Energy Efficient Mapping of Mixed Criticality Applications on Unrelated Heterogeneous Multicore Platforms

Muhammad Ali Awan, Damien Masson, Eduardo Tovar

Energy Efficient Mapping of Mixed Criticality Applications on Unrelated Heterogeneous Multicore Platforms

Muhammad Ali Awan
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: muaan@isep.ipp.pt

Damien Masson
Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE Paris, UPEM, F-93162, Noisy-le-Grand, France
Email: damien.masson@esiee.fr

Eduardo Tovar
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: emt@isep.ipp.pt

***Abstract*—Heterogeneous multicore platforms are becoming an attractive choice to deploy mixed criticality systems demanding diverse computational requirements. One of the major challenges is to efficiently harness the computational power of these multicore platforms while deploying mixed criticality applications with timeliness properties. Energy efficiency is also one of the desired requirements in the design phase, and therefore it is often difficult for the system designer to simultaneously satisfy those sometimes contradictory requirements. In this paper, we propose a novel partitioning algorithm for unrelated heterogeneous multicore platforms to map mixed criticality applications. The algorithm not only ensures the timeliness in different modes of execution but also tries to allocate the applications to their energy-wise favourite cores. We considered a realistic power model that further increases the relevance of the proposed approach. We have performed an extensive set of experiments to evaluate the performance of the proposed approach, and we show that in the best-case, we achieve a 23.8% gain in the average power dissipation over the state-of-the-art partitioned algorithm. Our proposed algorithm also has a better weighted schedulability when compared to the existing partitioned algorithms.**

1. Introduction

Modern real-time (RT) applications are becoming increasingly complex everyday. In particular, these applications may have different criticality levels. A criticality level corresponds to a level of assurance against failure [1]. The highest-criticality functions are vital for the operations of the system and any violation of temporal constraint (fault) may lead to disastrous consequences. However, fault of lower criticality functions are tolerable and may cause a decrease in the quality of service.

In the past, applications of different criticality were hosted on separate components. A recent trend in RT and embedded system domain to integrate the functionalities belonging to different criticality levels on a single hardware platform has paved a way towards mixed criticality systems (MCS). Mixed criticality systems are common in many application domains such as avionics and automotive. For example, an unmanned aerial vehicle may have functionalities corresponding to flight critical (concerning the safety of the flight) and mission critical (reconnaissance and surveillance) operations [1]. It is vital for the integrity of the aerial vehicle to ensure the safe operation of the flight in all conditions, while reconnaissance and surveillance are secondary functionalities.

In order to increase the level of assurance against failure for the high criticality functions, pessimistic estimates of the worst-case-execution time (WCET) are determined using static analysis. However, a mixed criticality system designed with pessimistic estimates of WCET for the high criticality functions (tasks) will lead to an inefficient utilisation of the resources and therefore to an unnecessary over-provisioning of the system. This is driven by the fact that high criticality WCET estimates rarely occur during execution. To efficiently exploit the available resources Steve Vestal [2] proposed a mixed criticality system model. In that model, tasks have different estimates of WCET corresponding to different criticality levels. The system starts in a low criticality mode and its schedulability is ensured assuming a WCET estimate corresponding to current low criticality mode. If any task exceeds its execution budget beyond the WCET estimate defined for the given criticality mode of operation, the system transitions to the high criticality mode.

Modern multicore platforms provide sufficient computing capabilities to deploy complex applications on a single chip [3], [4]. Among different kinds of multicore architectures, heterogeneous multicore platforms – composed of more than one heterogeneous processing unit (core) – has gained popularity due to diverse computing capabilities. The cores with different characteristics on such platforms are designed to perform specific functions efficiently with minimal energy consumption. This flexibility allows a system designer to map applications of different types with minimal resources and reduce the overall cost. Various components (applications) in a mixed criticality system have different computing requirements, which makes heterogeneous multicore platforms an attractive choice to opt for. However, efficient mapping of applications with different characteristics on such a diverse computing platform with an objective of maximising the resource usage is a non trivial exercise. It leads to a multi objective optimisation problem. Apart from exploiting the computational resources, energy efficiency has become an important resource to optimise due to several reasons such as thermal issues, weight, size, battery life, etc. Therefore, it is not only important to feasibly map the applications onto the processors but also to try to minimise the overall energy consumption. on the given multicore platform.

The energy consumption of an heterogeneous multicore platform can be reduced in two different ways [5]. Firstly, a task should be mapped to a processor where it dissipates minimum dynamic power (favourite core). Secondly, energy-aware scheduling mechanisms can be employed to further reduce the energy consumption on each core. We propose a novel task-to-core mapping algorithm that can be combined with other energy-aware scheduling algorithm. In our work, a major concern has been the provision of a realistic power model. In reality, the energy consumption of an application is not merely a function of the execution time but also depends on the set of instructions that it executes to perform the desired functionality on the given core. These instructions may use different parts of the core and exhibit different energy consumption characteristics. In this work, we consider a more realistic power model and use partitioned scheduling to map the given set of tasks belonging to different criticality levels on an heterogeneous multicore platform such that the available computational capacity of the hardware platform is efficiently exploited and the energy consumption is minimised while ensuring the timeliness requirements of the system. This is an NP-hard problem in a strong sense as it is a special case of bin packing and hence, heuristics are the way forward.

This work is an extension of a previously proposed Improved Least Loss Energy Density (ILLED) allocation algorithm [6]. The main contribution of this paper is to extend it to mixed criticality systems.

The rest of the paper is organised as follow. Section 2 presents related work followed by the system model in Section 3. Our proposed novel approach is detailed in Section 4 as well as the original ILLED algorithm, and an evaluation of this approach is presented in Section 5. We conclude and present future work directions in Section 6.

2. Related work

There are three types of contributions in the state-of-the-art related to our work on mixed criticality systems. The first category of contributions performs task-to-core allocation with an objective of optimising the system utilisation. The second category of contributions performs the energy-aware task-to-core allocations. In the last category, the energy consumption of the individual core is optimised based on existing energy saving algorithms such as shutdown or frequency scaling. The three next paragraphs will present these state-of-the-art contributions.

There has been some efforts in the state-of-the-art to propose new techniques and evaluate the potential of existing bin packing heuristics to perform the allocation with the objective of optimising the utilisation of the system. Lakshmanan et al. [7] proposed an allocation heuristic known as compress-on-overload packer for the distributed mixed criticality systems with an objective of minimising the deadline misses and the number of processors needed by the system. They devised a metric called ductility to characterise the overload behaviour of the mixed-criticality systems. Tamas-Selicean and Pop [8] consider applications at different safety-integrity levels (SIL) and propose a tabu search-based approach to perform the task-to-core/partition mapping on a platform composed of heterogeneous processing elements interconnected by a broadcast bus. The tasks and messages are scheduled using static cyclic scheduling. The SIL level of an application is incremented to utilise the slack of higher SIL application. Though it increases the development cost, it avoids the hardware upgrades. Kelly et al. [9] analysed the mixed criticality task-set allocation and priority assignment problems assuming a fixed priority preemptive scheduling on a homogeneous multicore platform. They considered first-fit, best-fit and worst-fit heuristics in combination with decreasing criticality and decreasing utilisation. The rate-monotonic and Audsley's priority assignment algorithms are used to allocate priorities. It is concluded that decreasing criticality and Audsley's priority assignment algorithm perform better than other variants. Paul Rodriguez et al. [10] also compares different heuristics in the context of mixed criticality systems on identical processors with an objective of optimising the utilisation. They considered several heuristics such as best-fit, worst-fit, first-fit and task ordering criteria of utilisation, period, deadline and density both in decreasing and increasing order. Many allocations heuristics are proposed based on the combination of above mentioned criterions. They also proposed a pruning mechanism to select the best allocation heuristics.

The state-of-the-art on the energy minimisation of mixed criticality systems is very limited and very few results exists. Broekaet et al. [11] studied the integration of reliable power-magement policies into a real-time operating system such as SYSGO PikeOS. They also proposed an outline of a system level power management approach that allocates energy budgets to virtual machines. In their proposal, an overrunning critical virtual machine deducts its additional energy budget from the energy budget of low priority virtual machine to be scheduled in future. If the low criticality virtual machine surpasses its allocated energy budget, the CPU is either put to sleep mode or the frequency of the processor is reduced to save the energy consumption. Finally, for the battery powered devices, in case of low-level battery signal event, power budget of less critical virtual machines can be scaled down in favour of more critical virtual machines. This work is different from our proposed algorithm as it mostly considers the integration of system level power management algorithms while we consider the energy-aware task-to-core allocation. Legout et al. [12] proposed a technique that ensures the schedulability of high criticality tasks and provides a trade-off between the energy reduction and the number of deadline misses of low criticality tasks on an identical multicore platform. This approach also relies on the dynamic slack [13] to schedule the low criticality tasks which are allocated beyond the provisioned execution time. It is assumed that low power states on different cores can be achieved independently. A linear programming approach is used to map the tasks in the hyper-period which is divided into small intervals. Their approach is different from our proposed approach as it does not consider the sporadic task-model and unrelated heterogeneous multicore platforms.

Zhang et al. [14] allocate energy budgets to different cores and perform allocation through a genetic algorithm to extend the battery life time of the system. The authors assume however a naive power model where the energy consumption of a task is only a function of its execution time. Huang et al. [15] integrated the dynamic voltage and

frequency scaling (DVFS) with mixed criticality scheduling algorithm EDF-VD. The available slack is utilised to reduce the energy consumption in normal mode of operation and the frequency is scaled up to meet the deadlines of high criticality task in case of an overrun. This approach can be used in combination with our approach. Marcus Volp et al. [16] discussed why energy constraint may become critical in certain scenarios. He advocates deadline misses of low criticality tasks in favour of executing the high critical tasks when system is short of energy supply. Recently, Narayana et al. [17] proposed a DVFS approach for the mixed criticality systems on a unicore platform and further reduce the energy consumption by providing an energy-aware task-to-core mapping on identical multicore processor. Unlike their approach, this paper considers the heterogeneous multicore platform, however, their DVFS approach can be used in combination with our work.

3. System model

This work considers a partitioned unrelated heterogeneous multicore platform $\pi \stackrel{\text{def}}{=} [\pi^1, \pi^2, \ldots, \pi^M]$ composed of M distinct cores. Cores on an unrelated heterogeneous multicore platform have no relation among each other. They have the highest degree of heterogeneity. Usually, different cores have different instruction set architecture. The energy consumption and the WCET of a task vary substantially on these different cores. For example, a task τ_a may have a WCET of 2 and 5 time units on core π^i and core π^j, respectively. It is equally possible that another task τ_b may have WCET of 10 and 1 time units on core π^i and core π^j, respectively.

We adopt a common mixed criticality task model, initially proposed by Steve Vestal [2] and later extended by other researchers in the RT research community [18]. We consider a dual criticality system in which a system is defined to execute in either low criticality mode (L-mode) or high criticality mode (H-mode) of operation. Each task (defined below) belongs to either low or high criticality level (safety assurance level). Each task of high criticality level has two different estimates of WCET: its WCET estimate for the L-mode of operation is considered safe but lack proofs, and its WCET estimate for H-mode of operation is provable safe and possibility much higher than the L-mode WCET. We assume a set of n independent sporadic tasks $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task $\tau_i \stackrel{\text{def}}{=} \langle T_i, D_i, \kappa_i, \vec{C_i}(H), \vec{C_i}(L), \vec{E_i} \rangle$ is characterised by its minimum inter-arrival time T_i, relative deadline D_i, criticality level κ_i, a vector of WCET profile $\vec{C_i}(H)$ in H-mode of operation, a vector of WCET profile $\vec{C_i}(L)$ in L-mode of operation and, finally, a vector of energy consumption profile $\vec{E_i}$. , i.e., $\kappa_i \in \{L, H\}$. The WCET profiles $\vec{C_i}(H) \stackrel{\text{def}}{=} (C_i^1(H), C_i^2(H), \ldots, C_i^M(H))$ and $\vec{C_i}(L) \stackrel{\text{def}}{=} (C_i^1(L), C_m^2(L), \ldots, C_i^M(L))$ are the WCET estimates in the H-mode and L-mode of operation, respectively, on different cores indexed from 1 to M. Similarly, energy consumption profile $\vec{E_i} \stackrel{\text{def}}{=} (E_i^1, E_m^2, \ldots, E_i^M)$ represents the energy consumption of task τ_i on different cores in L-mode.

It is very common to assume in state-of-the-art that the energy consumption of a task is a function of its execution time. However, in reality, the energy consumption on a certain processor depends also on the set of instructions it has to execute to perform its desired functionality. Various instructions use different parts of a core, and hence may result in a different estimate of energy consumption. Therefore, two different applications with similar execution time may result in different energy consumption depending on the characteristics of the instructions used, and on the number of cache misses involved. In this work, we consider a more realistic power model in which the energy consumption of a task depends on the characteristics of the core type and hence, computed on each individual core type using a well known energy measurement technique [19]. That technique, proposed by Snowdon et al. [19], has the ability to incorporate the effect of other system resource usage, such as memory subsystem and caches etc, on the energy consumption.

We assume that $\tau(L) \stackrel{\text{def}}{=} \{\tau_i \in \tau | \kappa_i \in L\}$ and $\tau(H) \stackrel{\text{def}}{=} \{\tau_i \in \tau | \kappa_i \in H\}$ represent the subsets of L-tasks and H-tasks in τ, respectively. It is assumed that $\forall \tau_i \in \tau(H) \wedge \forall m \in [1, 2, \ldots, M]$, $C_i^m(L) \leq C_i^m(H)$ and $\forall \tau_i \in \tau(L) \wedge \forall m \in [1, 2, \ldots, M]$, $C_i^m(H) = 0$. Tasks are not allowed to migrate after assignment as we assume partitioned scheduling. The schedulability analysis proposed by Ekberg and Yi [20] is used to test the feasibility of the system in both low and high criticality modes. This analysis shortens the deadlines of $\tau(H)$ in the L-mode to keep the schedule ahead of time, which allows a safe transition from L to H-mode in an event of overrun or other activity that causes a mode transition. They allow having independent scaling factors for different H-tasks and uses a more precise demand bound function based schedulability test. This analysis shows substantial gains over the state-of-the approaches. The utilisation of task τ_i in low and high criticality modes on processor π^m are represented as $U_i^m(L) \stackrel{\text{def}}{=} \frac{C_i^m(L)}{T_i}$ and $U_i^m(H) \stackrel{\text{def}}{=} \frac{C_i^m(H)}{T_i}$, respectively. A subset of the tasks assigned to a core π^m is represented as $\tau(\pi^m)$. The overall utilisation of tasks assigned to core π^m in L and H-modes is defined as $U^m(L) \stackrel{\text{def}}{=} \sum_{\forall \tau_i \in \tau(\pi^m)} C_i^m(L)$ and $U^m(H) \stackrel{\text{def}}{=} \sum_{\forall \tau_i \in \tau(\pi^m)} C_i^m(H)$, respectively. We assume a constrained deadline model in our system in which a task relative deadline is smaller than or equal to its minimum inter-arrival time, i.e., $D_i \leq T_i$.

4. Energy-aware allocation heuristic

We divide this section into two parts. Initially, we define some concepts needed, and then, in Section 4.2, the proposed heuristic is detailed.

4.1. Preliminaries

4.1.1. Definitions.

Definition 1 (Energy density ED_i^m)**.** The energy density of a task τ_i on a core π^m is defined as $ED_i^m \stackrel{\text{def}}{=} \frac{E_i^m}{T_i}$, where E_i^m corresponds to the energy consumption of a task in L-mode. This is a measure of the average power dissipation in L-mode.

Definition 2 (Energy density difference EDD_i^m)**.** The energy density difference of a task τ_i on a core π^m is defined as $EDD_i^m \stackrel{\text{def}}{=} \min\{ED_i^h : h \neq m \wedge ED_i^h \geq ED_i^m\} - ED_i^m$. This value corresponds to the difference between the next higher energy density of τ_i on any other core and the energy density on the current core.

Definition 3 (Low criticality utilisation difference LUD_i^m)**.** The low criticality utilisation difference of a task τ_i on a core π^m is defined as $LUD_i^m \stackrel{\text{def}}{=} \min\{U_i^h(L) : h \neq m \wedge U_i^h(L) \geq U_i^m(L)\} - U_i^m(L)$. This value is the difference between the next higher low criticality utilisation of τ_i on any other core and the low criticality utilisation on the current core.

Definition 4 (High criticality utilisation difference HUD_i^m)**.** The high criticality utilisation difference of a task $\tau_i \in \tau(H)$ on a core π^m is defined as $HUD_i^m \stackrel{\text{def}}{=} \min\{U_i^h(H) : h \neq m \wedge U_i^h(H) \geq U_i^m(H)\} - U_i^m(H)$. This quantity represents the difference of next higher high criticality utilisation of $\tau_i \in \tau(H)$ on any other core and the high criticality utilisation on the current core.

Definition 5 (favourite / least preferred core)**.** The favourite core of a task with respect to the parameter p is defined as the core on which the value of the parameter p of this task is minimal. Similarly, the least preferred core of a task is defined as the core on which the value of the parameter p of this task is maximal. This parameter p may correspond to energy consumption, L-mode utilisation, H-mode utilisation or energy density etc.

4.1.2. Density difference lists. We compute EDD_i^m and LUD_i^m for all tasks, and HUD_i^m for high criticality tasks (i.e., $\forall \tau_i \in \tau(H)$) on each core. Afterwards, we generate the following three lists out of these values.

1) Assume a set LUD such that $LUD \stackrel{\text{def}}{=} \left\{LUD_1^{f_1}, LUD_2^{f_2}, LUD_3^{f_3}, \ldots, LUD_{|\tau|}^{f_{|\tau|}}\right\}$, where $LUD_i^{f_i}$ is the low criticality utilisation difference of task τ_i on core π^{f_i} and $\pi^{f_i} \in \pi$ corresponds to the favourite core of a task τ_i with respect to L-mode utilisation parameter (i.e., $U_i^{f_i}(L)$ is minimal on core π^{f_i}). S^{LUD} is a list that contains a sequence of numbers of LUD sorted in a non-increasing order.

2) Assume a set $HT \stackrel{\text{def}}{=} \left\{HUD_{j_1}^{f_1}, HUD_{j_2}^{f_2}, \ldots, HUD_{j_{|\tau(H)|}}^{f_{|\tau(H)|}}\right\}$, where $HUD_{j_x}^{f_x}$ is the high criticality utilisation difference of task $\tau_{j_x} \in \tau(H)$ on core π^{f_x} and $\pi^{f_x} \in \pi$ corresponds to the favourite core of a task τ_{j_x} with respect to H-mode utilisation parameter (i.e., $U_{j_x}^{f_x}(H)$ is minimal on core π^{f_x}). Similarly, assume another set $LT \stackrel{\text{def}}{=} \left\{LUD_{k_1}^{f_1}, LUD_{k_2}^{f_2}, \ldots, LUD_{k_{|\tau(L)|}}^{f_{|\tau(L)|}}\right\}$, where $LUD_{k_x}^{f_x}$ is the low criticality utilisation difference of task $\tau_{k_x} \in \tau(L)$ on core π^{f_x} and $\pi^{f_x} \in \pi$ corresponds to the favourite core of a task τ_{k_x} with respect to L-mode utilisation parameter (i.e., $U_{k_x}^{f_x}(L)$ is minimal on core π^{f_x}). Let S^{HT} and S^{LT} represent the sequence of elements of set HT and LT, respectively, sorted in a non-increasing order, then, S^{HUD} is a list that concatenates S^{HT} and S^{LT}, i.e., $S^{HUD} \stackrel{\text{def}}{=} S^{HT}||S^{LT}$, where $||$ represents the concatenation sign.

3) Assume that EDD is a set defined as $EDD \stackrel{\text{def}}{=} \left\{EDD_1^{f_1}, EDD_2^{f_2}, EDD_3^{f_3}, \ldots, EDD_{|\tau|}^{f_{|\tau|}}\right\}$, where $EDD_i^{f_i}$ is the energy density difference of a task τ_i on core π^{f_i} and $\pi^{f_i} \in \pi$ corresponds to the favourite core of a task τ_i with respect to energy density parameter (i.e., $ED_i^{f_i}(L)$ is minimal on core π^{f_i}). S^{EDD} is a list that contains all the elements of EDD sorted in a non-increasing order.

Algorithm 1 Improved least-loss energy density algorithm (ILLED)

Input: Sorted list of elements with current density difference values CDD in a non-increasing order, τ and π
Output: Assignment corresponding to CDD

```
Allocations = 0
while CDD is not empty do
    Assume DD_i^m corresponds to the top most entry in a list CDD with the highest density difference value
    τ(π^m) := τ(π^m) ∪ {τ_i}
    if Ekberg_Yi_Analysis(τ(π^m)) == SUCCESS then
        Remove DD_i^m from set CDD
        Allocations := Allocations +1
    else
        τ(π^m) := τ(π^m) \ {τ_i}
        DD_i^h := min{DD_i^h : h ≠ m ∧ D_i^h ≥ D_i^m}, where D_i^m is either energy density or utilisation. In other words, get the density difference value of task τ_i on its next preferred core
        if DD_i^h does not exists then
            break;
        else
            Reinsert DD_i^h in a list CDD maintaining its non-increasing order
        end if
    end if
end while
if Allocations == n then
    return Tasks assignment on the platform
else
    return Task-set cannot be assigned with current CDD
end if
```

4.1.3. Improved Least-Loss Energy Density Algorithm (ILLED). The improved least-loss energy density algorithm was originally proposed by Awan et al. [6] for single criticality systems. This algorithm can be easily adapted for mixed criticality systems by just changing the feasibility test. We present the resulting pseudo-code in Algorithm 1. It takes as an input a sorted list of tasks in a non-increasing order of their density difference values on their favourite cores. This list is termed as current

density difference list CDD. The density difference value is a generic term used to indicate any of the following quantities, i) low criticality utilisation difference, ii) high criticality utilisation difference or iii) energy density difference. The density difference of a task shows how much a system will lose in terms of a given criteria (utilisation or energy) if a task is not allocated to its preferred core. This ranking plays an important role in the allocation process. The tasks ranked higher have the higher probability of getting mapped to their preferred core.

In the allocation process (lines 2 to 17), the algorithm allocates the task corresponding to the highest current density difference value in CDD to its favourite core and checks the feasibility on its favourite core using the Ekberg and Yi [20] analysis. If the allocation is feasible, it is made permanent and its entry in the CDD list is removed (lines 5 to 7). Otherwise, the task is removed from this core (line 9) and its next density difference on the next preferred core is computed (line 10). This newly computed density difference values are added again in the CDD list at the appropriate location maintaining its non-increasing order (line 14). This process is repeated again unless, i) all the tasks are allocated and the CDD list is empty or ii) any task within a given task-set cannot be allocated even to its least preferred core (lines 11 to 12).

4.2. Proposed Algorithm

Our proposed approach generates a set of feasible allocations and selects the one that minimises the overall energy consumption of the system. Each feasible allocation ensures the schedulability of assigned tasks on each core both in L-mode and H-mode. To compute a corpus of feasible allocations, we run several times the ILLED algorithm (Algorithm 1) on different sorted lists of tasks. The pseudo-code of the proposed approach is presented in Algorithm 2. In the rest of this section, we explain the details of the proposed algorithm and provide the rational behind the selection of the input-sorted-task-lists. The ILLED algorithm applied on a given sorted list of tasks σ is denoted as $ILLED(\sigma)$.

The first step is to generate S^{LUD}, S^{HUD} and S^{EDD} density difference lists (line 1).

The first run of the ILLED algorithm is then made with the list S^{EDD}. The tasks in S^{EDD} are sorted with respect to energy density difference values and hence, in case of success, $ILLED(S^{EDD})$ generates a mapping that minimises the energy consumption. The algorithm ends in that case (line 5). However, if $ILLED(S^{EDD})$ returns failure, we further search for a feasible task-to-core mapping ensuring schedulability in both modes.

The second step is then to run ILLED with lists derived from S^{LUD}. Indeed, in practice, mixed criticality systems mostly stay in the L-mode and occasionally transition into the H-mode due to an overrun or other external/internal events. Therefore, it is more beneficial to optimise the energy efficiency in the L-mode. The allocation given by $ILLED(S^{LUD})$ will maximises the schedulability in the L-mode as it is sorted with respect to L-mode utilisation. However, there may be a possibility that it fails because the system is not feasible in the H-mode. Therefore, we keep applying $ILLED(S^{LUD})$ but

Algorithm 2 Pseudo-code of the proposed algorithm

Input: τ, π
Output: Assignment
1: Compute S^{EDD}, S^{LUD} and S^{HUD}
2: Assume A[i] represent the i^{th} assignment
3: A[0] := Perform the ILLED allocation (Algorithm 1) with S^{EDD}
4: **if** (A[0] is feasible) **then**
5: **return** A[0]
6: **end if**
7: $i := 0$
8: **while** (true) **do**
9: A[i] := Perform the ILLED allocation with S^{LUD}
10: **if** (A[i] is feasible) **then**
11: EE[i] := Compute energy efficiency of A[i] through Equation 1
12: $i := i + 1$
13: **end if**
14: Promote high criticality task's entry in S^{LUD}
15: **if** (Promotion fails) **then**
16: break
17: **end if**
18: **end while**
19: A[i] := Perform the ILLED allocation with S^{HT} followed by S^{LT},
20: **if** (A[i] is feasible) **then**
21: $EE[i]$:= Compute energy efficiency of A[i] through Equation 1
22: **end if**
23: index := Find index that gives $\min_{\forall i}(EE[i])$
24: **return** A[index]

with progressive alterations on S^{LUD} to make it tends toward S^{HUD}. Indeed the ordering of S^{HUD} is the one that will maximise the schedulability in the H-mode. We start with S^{LUD} and gradually transform it by promoting one by one the H-tasks. The details of the S^{LUD} reordering mechanism are explained as follows. We select the H-task corresponding to the entry on top of the S^{HUD} list and promote its corresponding entry in S^{LUD} by one position. If the task corresponding to the top element in S^{HUD} is also at the same level as in S^{LUD}, we need to select the task corresponding to the second element of S^{HUD} in S^{LUD} and promote it by one position in S^{LUD} unless it reaches the same position as in S^{HUD}.

The following example explains the process of tasks reordering in the S^{LUD} list. Assume that we have only three H-tasks $\tau = \{\tau_1, \tau_2, \tau_3\}$ in the system. Also assume that $\{\tau_3, \tau_2, \tau_1\}$ and $\{\tau_1, \tau_2, \tau_3\}$ are the task orderings in a list S^{HUD} and S^{LUD}, respectively. To find the second ordered list of tasks in S^{LUD}, we select τ_3 in S^{LUD} to promote (as this is the first task in S^{HUD}) and get an ordered list of $\{\tau_1, \tau_3, \tau_2\}$. The further promotion of τ_3 will give us a task ordering of $\{\tau_3, \tau_1, \tau_2\}$. At this point, we cannot further promote τ_3, therefore, we promote the second task τ_2 and get a task ordering of $\{\tau_3, \tau_2, \tau_1\}$. We finish the reordering process here as we cannot further promote any task.

After all these promotions of H-tasks, S^{LUD} will have the same ordering as given in S^{HUD}. However, S^{LUD} does not represent the same list as S^{HUD} since H-tasks

TABLE 1. OVERVIEW OF PARAMETERS

Parameters	Values
Task set sizes n	$\{8, 10, \underline{12}, 16, 24\}$
Pct. of H-tasks (phct)	$\{20\%, 30\%, \underline{40\%}, 50\%, 60\%\}$
Number of processors M	$\{2, 3, \underline{4}, 5\}$
$U_i^m(H)$ multiplier $k \in$	$\{2, \underline{3}, 4\}$
Characteristic factor $\beta \in$	$\{5\%, \underline{10\%}, 15\%, 20\%, 25\%, 30\%\}$
Utilisation helper variable ζ	$\{0.1, 0.15, 0.2, 0.25, \ldots, 0.95\}$

TABLE 2. PROCESSORS POWER MODEL PARAMETERS

π^m	π^1/P^D	π^2	π^3	π^4	π^5
η^m	1.0	0.75	0.6	0.5	0.4
P_a^m	7.5	10	12.1	15	17.5

are ordered with respect to HUD_i^m in S^{HUD}. Therefore, as a final step, we run $ILLED(S^{HUD})$.

During these three steps, we store each feasible allocation returned by the successive runs of the ILLED algorithm. Then we can compute the energy efficiency of all these allocations. The energy efficiency (EE) of any allocation on an heterogeneous multicore platform is computed with a metric called *overall average power dissipation of the system* and is defined in Equation 1.

$$EE \stackrel{\text{def}}{=} \sum_{\forall \pi^m \in \pi} \sum_{\forall \tau_i \in \tau(\pi^m)} ED_i^m \tag{1}$$

We can therefore select the allocation that minimises this metric and return it (lines 23, 24).

Complexity Analysis: The worst-case — in which our proposed algorithm has to apply the ILLED algorithm on the maximum number of sorted lists of tasks — occurs when i) the S^{EDD} list is not feasible, ii) a task-set is composed of only high criticality tasks and iii) S^{LUD} is in reverse order when compared to S^{HUD}. In such a scenario, the high criticality tasks from bottom to the top in the S^{LUD} list will be shifted by $(n-1), (n-2), (n-3), \ldots, (n-n+1), (n-n)$ times, respectively. Hence, there are maximum of $(n-1)+(n-2)+(n-3)+\ldots+(1) = \frac{n^2-n}{2}$ lists due to re-ordering in S^{LUD} plus additional overhead of S^{EDD} and S^{HUD}. In total, our proposed algorithm has to check $\frac{n^2-n+4}{2}$ lists. The ILLED algorithm has a complexity of $O(n \times M)$, i.e., this algorithm performs Ekberg and Yi analysis [20][1] for $n \times M$ times. Therefore, in the worst-case, our proposed algorithm has to perform $\frac{(n^3-n^2+4n) \times M}{2}$ Ekberg and Yi analyses. In reality, S^{EDD} is feasible for most of the task-sets. We show the percentage of feasible S^{EDD} instances for the set of experiments performed in our evaluation section, next.

5. Evaluation

The performance of the proposed allocation heuristic is analysed and compared to state-of-the-art in this section.

1. The details on the complexity of Ekberg and Yi analysis are detailed in their original work [20].

5.1. Experimental setup

We developed a Java tool to implement the proposed approach and a random task generator. The heterogeneous multicore platform used in this evaluation is derived from a Freescale PowerQUICC III integrated communication processor MPC8536 [21]. The parameters of the derived cores are presented in Table 2. In that table, P_a^m and η^m denote the average active power dissipation at maximum frequency and the speed-up factor, respectively. The speed-up factor of core π^m is the ratio of the clock cycle of π^m and π^D, where π^D is the default core. We can choose any core as the default core π^D. In this experimental setup, we selected π^1 as the default core. The speed-up factors are used to compute the average computing capacity U_a of the platform. The average computing capacity is defined as $U_a \stackrel{\text{def}}{=} \frac{1}{\eta^1} + \frac{1}{\eta^2} + \ldots + \frac{1}{\eta^m}$. The effective utilisation U of a task set is generated with a helper variable ζ, and $U \stackrel{\text{def}}{=} U_a \times \zeta$. The value of the helper variable is varied from 0.1 to 0.95 with a step size of 0.05.

Initially, a task set is generated with a given effective utilisation U for a default core π^D. Afterwards, the parameters of the individual tasks are computed for the other cores on the heterogeneous multicore platform. The following approaches and mechanisms are used to compute the tasks' parameters for the default core.

- **Task utilisation.** We compute the utilisation of each task using the UUnifast-discard algorithm [22]. This approach allows unbiased distributions of utilisation values. A task set is discarded if the utilisation of an individual task exceeds 1 or the task-set utilisation exceeds the effective utilisation U.
- **Task periods.** We use the log-uniform distribution to generate periods for all tasks. The period of a task is selected within an interval of 10ms to 100ms.To get a period with a log-uniform distribution, a random number x is generated within $[log_{10}10, log_{10}100]$ and raised to the power of 10, i.e., $T_i = 10^x : x \in [log_{10}10, log_{10}100]$.
- **Task deadline.** The approach in this paper can be applied to constrained deadline model. However, for the sake of simplicity, we assume an implicit deadline model in which task deadlines are equal to their periods.
- **Distribution of high and low criticality tasks.** In a given task-set size, we assign a constant percentage to the high criticality tasks. This percentage assignment is detailed at the end of this section.
- **WCET in L-mode.** To compute this, we multiply the period of a task with its generated utilisation.
- **WCET in H-mode.** This value is computed by multiplying the WCET in the low criticality mode by a constant factor $k > 1$. However, this approach can occasionally lead to a utilisation in the high criticality mode greater than 1.This issue is solved with a transfer function which, for small values of $U_i^D(L)$, would approximate multiplication with constant factor, but for greater values would progressively reduce the gain, so that we never get $U_i^D(H) > 1$. The details of the transfer function are provided in Appendix A.
- **System resolution.** Our tool considers a resolution in the order of the microsecond.

A task set generated for the default processor π^D is considered valid if the sum of the utilisations of all tasks in the low criticality mode and/or high criticality mode is less than the effective utilisation U. The UUnifast-discard algorithm ensures that any task in the low criticality mode will not have an utilisation greater than 1. Similarly, our transfer function does not allow the utilisation of a task in high criticality mode to go beyond 1. This latter valid criterion is a necessary but not sufficient condition for the task set to be considered feasible. In this paper, a task set is deemed valid (VT) if $\sum_{\forall \tau_i \in \tau(H)} U_i^D(H) \leq U_a$ and $\sum_{\forall \tau_i \in \tau} U_i^D(L) \leq U_a$. The parameters of the tasks generated for π^D are computed with the help of a characteristic factor β that models the fact that tasks behave differently in terms of execution and energy consumption on different cores. On any core π^m, the WCET of a task in the low and high criticality modes is computed as $U_i^m(L) = [\eta^m(1-\beta), \eta^m(1+\beta)]U_i^D(L)$ and $U_i^m(H) = [\eta^m(1-\beta), \eta^m(1+\beta)]U_i^D(H)$. Similarly, the average energy consumption is computed to be $E_i^m = [P_a^m(1-\beta), P_a^m(1+\beta)]U_i^m(L)$. We follow the mechanism proposed by Raj Jain [23] to reuse the seed in successive replications and create different objects of random class (seeded with different odd integers) to generate random values for periods, utilisations and energy consumption. Each set of input parameter values is repeated 100 times.

An overview of the parameters used in our experiments is provided in Table 1. The underlined values are default values, if not specified in the description of an individual experiment. An heterogeneous multicore platform is used for variety of complex applications, therefore, the task-set size is varied from 8 up to 24 tasks. The percentage of high criticality tasks is varied from 20% up to 60% of the task-set size. We investigate multicore platforms with $2-5$ processors, with different features. The value of the characteristic factor is varied from 5% up to 30%. Three different values of the constant parameter $k \in 2, 3, 4$ are used to compute the WCET in the high criticality mode.

5.2. Results

We compared the following approaches for the given set of parameters in our experiments.

- **Naive First-fit allocation (NFF).** In this approach, allocation is performed following a first-fit bin packing heuristic on cores arranged in non-increasing order with respect to their speedup factor. The schedulability on any core π^m is tested by considering $U_i^m(H)$ and $U_i^m(L)$ for high and low criticality tasks, respectively.
- **Partitioned Ekberg (PEKB).** This approach also performs allocations using a first-fit bin packing heuristic. Similar to NFF, cores are arranged in a non-decreasing order of their speedup factor. The schedulability of each core is tested with the Ekberg and Yi [20] analysis.
- **Random allocation (RA).** As the name suggests, in this approach, we perform random allocations. Initially, a task is randomly selected from a given task set and allocated to a randomly selected core. We use the Ekberg and Yi's [20] analysis to ensure the schedulability during allocation.
- **Mixed criticality power management allocation (MCPM).** This is our proposed novel approach.

We compare the overall average power dissipation of the system (energy efficiency) and the schedulability of all aforementioned algorithms. The overall average power dissipation (APD) of a given task-set with any allocation algorithm x on the given platform is computed to be $APD_x = \sum_{\forall \pi^m \in \pi} \sum_{\forall \tau_i \in \tau(\pi^m)} ED_i^m$ (Equation 1). We consider the PEKB as a baseline algorithm and compare the gain of the other algorithms over this approach. The gain[2] (in average power dissipation) of any algorithms x over PEKB is given by $\frac{APD_{PEKB} - APD_x}{APD_{PEKB}}$, where $x \in \{NFF, RA, MCPM\}$. The weighted schedulability measure [24] is the second performance metric used to compare the various algorithms. It is a weighted average, in which more weight is given to task sets with higher utilisation. We borrow the notation from [25] and adapt the metric for effective utilisation on multicore platforms. Assume that $W_y(p)$ represents the weighted schedulability measure for schedulability test y as a function of parameter p. For each parameter p, this measure combines the results of the task sets generated for all the utilisations values mentioned in our Table 1. Let $S_y(\tau, p)$ represents the binary result (0 or 1) for any schedulability test y for a given task set τ with an input parameter p and $U(\tau)$ is the effective utilisation of a task set τ, then $W_p(p)$ is given by Equation 2 (this equation is taken from [25]).

$$W_y(p) = \frac{\left(\sum_{\forall \tau} U(\tau) \cdot S_y(\tau, p)\right)}{\sum_{\forall \tau} U(\tau)} \quad (2)$$

In our first experiment, we compare the effect of variation in $U_i^m(H)$ multiplier k on the average power dissipation of the various approaches. Figures 1, 2 and 3 present the gain of the different approaches (NPP, RA and MCPM) over the PEKB, for $k = 2$, 3 and 4, respectively. It becomes difficult with an increase in effective utilisation to find the energy efficient solution with MCPM and hence, there is a decrease in gain with an increase in the effective system utilisation. RA and NFF are energy agnostic algorithms and behave similar to PEKB with slight variations. In the best case, MCPM has a gain of approximately 9.12% over PEKB in this experiment. In general, the schedulability of all heuristics reduces with an increase in effective utilisation and some heuristics (such as NFF) even fail to schedule any task-set at high utilisations. This is illustrated in Figure 4 that presents the schedulability ratio (SR) for $k = 3$ (or default values given in Table 1). The increase in k also decreases the schedulability of all algorithms as shown through the weighted schedulability (WS) metric presented in Figure 5. The MCPM algorithm though not designed to achieve high schedulability remains superior when compared to other algorithms in terms of the schedulability metric.

2. We generate 100 random task sets for each set of input parameters and it represents a single point in our graphs. In order to compute the APD_x of any algorithm x with hundred runs, we divide the average power dissipation values of all the feasible allocations over the number of feasible solutions.

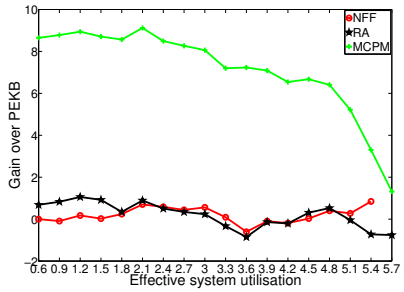

Figure 1. gain with $k = 2$

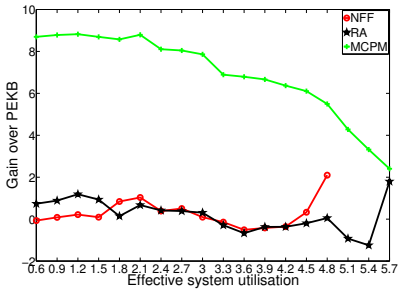

Figure 2. gain with default values

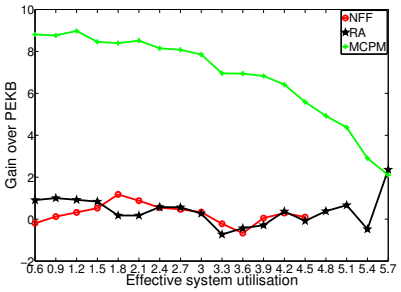

Figure 3. gain with $k = 4$

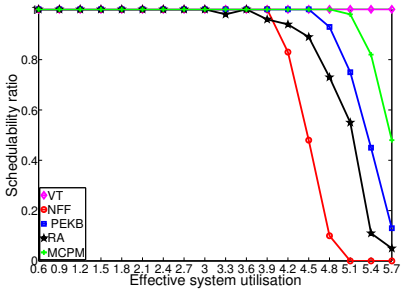

Figure 4. SR with default values

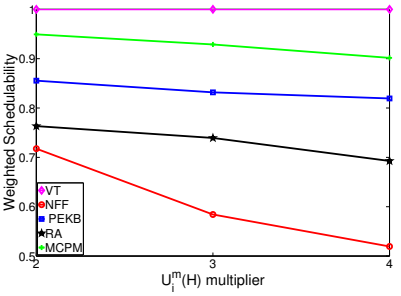

Figure 5. WS of $U_i^m(H)$ multiplier

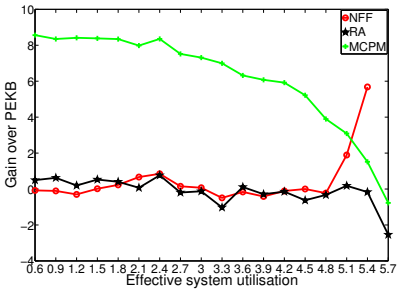

Figure 6. gain with $n = 8$

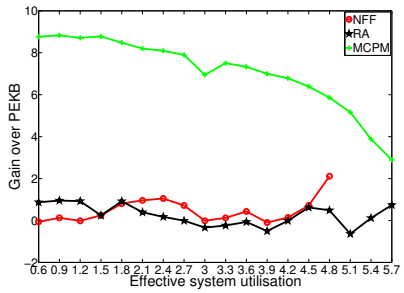

Figure 7. gain with $n = 16$

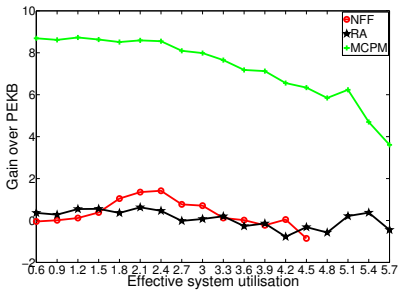

Figure 8. gain with $n = 24$

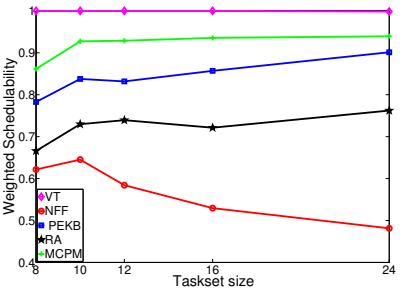

Figure 9. WS of taskset size

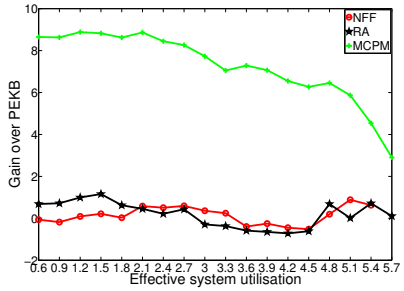

Figure 10. gain with $phct = 20\%$

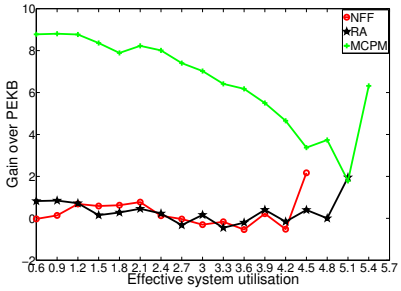

Figure 11. gain with $phct = 60\%$

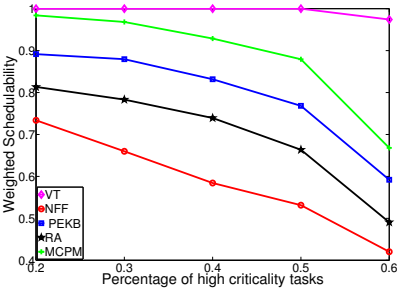

Figure 12. WS of PHCT

The gains of all algorithms over PEKB for task-set sizes of 8, 12, 16 and 24 are presented in Figures 6, 2, 7 and 8, respectively. The gain of MCPM increases with larger task-set sizes at high utilisations. There are some oddities in Figure 6: i) NFF shows an increase of gain at $U = 5.1, 5.4$, and ii) MCPM has lower gain when compared to PEKB. The NFF, RA and PEKB algorithms have fewer feasible solutions (with low average power dissipation) at high utilisations when compared to MCPM and this leads to their higher gains over MCPM. This behaviour is also illustrated in Figure 9 with a weighted schedulability metric that shows MCPM has more feasible solutions than the other approaches. Except NFF, the schedulability of all algorithms improves with larger task-set sizes. The NFF algorithm is negatively affected due to an increase in the relative number of high criticality tasks and a pessimistic schedulability test.

We also varied the percentage of high criticality tasks ($phct$) in a given task set and analyzed its effect on all heuristics. Figures 10, 2 and 11 present the results for $phct = 20\%$, 40% and 60%, respectively. A larger number of high criticality tasks leads to high utilisation in H-

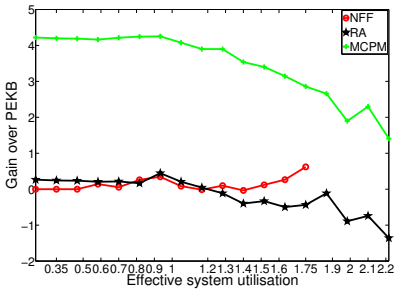

Figure 13. gain with $M = 2$

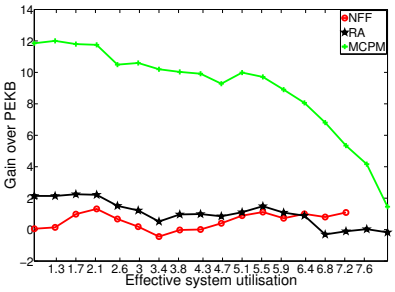

Figure 14. gain with $M = 5$

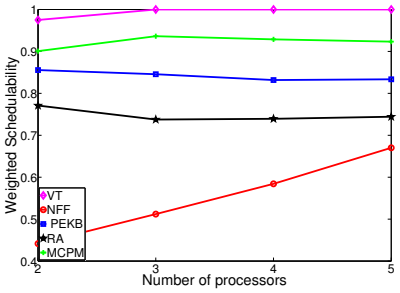

Figure 15. WS of M

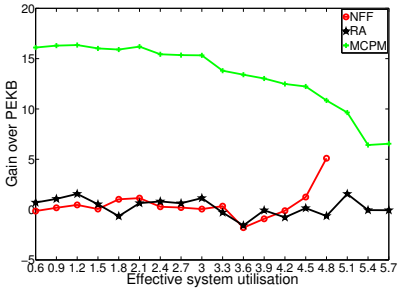

Figure 16. gain with $\beta = 20\%$

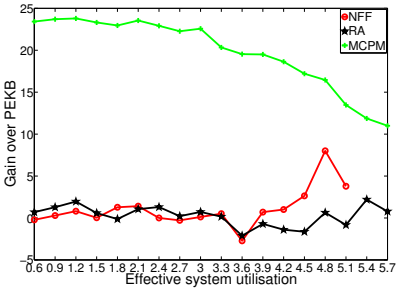

Figure 17. gain with $\beta = 30\%$

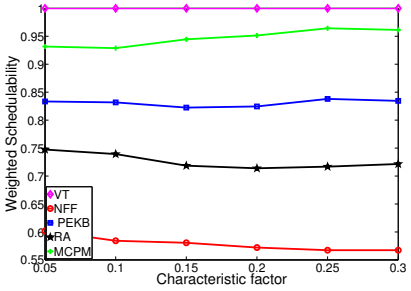

Figure 18. WS of characteristics factor

mode which makes it harder to find the feasible solution for an allocation heuristic. As it can be seen in Figure 11, not even a single task set was schedule at $U = 5.7$ with $phct = 60\%$ with any heuristic. Therefore, the gain of MCPM decreases with a higher value of $phct$ at high utilisations. The weighted schedulability graph given in Figure 12 confirms this observation.

The gain of MCPM over PEKB increases with an increase in the number of cores on the given platforms. Figures 13, 2 and 14 present the gain of allocation heuristics over PEKB for 2, 4 and 5 cores, respectively. Extra cores provide more flexibility to MCPM to find energy efficient solutions and hence, increased gain over PEKB. Figure 15 shows that the weighted schedulability of NFF improves with an increase in the number of cores. This results from the extra flexibility that larger platforms provide to schedule the given task set. The effect of characteristic factor β (that models the fact that tasks behave differently on different cores) is also analysed for different heuristics. Figures 2, 16 and 17 present the gain of different heuristics over PEKB for $\beta = 10\%$, 20% and 30%, respectively. In our experimental setup, cores are arranged in a non-increasing order of their speedup factor (or increasing order of power dissipation). This arrangement on average provides ideal conditions to the NFF and PEKB algorithms to first allocate tasks to slow cores (or energy efficient cores). The large values of β give more range to the tasks to choose their energy consumption values on each core. This increases the ability of low speedup cores to also come up with energy efficient alternatives. Hence, the gain of the MCPM algorithm increases with an increase in the value of β as it has the ability to find energy efficient allocations. In the best case ($\beta = 30\%$), MCPM has a gain of approximately 23.8%. Figure 18 shows that this metric does not vary much the weighted schedulability of allocation heuristics given it does not directly affect on the WCET of the tasks for different parameters.

TABLE 3. SUMMARY OF RESULTS

Metric	k	M	n	$phct$	β
E.G.	9.12%	12.01%	8.91%	8.93%	23.8%
S.D.	44%	43%	44%	63%	52%

The results are summarised in Table 3. The Energy gain (E.G.) metric presents the best-case gain of MCPM over PEKB corresponding to variation in different parameters. The schedulability difference (S.D.) metric shows the maximum difference between the schedulable task-set of MCPM and PEKB.

5.3. Complexity Discussion

In the majority of the cases, MCPM finds the energy efficient mapping with the S^{EDD} list and hence, it does not need to explore the other lists (S^{LUD} and S^{HUD}). Among total feasible task-sets, the S^{EDD} list is successful to provide the feasible solution with a percentage of 88.35%, 89.52%, 88.68%, 86.12% and 88.25% for all set of experiments performed with different values of k, β, M, $phct$ and n, respectively.

6. Conclusions and future directions

In this work, we have presented an energy-aware allocation heuristic to map mixed criticality applications (task-set) on an unrelated heterogeneous multicore platform. The proposed allocation heuristic considers both energy efficiency and schedulability constraints in the optimisation process. We consider a realistic power model where tasks' energy consumptions are not a function of their execution times. The extensive simulations suggest a substantial gain over the state-of-the-art algorithms both in terms of energy efficiency and schedulability. We plan to

extend this approach to a multiple criticality level model. Moreover, it is also interesting to consider the effect of I/O devices that many applications (tasks) may share among each other. The main idea of running several times the ILLED algorithm with different inputs could be adapted in this case, and more generally to any partitioning problem where multiple optimisation objectives exist.

Acknowledgements

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2) and ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO).

References

[1] A. Esper, G. Nelissen, V. Nelis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *Proceedings of RTNS*, 2015, pp. 139–148.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of RTSS*, 2007, pp. 239–243.

[3] D. Geer, "Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11–13, 2005.

[4] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: preparing for a new exponential," in *ICCAS 2006*, ser. ICCAD '06. ACM, 2006, pp. 67–72.

[5] M. A. Awan, P. M. Yomsi, G. Nelissen, and S. M. Petters, "Energy-aware task mapping onto heterogeneous platforms using dvfs and sleep states," *Journal of real-time systems*, pp. 1–36, 2015.

[6] ——, "Energy-aware task mapping onto heterogeneous platforms using dvfs and sleep states," *Real-Time Systems*, pp. 1–36, 2015.

[7] K. Lakshmanan, D. de Niz, R. Rajkumar, and G. Moreno, "Resource allocation in distributed mixed-criticality cyber-physical systems," in *Proceedings of ICDCS*, 2010, pp. 169–178.

[8] D. TamasSelicean and P. Pop, "Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures," in *Proceedings of RTSS*, 2011, pp. 24–33.

[9] O. Kelly, H. Aydin, and B. Zhao, "On partitioned scheduling of fixed-priority mixed-criticality task sets," in *Proceedings of TrustCom*, 2011, pp. 1051–1059.

[10] P. Rodriguez, L. George, Y. Abdeddaïm, and J. Goossens, "Multi-criteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors," in *Proceedings of WMC*, 2013.

[11] L. s. Florian Broekaert, Agnes Fritsch and S. Tverdyshev, "Towards power-efficient mixed critical systems," *OSPERT*, vol. 2013, pp. 30–35, 2013.

[12] V. Legout, M. Jan, and L. Pautet, "Mixed-Criticality Multiprocessor Real-Time Systems: Energy Consumption vs Deadline Misses," in *Proceedings of ReTiMiCS*, 2013, pp. 1–6.

[13] M. A. Awan and S. M. Petters, "Race-to-halt energy saving strategies," *Journal of Systems Architecture*, vol. 60, no. 10, pp. 796 – 815, 2014.

[14] X. Zhang, J. Zhan, W. Jiang, Y. Ma, and K. Jiang, "Design optimization of security-sensitive mixed-criticality real-time embedded systems," in *Proceedings of ReTiMiCS*, 2013.

[15] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," in *Proceedings of EMSOFT*, 2014, pp. 11:1–11:10.

[16] M. Vlp, M. Hhnel, and A. Lackorzynski, "Has energy surpassed timeliness? scheduling energy-constrained mixed-criticality systems," in *Proceedings of RTAS*, 2014, pp. 275–284.

[17] G. G. L. T. Sujay Narayanayz, Pengcheng Huangy and R. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," in *Proceedings of RTAS*, 2016, pp. 135–146.

[18] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[19] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, "Koala: A platform for os-level power management," in *Proceedings of EuroSys*, 2009, pp. 289–302.

[20] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems*, vol. 50, no. 1, pp. 48–86, 2014.

[21] F. Semiconductor, *MPC8536E PowerQUICC III Integrated Processor Hardware Specifications*, number: MPC8536EEC,Rev. 5, 09/2011. [Online]. Available: http://www.freescale.com/files/32bit/doc/data_sheet/MPC8536EEC.pdf

[22] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2009.

[23] R. Jain, *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.*, ser. Wiley professional computing. Wiley, 1991.

[24] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proceedings of 6th annual workshop on Operating Systems Platforms for Embedded Real-Time applications*, pp. 33–44, 2010.

[25] A. Burns and R. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in *Proceedings of RTSS*, 2014, pp. 21–30.

[26] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling with temporal isolation and deadline scaling," in *Under submission to ECRTS*, 2016.

Appendix A. Transfer function

TABLE 4. CORRESPONDING VALUES OF k AND z AND CLOSED-FORM EXPRESSION FOR THE TRANSFER FUNCTION f.

k	$z(k)$	$f(U,k)$ (from Equation 3)
2	4.92155	$f(U,2) = 1.25500 \cdot (1 - 4.92155^{-U})$
3	16.80101	$f(U,3) = 1.06329 \cdot (1 - 16.80101^{-U})$
4	50.43525	$f(U,4) = 1.02023 \cdot (1 - 50.43525^{-U})$

This transfer function was initially used by Awan et al. [26] in their work on mixed criticality semi-partitioned scheduling algorithms. It provides an efficient mechanism to generate more feasible task-sets. As explained in Section 5.1 that to generate $U_i^m(H)$ from $U_i^m(L)$ we use a continuous "transfer function" f which, for small values of $U_i^m(L)$ approximates multiplication with k, but for greater values *progressively* reduces the gain, so that we always get $U_i^m(H) \leq 1$. After some experimentation, we came up with the form

$$f(U) = \frac{z}{z-1} \cdot (1 - z^{-U}) \tag{3}$$

with z chosen such that $f'(0) = k$. Using calculus (details omitted), the corresponding values for z as a function of k, and the closed form expressions for the transfer function $f(U,k)$, with k as an additional parameter, are given in Table 4.