer*, vol. 50, p. 267–294, 2022.
16. J. Zhou, J. Sun, X. Zhou *et al.*, "Resource management for improving soft-error and lifetime reliability of real-time MPSoCs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2215–2228, 2019.
17. D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," *IEEE/ACM International Conference on Computer Aided Design*, pp. 35–40, 2004.
18. K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, p. 94–125, 2004.
19. S. Tosun, "Energy- and reliability-aware task scheduling onto heterogeneous MPSoC architectures," *The Journal of Supercomputing*, vol. 62, p. 265–289, 2012.

20. S. Rokicki, D. Pala, J. Paturel *et al.*, "What you simulate is what you synthesize: Designing a processor core from c++ specifications," in *IEEE/ACM Int. Conf. Computer-Aided Design*, 2019.
21. G. Quan and V. Chaturvedi, "Feasibility analysis for temperature-constraint hard real-time periodic tasks," *IEEE Trans. on Industrial Informatics*, vol. 6, no. 3, pp. 329–339, 2010.



Minyu Cui received the B.E. degree in optical information science and technology and the M.S. degree in electronics and communication engineering, in Shandong University in 2013 and 2018 respectively, and the Ph.D. degree at École Normale Supérieure de Rennes under the supervision of Prof. Emmanuel Casseau and prof. Angeliki Kritikakou at IRISA/Inria Rennes in 2022. Her research interest includes algorithm design, resource optimization, task mapping, multicore architecture, fault-tolerance, computing systems, real-time systems.

Angeliki Kritikakou is currently an Associate Professor at University of Rennes 1 and IRISA - INRIA Rennes research center. She received her Ph.D. in 2013 from the Department of Electrical and Computer Engineering at University of Patras, Greece and in collaboration with IMEC Research Center, Belgium. She worked for one year as a Postdoctoral Research Fellow at the Department of Modelling and Information Processing (DTIM) at ONERA in collaboration with Laboratory of Analysis and Architecture of Systems (LAAS) and the University of Toulouse, France. Her research interests include embedded systems, real-time systems, mixed-critical systems, hardware/software co-design, mapping methodologies, design space exploration methodologies, memory management methodologies, low power design and fault tolerance.

Lei Mo is currently an associate professor with the School of Automation, Southeast University, Nanjing, China. He received the B.S. degree from College of Telecom Engineering and Information Engineering, Lanzhou University of Technology, Lanzhou, China, in 2007, and the Ph.D. degree from College of Automation Science and Engineering, South China University of Technology, Guangzhou, China, in 2013. From 2013 to 2015, he was a research fellow with the Department of Control Science and Engineering, Zhejiang University, China. From 2015 to 2017, he was a research fellow with INRIA Nancy-Grand Est, France. From 2017 to 2019, he was a research fellow with INRIA Rennes-Bretagne Atlantique, France. His current research interests include networked estimation and control in wireless sensor and actuator networks, cyber-physical

systems, task mapping and resources allocation in embedded systems. He serves as an Associate Editor for KSII Transactions on Internet and Information Systems, International Journal of Ad Hoc and Ubiquitous Computing, Journal of Computer and Journal of Electrical and Electronic Engineering. He also serves as a Guest Editor for IEEE Access and Journal of Computer Networks and Communications and a TPC Member for several international conferences.

Emmmanuel Casseau is a Professor at ENSSAT Graduate Engineering School, University of Rennes 1, Lannion, France. He is a member of the IRISA Lab./INRIA (French Institute for Research in Computer Science and Automation), France. His research interests are in the broader area of computer architecture, where he investigates the design of high-performance and cost-effective embedded systems and reconfigurable-based architecture design. Within this context, he has performed research in high-level synthesis, mapping/scheduling techniques, custom and application-specific hardware architectures targeting multimedia and signal processing applications, reconfigurable architectures and FPGA-based accelerators.