



**CISTER**

Research Center in  
Real-Time & Embedded  
Computing Systems

# Technical Report

---

## **Energy-Aware Partitioning of Tasks onto a Heterogeneous Multi-core Platform**

**Muhammad Ali Awan**

**Stefan M. Petters**

---

CISTER-TR-130506

Version:

Date: 5/29/2013

# Energy-Aware Partitioning of Tasks onto a Heterogeneous Multi-core Platform

Muhammad Ali Awan, Stefan M. Petters

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: [muaan@isep.ipp.pt](mailto:muaan@isep.ipp.pt), [smp@isep.ipp.pt](mailto:smp@isep.ipp.pt)

<http://www.cister.isep.ipp.pt>

## Abstract

Modern multicore processors for the embedded market are often heterogeneous in nature. One feature often available are multiple sleep states with varying transition cost for entering and leaving said sleep states. This research effort explores the energy efficient task-mapping on such a heterogeneous multicore platform to reduce overall energy consumption of the system. This is performed in the context of a partitioned scheduling approach and a realistic power model, which improves over some of the simplifying assumptions often made in the state-of-the-art. The developed heuristic consists of two phases, in the first phase, tasks are allocated to minimise their active energy consumption, while the second phase trades off a higher active energy consumption for an increased ability to exploit savings through more efficient sleep states. Extensive simulations demonstrate the effectiveness of the approach.

# Energy-Aware Partitioning of Tasks onto a Heterogeneous Multi-core Platform

Muhammad Ali Awan      Stefan M. Petters  
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal  
maan,smp@isep.ipp.pt

**Abstract**—Modern multicore processors for the embedded market are often heterogeneous in nature. One feature often available are multiple sleep states with varying transition cost for entering and leaving said sleep states. This research effort explores the energy efficient task-mapping on such a heterogeneous multicore platform to reduce overall energy consumption of the system. This is performed in the context of a partitioned scheduling approach and a realistic power model, which improves over some of the simplifying assumptions often made in the state-of-the-art. The developed heuristic consists of two phases, in the first phase, tasks are allocated to minimise their active energy consumption, while the second phase trades off a higher active energy consumption for an increased ability to exploit savings through more efficient sleep states. Extensive simulations demonstrate the effectiveness of the approach.

## I. INTRODUCTION

For embedded real-time (RT) systems it is imperative that timing constraints posed by the environment are met. In this general context a number of trends can be identified. Firstly, Moore's law is no longer sustained by increasing clock frequencies, but rather by addition of extra cores in multi-processors. This is driven for example, by the performance per watt ratio, as higher clock ratios demand also higher supply voltages. Besides symmetric multicore processors, homogeneous and heterogeneous multicores gain in popularity. The move beyond symmetric multicores is driven by both using cores geared to perform specific tasks well and cheap. A second trend is an increased interest in multi-criticality devices, where part of the system is critical and other parts are executed in a best effort manner. Finally, there is the move towards increased use of embedded devices with limited energy supply. These might be, for example, solar powered devices in the field or handheld rechargeable devices. In these kind of devices effective management of the limited resource (energy) is another constraint in the system requirements.

The real-time community has recognised these trends and provided solutions to these challenges. However, in most cases, the power and energy models used make many simplifying assumptions, which limit the applicability of the presented solutions. Common assumptions are, on one side homogeneous multicore processors with a constant speed factor between the

different cores; on the other side, the energy consumption of different applications are only a function of execution time rather than other task characteristics (e.g. number of cache misses). The latter has been shown to be widely off the mark [1]. Finally, the use of multiple available sleep states is rare.

The fact that task characteristics, like the cache miss pattern have an influence on the energy consumption beyond the mere change of execution time, means that analytical solutions are bound to be suboptimal for most specific cases. As such, the way forward is an effective heuristic to be used for energy management. Within this work, we assume that the system has such non-linear dependencies on execution time and energy consumption and several sleep states. In order to guarantee the temporal isolation requirement, we work with a partitioned scheduling approach. The underlying approach per CPU, ERTS [2], allows reconfiguration at run-time and thus enables limited migration, however, in this work we focus on the task partitioning and mapping problem. In the allocation stage the approach considers average-case energy consumption as objective function, considering real-time constraints based on worst-case execution and minimum inter-arrival time.

The proposed approach is divided into two phases. Firstly, the novel algorithm performs assignments with an objective to reduce the active/dynamic energy consumption of the system by allocating tasks to their favourite processors. A processor is considered favourite for a task where its active energy consumption is minimal when compared to all other processor types. In the second phase, it trades off the higher active energy consumption of tasks to enhance the processor's ability to use more efficient sleep states. The sleep states allow the processor to reduce the static power consumption of the system in idle intervals. The second phase is motivated by the fact that the static power consumption has become non negligible portion of the overall energy consumption of the system.

Traditional task assignment algorithms aim to reduce the active power consumption of the system by assigning the tasks to their favourite processor, while ignoring the static power consumption. The management of the static power consumption of the processor depends on the properties of the tasks such as their respective minimum inter-arrival times and worst-case execution times. For instance, assume the task assignment is such that it generates large amount of idle intervals in combination with a short period task. The processor may not be able to exploit it to use deeper sleep states due to a combination of the larger transition overhead of those and a short period task.

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within REPOMUC project, ref.FCOMP-01-0124-FEDER-015050, and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/70701/2010.

The paper is organised as follows. Section II discusses the related work followed by the system model in Section III. Section IV presents the two phase approach to do the task assignment followed by the experimental setup and results given in Section V. We conclude and provide future directions in Section VI.

## II. RELATED WORK

Energy efficient scheduling for the homogeneous multiprocessors has been widely explored in RT systems in the last decade. For instance, Kandhale et al. [3] recently presented a Single-clock domain multi-processor Frequency Assignment Algorithm (SFAA) for periodic, implicit deadline tasks under fixed priority (Rate-Monotonic) scheduling. It exploits the task period relationships to determine energy efficient frequency assignment. Chen et al. [4] provided a comprehensive survey of such techniques. In contrast, the state-of-the-art in power-aware heterogeneous multiprocessors is limited.

Yu and Prasanna [5] proposed the static allocation of the tasks in a RT system for the heterogeneous processing units under Dynamic Voltage Scaling (DVS). They formulated the problem as an Integer Linear Programming (ILP) and provided a linearisation heuristics. A pseudo polynomial time greedy algorithm [6] is proposed by Huang et al. for the frame-based RT task model and heterogeneous systems. Furthermore, a greedy heuristics is provided to migrate the tasks from the overloaded processor to reduce energy consumption. Luo and Jha addressed the task model with precedence constraints and proposed the list-scheduling strategy [7] for the heterogeneous distributed systems. Chen and Thiele [8] considered a case of 2-type heterogeneous processors and proposed a polynomial time approximation scheme based on the ratio of task execution times on the different processor types. The synthesis problem for heterogeneous platform is addressed by Hsu et al. [9] for the RT task model. They proposed an approximation algorithm based on a rounding technique by applying a parametric relaxation on an ILP to minimise the processor cost under the given timing and energy cost. Hung et al. [10] considered a heterogeneous platform with 2 processing elements, one with DVS enabled core and second without DVS capability, with an objective to reduce the overall energy consumption and maximise the energy saving in migration from DVS enabled core to non-DVS core. While DVS has its advantages, the state-of-the-art [5]–[10] ignores the static power consumption. We focus on the shut-down mechanism in this paper that effectively exploits the idle intervals in the schedule to reduce the static power consumption of the system that has become a considerable factor of the overall power consumption of the modern embedded systems.

Yang et al. [11] proposed an approximation algorithm based on dynamic programming and provides polynomial-time solution when the number of processor types is a small constant. However, in the general case when the restriction over the number of processor types is relaxed, this scheme has exponential time/space complexity. They also assume static power consumption of the system as a constant factor. The work of Chen et al. [12] presented a task assignment algorithm

for periodic real-time tasks on heterogeneous platforms. The problem is formulated as an ILP problem. They relax some of the assumptions to adopt it into linear programming (LP) and solve it through extreme point theory [13]. The tasks assigned fractionally in the previous steps are reassigned through known heuristics such first-fit, best-fit, worst-fit or last-fit. They ([11], [12]) assume the static power consumption of the system is a constant factor and it cannot be reduced due to the significant overhead of the sleep transitions. This assumption does not hold for modern processors which contains several sleep states to reduce the static power consumption of the system. Moreover, the static power consumption has become a considerable part of the overall energy consumption. Therefore, the effect of the task allocation on the power consumption in the sleep states should be considered to avoid suboptimal assignments.

Our proposed algorithm is based on the realistic power model. It considers the effect of task properties on both active and static power consumption of an assigned processor. In the context of heterogeneous multicores, the state-of-the-art assumes only dynamic power consumption, either ignores static power consumption or considers it a constant factor while doing task allocation on such platforms.

## III. SYSTEM MODEL

### A. Platform

We assume a partitioned multicore architecture, with  $M$  different types of heterogeneous processors/cores. Each processor type has a unique characteristic of power consumption and execution capability when compared to others. We consider only a single processing unit of each processor type  $\pi^m$ , for the separation of concerns and ease of notation. Each processor type  $\pi^m$  has a utilisation of  $U^m$ .

### B. Task Model

We assume sporadic task-model with  $\ell$  independent tasks  $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \dots, \tau_\ell\}$ . Each task  $\tau_i$  is represented as a quadruple  $\langle C_i^{\text{all}}, D_i, T_i, \bar{E}_i^{\text{all}} \rangle$ , where  $C_i^{\text{all}}$  is a vector of worst-case execution times of  $\tau_i$  on  $M$  different processor types.  $D_i$  is the deadline and  $T_i$  is the minimum-inter arrival time.  $\bar{E}_i^m$  is a vector of the average-case energy consumption of  $\tau_i$  on  $M$  different processor types at their maximum speed. The worst-case execution time and energy consumption of task  $\tau_i$  on processor type  $\pi^m$  is represented as  $C_i^m$  and  $\bar{E}_i^m$  respectively. As derived value individual utilisation of  $\tau_i$  on  $\pi^m$  is  $U_i^m = C_i^m / T_i$ . For the sake of simplicity, we assume implicit deadline meaning  $D_i = T_i$  for  $\tau_i$ .

Each independent task will release a sequence of unlimited jobs  $j_{i,k}^m = \langle r_{i,k}, \hat{c}_{i,k}, d_{i,k} \rangle$ , where  $r_{i,k}$ ,  $\hat{c}_{i,k}$  and  $d_{i,k}$  are the absolute release time, actual execution time and absolute deadline respectively. Jobs of the same task are allowed to vary their execution between  $\tau_i$ 's best-case execution time (BCET) and the worst-case execution time (WCET).

The Enhanced Race-To-Halt (ERTH) algorithm [2] is used on each processor, which is a leakage aware energy management approach for dynamic priority systems. It allows multiple sleep states per processor and utilises spare capacity available online to save total energy consumption of the system. ERTH

is based on the Rate-Based Earliest Deadline first (RBED) framework [14], which provides temporal isolation via an enforced budget associated with each task. This temporal isolation allows for mixed criticality workloads. Though RBED supports many application classes (such as Hard RT, Soft RT and Best Effort (BE) tasks), we focus in our discussion on BE and Hard RT tasks without loss of generality.

### C. Power Model

The power model used in state-of-the-art assumes two different parts: dynamic (active) power and static (leakage) power. Dynamic power consumption varies with the frequency of the processor, while static power consumption is considered as a constant factor. Consequently, such power model assumes the energy consumption of an application on a processor is only a function of its execution time. However, in real terms, energy consumption on a certain processor depends also on the set of instructions it has to execute to perform the desired functionality. Different instructions use different parts of CPU, and hence may result in a different energy consumption. Therefore, two applications with identical execution time may consume different energy depending on the characteristics of the instructions used, and the number of cache misses involved. Secondly, the static power consumption of the system cannot be regarded as a constant factor. If the energy saving mechanism is based on sleep states then the static power consumption of the system depends on the energy characteristics of the used sleep states. We employ this more refined power model where energy consumption of a system is not constant per unit time, rather depends on the behaviour of the application, the sleep-states characteristics of the processor and the use of sleep states by the scheduling algorithm.

We assume only a single speed per core (i.e. no DVS), as DVS would add another dimension for optimisation and is hence avoided due to separate concerns. The power consumption of the processor type  $\pi^m$  in active mode and idle mode are  $P_a^m$  and  $P_i^m$  respectively. Similarly, we assume each processor has  $N$  sleep states (low power states). Each sleep state  $S_n^m$  is characterised with the tuple  $\langle P_n^m, tr_n^m, Es_n^m \rangle$ , where  $P_n^m$  is the power consumption of the system in the sleep state  $S_n^m$ ,  $tr_n^m$  is the transition overhead of going into or out of sleep state and  $Es_n^m$  is the energy overhead associated to each sleep transition. For brevity, it is assumed that the transition overhead of going into or out of sleep state is same i.e.  $tr_n^m$ . The break-even-time  $BET_n^m$  of the sleep state  $S_n^m$  describes the minimum interval for which entering a given sleep state is more efficient than any shallower sleep state, despite the extra overhead (time/energy) of entering and leaving this sleep state. The sleeps states parameters can be used to derive its break-even-time  $BET_n^m$  using any known techniques [2]. Note that the  $BET_n^m$  for practical consideration is atleast  $2 \times tr_n^m$ .

The average energy consumption of all tasks on all processor types is determined offline using any known techniques (for instance, energy measurement technique based on performance monitoring counter [15]). Nevertheless, one can also use our approach with the naïve power model that assumes in active mode, the energy consumption of the processor is

constant per unit time or consider worst case energy consumption as optimisation target. The preference of the task to any processor is set with respect to its ascending order of energy consumption. The most favourite processor type for a task is the one where its energy consumption is minimal. Similarly, a processor type is least preferred where the energy consumption of a task is maximal. We assume the static power consumption of the system is not constant. It can be reduced by using efficient low power sleep states in the idle intervals.

### D. Problem Statement

We consider M-type Heterogeneous platform with per core several sleep states assuming their energy/time overhead in a setting of partitioned scheduling and map a given task-set onto this platform such that the overall energy consumption (active + sleep) of the system is minimised.

## IV. ALLOCATION HEURISTICS

In order to tackle active and static power consumption, a two phase algorithm is proposed to perform the task assignment for the given M-type heterogeneous platform. The first phase of the algorithm optimises the assignment such that it reduces the active energy consumption of the system. The second phase trades tasks active energy consumption to enhance the ability of the processors to use efficient sleep states to reduce static power consumption of the system. We will use the terms processor type, core type and core interchangeably.

### A. First Phase of Allocation

We propose two different assignment algorithms to reduce the dynamic power consumption of the system.

1) **Least Loss Energy Density Algorithm (LLED)**: This algorithm attempts to allocate tasks to their favourite core to optimise the individual task energy consumption of the system. However, not all tasks may be allocated to their respective favourite core type due to the limited capacity on each core. In such a scenario, where more than one task are competing for their favourite core type, we need to rank the tasks among each other on same core type.

We defined the energy density  $ED_i^m \stackrel{def}{=} \bar{E}_i^m / T_i$  of a task  $\tau_i$  on a core  $\pi^m$ . The energy density of a task gives its average energy consumption per unit time on the respective core type. This value does not provide any global perspective on how the power consumption of the system changes when a certain task is not allocated to its preferred core type. The global perspective can be achieved through a metric termed as density difference (DD). The density difference can be determined by subtracting the energy density of a task on the current core type from the next higher energy density value of the same task on another core. It can be computed with the following expression  $DD_i^m \stackrel{def}{=} \min\{ED_i^k : k \neq m \wedge ED_i^k \geq ED_i^m\} - ED_i^m$ . It defines how much extra energy will be consumed, if the task is allocated to the next higher energy consumption core instead of its current preferred core type. To get the ranking of the tasks on the given core, we sort all the tasks on this core in descending order with respect to their DD values. The tasks from the top of the list i.e. tasks with higher DD values

are allocated first. The intuition behind such a mechanism is to reduce the losses by allocating the tasks with higher energy density difference first. The process can be started from any core type. A task allocated to a core is not considered for an allocation on any other core where it consumes more energy than its currently allocated core. The same procedure is repeated for all cores. In the worst-case scenario, the process is iterated over each core at most  $\ell$  times.

The pseudo-code of Least Loss Energy Density algorithm (LLED) is given in Algorithm 1. Initially, we compute the energy density  $ED_i^m$  of every task on all core types (line 2). Using energy density values, the DD values of all tasks are estimated on each core and stored in a matrix called  $MT$  (line 3-6, 10). (Note:  $MT_w^q$  value in a matrix  $MT$  corresponds to the DD value of  $\tau_w$  on a core type  $\pi^q$ ). To obtain the DD value of the task  $\tau_w$  on its least preferred core type ( $\max_{x=1,\dots,M} ED_w^x$ ), its energy density value on the least preferred core type is subtracted from 0 (line 8) to obtain a negative value. Afterwards, the algorithm iterates through the processors in any order (for example, we used processors indices to order them). Starting from the first core type  $\pi^q$ , all tasks on  $\pi^q$  have their entries in  $MT^q$  sorted in descending order with respect to their  $MT_w^q$  or DD values. Our algorithm iterates by picking a task from the top of the sorted list and attempts to allocate it to  $\pi^q$ . For instance,  $\tau_x$  is the current task on top of the sorted list with respect to DD values on core  $\pi^q$ . The algorithm attempts to allocate  $\tau_x$  to  $\pi^q$ . If  $\pi^q$  can accommodate  $\tau_x$  (line 17-18), it does not consider  $\tau_x$  on other cores  $\pi^m$  for which this inequality  $\bar{E}_x^m \geq \bar{E}_x^q$  holds and removes its entries of DD values in  $MT$  matrix (line 19). In other words,  $\tau_x$  is not considered for allocation on other core types where it consumes more or equal energy compared to this core type  $\pi^q$ . If the task  $\tau_x$  was previously allocated to these higher energy consuming core types, it is deallocated on such cores (line 20). Once the allocation for  $\tau_x$  is completed on  $\pi^q$ , LLED attempts to allocate the next task in the sorted list. If any of the task in the order cannot be allocated to  $\pi^q$ , the algorithm moves to the next core type instead of checking the next tasks in the order. This action is performed to avoid allocation of any unfavourable task to the current core type, which may have a chance of allocation in the next iteration. The same procedure is repeated for the next core type and so on. On completion of the first iteration, the algorithm starts again from the first processor type. These iterations are repeated unless all the tasks are allocated to exactly one core type. In worst-case, the algorithm has to check each task in each core type for  $\ell$  times. Lines 13–26 in Algorithm 1 corresponds to these steps. Therefore, complexity of this algorithm is  $O(\ell^2 \times M)$ . The working of the algorithm is demonstrated with an example.

a) *Example:* We consider a set of 4 tasks and 3 core types. The tasks specifications are given in Figure 1(a). Entries under each core type specifies  $C_i^m, \bar{E}_i^m, ED_i^m$  for  $\tau_i$ . The DD values are computed for all tasks and presented in Figure 1(b). As an example, the DD value of  $\tau_1$  in  $\pi^1$  is computed by an expression  $ED_1^2 - ED_1^1$ . We start from the first core

---

### Algorithm 1 First Phase: Least Loss Energy Density (LLED)

---

```

1:  $U^m = 0$  for each core  $\pi^m$ 
2: Compute  $ED_i^m$  for each  $\tau_i$  on each core
3: for  $q = 1$  to  $M$  do {/* For all processor types */}
4:   for  $w = 1$  to  $\ell$  do {/* For all tasks */}
5:     if  $ED_w^q \neq \max_{x=1,\dots,M} ED_w^x$  then
6:        $ED_w^r = \min_{x=\{1,\dots,M\} \setminus q \&\& ED_w^x \geq ED_w^q} ED_w^x$ 
7:     else
8:        $ED_w^r = 0$ 
9:     end if
10:     $MT_w^q = ED_w^r - ED_w^q$ 
11:  end for
12: end for
13: for all Tasks  $\ell$  do
14:   for  $q = 1$  to  $M$  do {/* For all processors types */}
15:     Sort all tasks having entry in  $MT^q$ , w.r.t  $MT_w^q$  values in descending order
16:     for all  $\tau_w \in \tau$  on core type  $q$  in descending order of  $M_w^q$  values do
17:       if  $U^q + U_w^q \leq 1$  then
18:         Assign  $\tau_w$  to  $\pi^q$ 
19:          $\forall_{x \in \{1,\dots,M\} \setminus q}$  remove  $MT_w^x$  iff ( $\bar{E}_w^x \geq \bar{E}_w^q$ )
20:          $\forall_{x \in \{1,\dots,M\} \setminus q}$   $U^x - = U_w^x$  iff ( $\bar{E}_w^x \geq \bar{E}_w^q \&\& \tau_w$  is assigned)
21:       else
22:         Break;
23:       end if
24:     end for
25:   end for
26: end for

```

---

type  $\pi^1$  and sort the tasks in descending order of DD values as presented in the first column of Figure 1(c).  $\tau_4$  can be allocated to  $\pi^1$ , therefore, its entry that consumes more energy compared to this core type is deleted in  $\pi^3$  type.  $\tau_2$  cannot be allocated, therefore we move to  $\pi^2$  and sort the task-set according. In core type  $\pi^2$ ,  $\tau_1$  and  $\tau_4$  can be allocated.  $\tau_1$ 's entry in  $\pi^3$  and  $\tau_4$ 's entries on  $\pi^1 \& \pi^3$  will be deleted due to higher energy consumption. Similarly, after appropriate sorting of tasks with respect to their DD values on  $\pi^3$ ,  $\tau_2$  and  $\tau_3$  can be allocated to  $\pi^3$ . Therefore,  $\tau_2$ 's entry in  $\pi^2$  and  $\tau_3$ 's entry in  $\pi^2, \pi^1$  are deleted. This completes our first iteration and status of the tasks after first iteration are shown in Figure 1(c). Similarly, we perform the second iteration. On  $\pi^1$ , the  $\tau_4$ 's entry is deleted, so it is not considered for allocation and the system attempts to allocate the next task in the order (i.e.  $\tau_2$ ). The rest of the process is similar to the first iteration. The end result of 2<sup>nd</sup> iteration is shown in Figure 1(d). We do not need any further iterations as all the tasks are assigned. The worst-case number of iterations is equal to a task-set size.

2) *MaxMin Algorithm (MM):* Another simple heuristic MaxMin labelled as MM can be used to assign tasks in M-type heterogeneous platform to reduce the active power consumption is given in Algorithm 2. Assume,  $ED_i^{min}$  is the energy density of task  $\tau_i$  on its most favourite core type, while  $ED_i^{max}$  corresponds to its energy density on the least preferred core type. This heuristic for each task computes the difference of  $ED_i^{max}$  and  $ED_i^{min}$ , i.e.  $ED_i^{max} - ED_i^{min}$ . All tasks are globally sorted in descending order with respect to

Fig. 1: First Phase Mapping of Least Loss Energy Density Algorithm

(a) $C_i^m/\bar{E}_i^m/ED_i^m$ Values					(b) Density Difference (DD) in $MT$			(c) $1^{st}$ Iteration			(d) $2^{nd}$ Iteration			
	$\pi^1$	$\pi^2$	$\pi^3$	$T_i$		$\pi^1$	$\pi^2$	$\pi^3$	$\pi^1$	$\pi^2$	$\pi^3$	$\pi^1$	$\pi^2$	$\pi^3$
$\tau_1$	4.5/16.5/1.65	3/17.2/1.72	7/52.5/5.25	10	$\tau_1$	0.07	3.53	-5.25	$\bar{\tau}_4$	$\tau_1$	$\tau_2$	$\bar{\tau}_4$	$\bar{\tau}_1$	$\bar{\tau}_2$
$\tau_2$	8/37.65/2.51	10/65.1/4.34	8/57/3.80	15	$\tau_2$	1.29	-4.34	0.54	$\tau_2$	$\tau_4$	$\tau_3$	$\tau_2$	$\tau_4$	$\tau_3$
$\tau_3$	18/84/2.80	12/78.9/2.63	10/75.9/2.53	30	$\tau_3$	-2.8	0.17	0.10	$\tau_1$	$\bar{\tau}_3$	$\bar{\tau}_1$	$\tau_1$	$\bar{\tau}_3$	$\bar{\tau}_1$
$\tau_4$	60/259.2/2.16	35/210/1.75	80/649.2/5.41	120	$\tau_4$	3.25	0.41	-5.41	$\bar{\tau}_3$	$\bar{\tau}_2$	$\bar{\tau}_4$	$\bar{\tau}_3$	$\bar{\tau}_2$	$\bar{\tau}_4$

**Algorithm 2** Alternative First Phase: MaxMin (MM)

- 1:  $U^m = 0$  for each core  $\pi^m$
- 2: Compute  $ED_i^m$  for each  $\tau_i$  on each core
- 3:  $\forall \tau_i$  : Find  $ED_i^{max} = \max_{x=1,\dots,M} ED_w^x$
- 4:  $\forall \tau_i$  : Find  $ED_i^{min} = \min_{x=1,\dots,M} ED_w^x$
- 5: Sort task-set with respect to  $(ED_i^{max} - ED_i^{min})$  in descending order
- 6: **for all** Tasks  $i = 1$  to  $\ell$  **do**
- 7:     Sort cores with respect to the energy consumption of  $\tau_i$  in ascending order
- 8:     **for all** Processors  $j = 1$  to  $M$  **do**
- 9:         **if**  $U^j + U_i^j \leq 1$  **then**
- 10:             Assign  $\tau_i$  to  $\pi^j$
- 11:              $U^j + U_i^j = U_i^j$
- 12:             Break
- 13:         **end if**
- 14:     **end for**
- 15: **end for**

this difference (line 5). The MM algorithm picks a task from the top of the list and assigns to its favourite core type. If the favourite core cannot accommodate this task, an allocation attempt is made on the next core type in its ascending order of energy consumption (line 8-14). If the task is assigned to a core type, the utilisation of the corresponding core type is incremented accordingly. The MaxMin algorithm is simple and has a complexity of  $O(\ell \times M)$ .

*B. Second Phase of Optimisation*

While, the first phase of allocation is derived with an objective to optimise an individual task's active energy consumption in the system, it ignores its effect on the mechanism to reduce the static power consumption. For instance, a core may have less active energy consumption but some small group of tasks allocated to it may prevent it from using a more efficient deeper sleep state in the idle intervals of the schedule to reduce the static power consumption of the system. In this second phase of optimisation, our algorithm analyses the properties of the allocated tasks to a core in this broader context and considers its effect on the core's ability to use more efficient sleep states by trading off higher active energy consumption of a task for energy savings in sleep states.

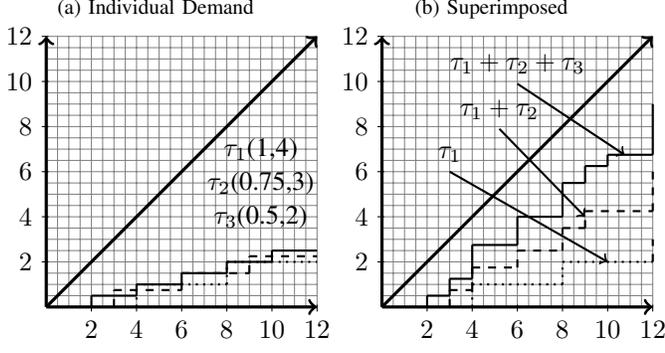
As mentioned previously, we assume ERTH per core. The ERTH scheduler is based on a race-to-halt strategy and reduces static power consumption with a shut-down mechanism. It determines the maximum time interval offline for which the processor may be enforced in a sleep state without causing any task to miss its deadline under worst-case assumptions. This maximum time interval of a sleep state is termed as

maximum-feasible-sleep-threshold  $th^m$  and it can be determined using the demand bound function (DBF) [2]. Assuming synchronous release of all tasks allocated to a core  $\pi^m$ ,  $th^m \stackrel{def}{=} \min_{\forall L \leq L^*} (L - dbf(L))$ , where  $L$  is an absolute deadline and  $L^*$  is the first idle time in the schedule. In other words,  $th^m$  is the minimum distance between the supply and the request bound functions in the first busy interval, assuming the synchronous release of all tasks allocated to  $\pi^m$ . ERTH initiates a sleep transition online when the system is idle or has sufficient slack. The length of  $th^m$  defines which sleep state can be used online on core  $\pi^m$ . For instance, if a sleep state  $S_n^m$  has a break-even-time  $BET_n^m > th^m$ , it is not beneficial (energy-wise) to use such sleep state, as a sleep state saves energy when used for greater than  $BET_n^m$  time interval.

ERTH selects a sleep state for a core  $\pi^m$  to be used online based on the value of  $th^m$ . However, the properties of tasks involved in the computation of  $th^m$  have a high impact on its value. For example, tasks with shorter difference between their  $T_i$  and  $C_i^m$  give a small value of  $th^m$  and restrict usage of those sleep states with  $BET_n^m > T_i - C_i^m$ . The intuition behind the second phase is to collate tasks on a core with similar properties such that it can use a more efficient sleep state. As we are using a heterogeneous platform, each core has sleep states with different characteristics. A task(s) restricting a more efficient sleep state on one core may not effect the sleep state on the other core and hence can be considered for migration. However, the algorithm must ensure that such migration reduces the overall average energy consumption.

We propose the heuristics given in Algorithm 3 to do such a trade-off. Tasks assigned in the first phase are sorted in each core with respect to their difference between  $T_i$  and  $C_i^m$  in descending order. Consider one of the core type  $\pi^m$  and assume  $\ell^m$  are the number of tasks allocated to it in the first phase (through LLED or MM). The second phase initially computes the maximum time interval of the sleep duration also known as a maximum-feasible-sleep-threshold with just one task picked from the top of the sorted list (w.r.t  $T_i - C_i^m$ ) of tasks allocated to  $\pi^m$ . This value is denoted as  $th_1^m$  and computed through DBF. As there is just one task, therefore,  $th_1^m = T_i - C_i^m$ . Now we superimpose the next task on the current DBF and new maximum-feasible-sleep-threshold  $th_2^m = \min_{\forall L \leq L^*} (L - dbf(L))$  is computed. Similarly, a third task is superimposed and correspondingly  $th_3^m$  is computed. This process is repeated for all sorted tasks allocated to  $\pi^m$  and at the end we have a set of maximum-feasible-sleep-threshold values called  $\rho^m = \{th_1^m, th_2^m, th_3^m, \dots, th_{\ell^m}^m\}$ . As the tasks are superimposed in the descending order of  $T_i - C_i^m$ , therefore, one of the property of  $\rho^m$  is that  $th_1^m \geq th_2^m \geq th_3^m \geq \dots \geq th_{\ell^m}^m$ .

Fig. 2: Demand Bound Function



Moreover,  $th^m \stackrel{def}{=} th_{\ell^m}^m$ . To illustrate the computation of  $\rho^m$  set, let us consider an example. Assume, we have three tasks  $\tau(C_i^m, T_i) \Rightarrow \tau_1(1, 4), \tau_2(0.75, 3), \tau_3(0.5, 2)$  sorted in the descending order of  $T_i - C_i^m$  and allocated to  $\pi^m$ . Individual demands of these tasks are shown in Figure 2(a). Firstly,  $th_1^m$  with  $\tau_1$  is computed, i.e. 3 units. Then  $\tau_2$  is superimposed on  $\tau_1$  and  $th_2^m$  is computed, which is equal to 2.25. Finally,  $\tau_3$  is superimposed on the demand of  $\tau_1 + \tau_2$  and  $th_3^m$  is estimated to be 1.25. These steps are demonstrated in Figure 2(b). This example has  $\rho^m = \{3, 2.25, 1.25\}$ .

The number of elements in  $\rho^m$  is equal to  $\ell^m$ . Each element in  $\rho^m$  gives the maximum sleep interval with the corresponding number of tasks. A core  $\pi^m$  can use this sleep interval to initiate a sleep state, if the tasks used to compute such interval are allocated to it. We determine the most efficient sleep state (among the available set of sleep states in  $\pi^m$ ) for all elements of  $\rho^m$  using the following expression  $\{\forall x \in \rho^m, \text{find } S_n^m : S_n^m \text{ minimises } (x \times P_a^m + E_s^m)\}$ . We know,  $\rho^m$  holds the property that  $th_1^m \geq th_2^m \geq th_3^m \geq \dots \geq th_{\ell^m}^m$ . Therefore,  $th_2^m$  cannot get a better sleep state when compared to  $th_1^m$  and so on.

After computing the sleep states for each  $\rho^m$  element we group the tasks that allow the same sleep state. We define a set  $G_n^m$  that holds the tasks for a sleep state  $S_n^m$ . Starting from  $th_1^m$ ,  $\tau_1$  is added to a set of its computed sleep state. Similarly,  $\tau_2$  is added to the set corresponding to a sleep state determined for  $th_2^m$  and so on. Lets demonstrate this step with an example. Suppose, we have five elements in  $\rho^m = \{th_1^m, th_2^m, th_3^m, th_4^m, th_5^m\}$ . Assume, sleep states corresponding to these  $\rho^m$  elements are determined to be  $\{S_1^m, S_1^m, S_2^m, S_3^m, S_3^m\}$ . Then the tasks are added to the sets corresponding to the sleep states as follow:  $G_1^m = \{\tau_1, \tau_2\}$ ,  $G_2^m = \{\tau_3\}$  and  $G_3^m = \{\tau_4, \tau_5\}$ . We refer to these sets as groups of tasks corresponding to different sleep states. These groups of tasks are ordered from the least efficient to the most efficient sleep states. Thus, if we remove the top most group of tasks, a core can achieve the next better sleep state. This complete process is repeated for all cores and finally we have different groups of tasks on each core corresponding to its different sleep states. This step is given in line 4 (Algorithm 3).

All cores compete to gain the next more efficient sleep state to save energy by getting rid of their tasks in the top most group that enforces the less efficient sleep state. However, the

---

### Algorithm 3 Second Phase of Task Mapping (SP)

---

```

1: repeat
2:   Previous Assignment = Current Assignment
3:   Energy Old = Energy New
4:   Group tasks per core such that next better sleep state can be
   achieved
5:   Order core by gains when removing group
6:   Feasible = TRUE
7:   for all Processor Types  $M$  do
8:     for all Tasks in a top group do
9:       Compute the local cost of migration on energy consump-
10:      tion of this task for all other cores
11:      Sort other cores by decreasing order of cost
12:      for all Cores except the core of the currently assigned
13:      task do
14:        if Feasible on core then
15:          Assign to a core
16:          Success = True
17:          Break
18:        end if
19:      end for
20:      if !Success then
21:        Feasible = FALSE
22:        Break
23:      end if
24:    end for
25:    if Feasible && Energy New < Energy Old then
26:      Break
27:    else
28:      Undo all Assignments
29:    end if
30:  end for
31: until Previous Assignment == Current Assignment

```

---

algorithm will first consider the core which would result in the most system energy gain. To identify this core, each core will remove all the tasks associated to the first group (that cause less efficient sleep state). Let  $G_{top}^m$  corresponds to the tasks in the top least efficient sleep state group on  $\pi^m$ . The energy saving by removing such group from this core will be equal to  $\Delta \bar{E}^m$  as given in Equation 1. Where  $th_{old}^m$  and  $th_{new}^m$  are the maximum-feasible-sleep-threshold intervals before and after removing  $G_{top}^m$  respectively on  $\pi^m$ . After computing  $th_{old}^m$  and  $th_{new}^m$ , their corresponding sleep states are determined. Suppose,  $S_{n1}^m$  and  $S_{n2}^m$  are the sleep states selected for  $th_{old}^m$  and  $th_{new}^m$  respectively. Moreover,  $\bar{E}_{old}^m = (th_{old}^m - tr_{n1}^m) \times P_{n1}^m + E_{s_{n1}}^m$  and  $\bar{E}_{new}^m = (th_{new}^m - tr_{n2}^m) \times P_{n2}^m + E_{s_{n2}}^m$  represent the energy consumption of a single sleep transition with and without  $G_{top}^m$ . All Cores are sorted in descending order with respect to  $\Delta \bar{E}^m$  as given in line 5 of Algorithm 3, which will be used to attempt a reallocation of the top most group of the cores in this determined order.

Assume  $\pi^1, \pi^2, \dots, \pi^m$  represent the cores in descending order of  $\Delta \bar{E}^m$ . Initially, we select  $\pi^1$ .  $G_{top}^1$  are the tasks in the top least efficient sleep state group of  $\pi^1$ . The local cost of migration  $LC_{\tau_j}^o$  of each task  $\tau_j \in G_{top}^1$  will be computed on every other core type  $\pi^o$  excluding  $\pi^1$ . The expression to determine the local cost of migration  $LC_{\tau_j}^o$  is given in Equation 2. This value is computed by finding  $\tau_j$ 's energy density on  $\pi^o$  plus the energy consumption per unit of time in

TABLE I: Overview of Simulator Parameters

Parameters	Specifications
Task-set sizes $ T $	{100, <u>200</u> , 500}
Inter-arrival time $T_i$ for RT tasks	[ <u>30ms</u> , 50ms]
Inter-arrival time $T_i$ for BE tasks	[50ms, <u>200ms</u> ]
Sporadic delay limit $\Upsilon$	{10%}
Best-Case execution-time limit $C^b$	{10%}
Share of RT/BE tasks $\xi$	{30%, 70%}
Characteristic Factor $\beta$	{10%, <u>20%</u> , 40%}

idle period with  $\tau_j$  on  $\pi^o$  minus the energy consumption per unit of time in idle period without  $\tau_j$  on  $\pi^o$ . Where,  $th_{new}^o$  and  $th_{old}^o$  are the maximum-feasible-sleep-thresholds with and without including  $\tau_j$  on  $\pi^o$  respectively. The sleep states corresponding to  $th_{new}^o$  and  $th_{old}^o$  are determined.  $\bar{E}_{new}^o$  and  $\bar{E}_{old}^o$  are the energy consumption of a single sleep transition with or without  $\tau_j$  respectively. The algorithm sorts all the core types in ascending order of  $LC_{\tau_j}^o$  to move  $\tau_j$ .

$$\Delta \bar{E}^m = \left( \sum_{\forall \tau_i \in \pi^m} \frac{\bar{E}_i^m}{T_i} + \frac{(1 - \sum_{\forall \tau_i \in \pi^m} U_i^m) \bar{E}_{old}^m}{th_{old}^m} \right) - \left( \sum_{\forall \tau_i \in \pi^m \setminus G_{top}^m} \frac{\bar{E}_i^m}{T_i} + \frac{(1 - \sum_{\forall \tau_i \in \pi^m \setminus G_{top}^m} U_i^m) \bar{E}_{new}^m}{th_{new}^m} \right) \quad (1)$$

$$LC_{\tau_j}^o = \frac{\bar{E}_j^o}{T_j} + \frac{(1 - \sum_{\forall \tau_i \in \pi^o + \tau_j} U_i^o) \bar{E}_{new}^o}{th_{new}^o} - \frac{(1 - \sum_{\forall \tau_i \in \pi^o} U_i^o) \bar{E}_{old}^o}{th_{old}^o} \quad (2)$$

$$TE = \sum_{\forall \pi^m} \left\{ \left( \sum_{\tau_i \in \pi^m} \frac{\bar{E}_i^m}{T_i} \right) + \left( \frac{(1 - \sum_{\tau_i \in \pi^m} U_i^m) \bar{E}_{S_n}^m}{th^m} \right) \right\} \quad (3)$$

The algorithm attempts to assign it to a core type with the least migration cost provided it is schedulable on that core. This process is repeated  $\forall \tau_j \in G_{top}^1$ . Lines 8 to 22 in Algorithm 3 correspond to this step. In case any of the tasks  $\tau_j \in G_{top}^1$  is not schedulable, all the assignments are undone and we move to the next core type. On the other side, if the assignments of  $G_{top}^1$  are successful, we compute the new expected total energy  $TE$  consumption of the system with Equation 3 and compare it with the previous expected total energy consumption, where  $\bar{E}_{S_n}^m$  is the energy consumption of a sleep state  $S_n^m$  and  $S_n^m$  is the selected sleep state that consumes the least energy when initiated for  $th^m$ . If it is less than previous expected  $TE$  consumption, we iterate over the algorithm again unless the energy consumption of the previous iteration is greater than this iteration.

The maximum number of groups (of tasks) in each processor is equal to its number of sleep states and we migrate the complete group to another core. The complexity of each iteration is  $O(\ell M)$ . Theoretically, the complexity of the entire algorithm is combinatorial, as a migrant task from one core type can be reassigned to it in another iteration, but for all practical reasons it converges very quickly. The algorithm avoids already computed assignments with a constraint that new assignment should reduce the energy consumption. The actual computation time and the number of migrations are discussed in Section V-B(2).

TABLE II: Processors Power Model Parameters

$\pi^m$	$P_a^m$	$P_i^m$	$\eta^m$	$S_1^m$	$S_2^m$	$S_3^m$	$S_4^m$
$\pi^1$	1.0	0.39	1.0	0.31	0.21	0.12	0.05
$\pi^2$	2.2	0.86	0.5	0.67	0.47	0.27	0.11
$\pi^3$	6.0	2.33	0.2	1.83	1.29	0.74	0.30
$\pi^4$	13.0	5.05	0.1	3.98	2.79	1.61	0.64
$\pi^5$	12.1	4.7	0.15	3.70	2.60	1.50	0.6

## V. EVALUATION

### A. Experimental Setup

In order to evaluate the effectiveness of our algorithms, we have extended the SPARTS (Simulator for Power Aware and Real-Time Systems) [16] and implemented our algorithms for the experiments. SPARTS is used with the parameters defined in Table I. The underlined values are the default values if not specified in the description of an individual experiment. Heterogeneous multicore platforms are used for a wide variety of complex applications, therefore, the task-set size is varied from small number of coarse grained 100 tasks to fine grained large tasks-set sizes of 500 tasks. The share distributions  $\xi$  divide the task-set size and overall effective system utilisation between RT and BE tasks. Moreover, the utilisation allocated to each task type is randomly distributed among the tasks of the same class. The minimum inter-arrival time of RT and BE tasks is randomly chosen within a range of [30ms; 50ms] and [50ms; 200ms] respectively. SPARTS selects one of the core type and reference it as a default core type  $\pi^D$ . The task-set is initially generated for  $\pi^D$ . The WCET  $C_i^D$  of  $\tau_i$  is deemed to be  $U_i^D \times T_i$ , where  $U_i^D$  is the utilisation of  $\tau_i$  on  $\pi^D$ .

The average system capacity  $U_a$  of the given platform is computed through the average speed-up-factor  $\eta^m$ . The speed-up-factor defines a ratio of the clock cycle of a core  $\pi^m$  with reference to  $\pi^D$ . Suppose speed-up-factor of a core type  $\pi^m$  is  $\eta^m$ , then the average capacity of the system will be  $U_a = 1/\eta^1 + 1/\eta^2 + \dots + 1/\eta^m$ . However, the effective utilisation  $U$  of the task-set in the experiments is controlled through a helper variable  $\zeta$ , and  $U = U_a \times \zeta$ . The range of  $\zeta$  is (0; 1]. In our experiments,  $\zeta$  is varied from 0.5 to 0.9 with a step size of 0.05. Individual utilisation of  $\tau_i$  on each  $\pi^m$  is a random number within a range of  $U_i^m = [(1 - \beta); (1 + \beta)] \times \eta^m \times U_i^D$ , where  $\beta$  is a characteristic factor that models the fact that different tasks will respond differently in terms of execution time when moved from one core to another.

Beyond those initial settings, a two level approach is used for generating a wide variety of different tasks and their subsequently varying jobs on all cores. Tasks are further annotated with a limit on the sporadic delay  $\Delta_i^s$  in the interval  $[0, \Upsilon \times T_i]$  and on the best-case execution time  $C_i^b$  in the interval  $[C^b \times C_i^m, C_i^m]$ . The second level varies the behaviour of individual jobs of a task. The interested reader is referred to [16] for details. Each set points of parameters is evaluated with 100 different task sets.

The hardware parameters of heterogeneous platform used in our experiments are shown in Table II. The power model for the default core in our experiments is modelled after the FreeScale PowerQUICC III Integrated Communication Processor MPC8536 [17]. The FreeScale PowerQUICC III core specifications are given in Table II under  $m = 5$ . The

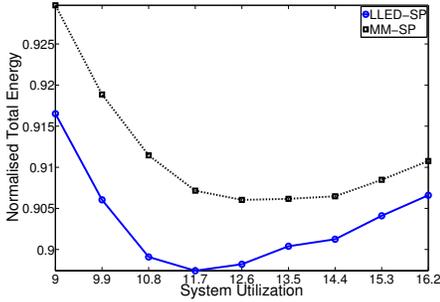


Fig. 3: 4 Core Types (S1)

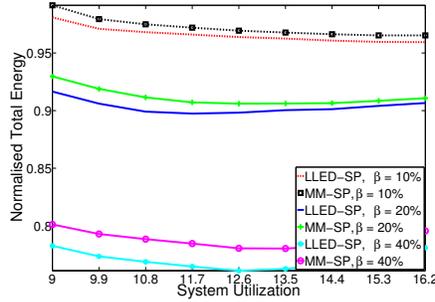


Fig. 4: Variation in  $\beta$  (S1)

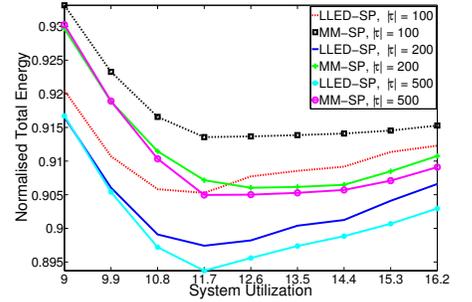


Fig. 5: Variation in Task-set Size (S1)

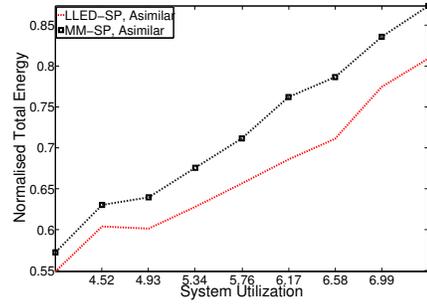


Fig. 6: Asimilar Platform (S1)

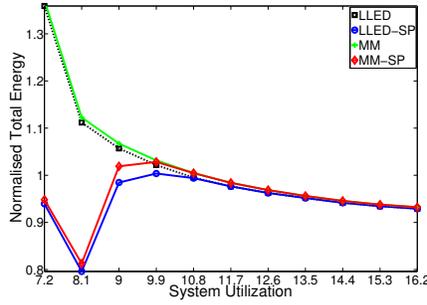


Fig. 7: 4 Core Types (S2)

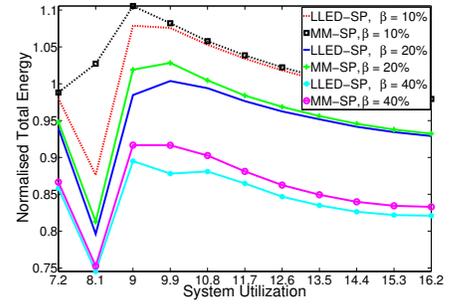


Fig. 8: Variation in  $\beta$  (S2)

values of the other core types are derived from this core type to generate a heterogeneous platform. We assume each core type has four sleep states, with  $\{S_x^m : x = 1, 2, 3, 4\}$  representing different sleep states such as Doze, Nap, Sleep and Deep Sleep respectively. We have assumed their transition overheads and estimated break-even-time accordingly. We assume a single unit of each core type. The average system capacity  $U_a = \frac{1}{1} + \frac{1}{0.5} + \frac{1}{0.2} + \frac{1}{0.1} = 18$ . As we are changing  $\zeta$  in an interval of  $[0.5; 0.9]$ , therefore, the effective utilisation of the system  $U$  is within a range of  $[0.5; 0.9] \times 18 = [9; 16.2]$ . The energy consumption of a task is, however, not a mere function of its execution time. As such the values of  $\bar{E}_i^m$  are computed using the average execution time  $\bar{C}_i^m$  and a random value similar to the utilisation conversion  $\bar{E}_i^m = [1 - \beta; 1 + \beta] \times P_a^m \times \bar{C}_i^m$ .

## B. Results

The parameters described previously remain the same, except where explicitly specified. In the state-of-the-art there is no such algorithm proposed that has a power model such that this work could be compared with it. Moreover, fundamental assumptions made in the state-of-the-art restrict their extension to the more realistic power model proposed in this paper. Therefore, we have implemented a worst-fit decreasing (*WFD*) and first-fit (*FF*) algorithm as a base line to compared against our algorithms. It has been shown by Aydin and Yang [18] that *WFD* performs better when compared to other conventional bin packing algorithms for homogeneous platforms. In our experiments, we observed that *WFD* performs worst in heterogeneous platforms. It was able to schedule few tasks-set at higher utilisations making it hard to compare against our algorithms. Therefore, we use only the *FF* algorithm for the comparison. The experiments of *WFD* are omitted in this paper but these results are available in a

technical report [19] for the interested readers. Moreover, the *FF* algorithm allocates the tasks sorted with respect to their  $D_i$  or  $T_i$  following the order from the slowest core type to the fastest core type. The results under labels *LLED-SP* and *MM-SP* represent the second phase applied on the allocation of *LLED* and *MM* respectively. We have created 2 different scenarios. In the first scenario, we have modelled the system with very efficient sleep states having low transition overhead (time and energy). The second scenario models the system, with substantially less efficient sleep states. All results are normalised to the corresponding values of the *FF* algorithm.

1) *First Scenario*: In this scenario, as the overheads of the sleep state is low, therefore, different cores can still achieve the most efficient sleep state even at high utilisation. This scenario does not leave much room for the second phase to save any additional energy when compared to *LLED*. Nevertheless, *MM-SP* saves in some cases energy over *MM* but it is fairly minimal. Therefore, for this scenario, we compare the energy consumption of *MM-SP* and *LLED-SP*.

Firstly, the performance of *LLED-SP* and *MM-SP* is analysed for different number of core types. Figure 3 shows the normalised energy consumption of the system with 4 core types. The figure for 2 cores looks similar to Figure 3 but does not provide as high energy gains over *FF* due to the limited scope for optimisation. Similarly, in the second case of 4 core types, initially, the difference of *LLED-SP* and *MM-SP* increases but then starts to shrink towards the higher utilisations. This behaviour is obvious as *LLED-SP* and *MM-SP* have more chance at low utilisation to allocate task to their favourite core. However, towards high utilisations, this flexibility decreases along with their difference. In best-case, *LLED-SP* consumes 10% less energy when compared to *FF*, while *MM-SP* saves energy slightly under 10%.

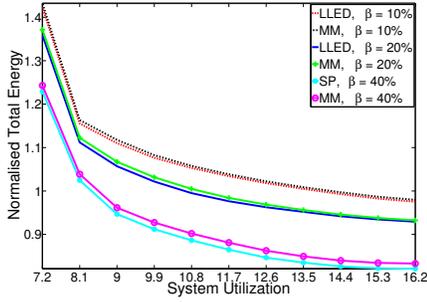


Fig. 9: Variation in  $\beta$  (S2)

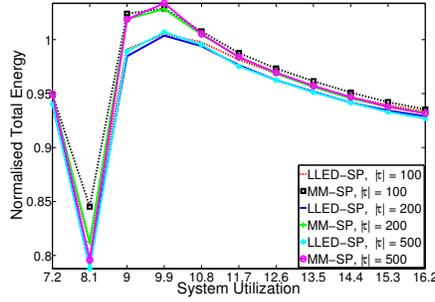


Fig. 10: Variation in Task-set Size (S2)

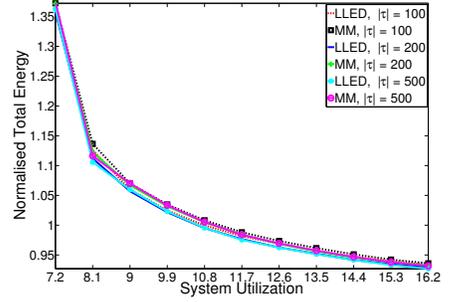


Fig. 11: Variation in Task-set Size (S2)

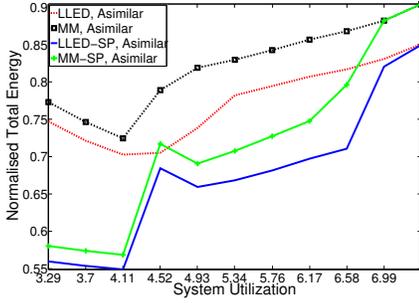


Fig. 12: Asimilar (S2)

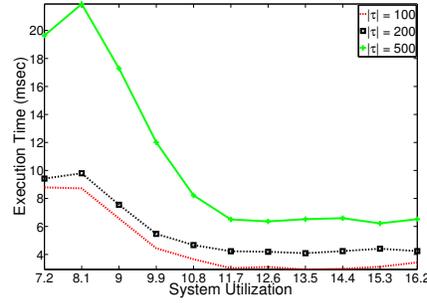


Fig. 13: Time Calculation (S2)

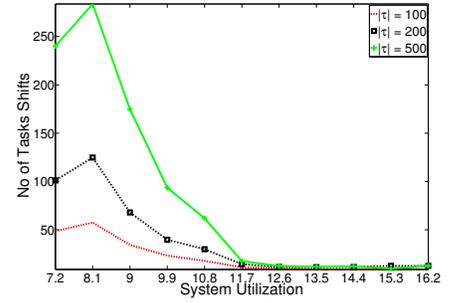


Fig. 14: Decisions (S2)

We evaluate the effect of variation in the characteristic factor  $\beta$  on the normalised total energy consumption of the system.  $\beta$  controls the variation of task dynamic power consumption from the average dynamic power consumption of the core. Figure 4 demonstrates that the energy consumption of both approaches decreases with an increase in the range of  $\beta$ . The developed power model on average favours the slow core. However, this factor ( $\beta$ ) can change this behaviour. With  $\beta = 10\%$ , small portion of tasks are more favourable to the fast cores. Hence, the *FF* algorithm that fills the slowest core first does a few task allocation to their unfavourable cores. Consequently, the gains of *LLED-SP* and *MM-SP* are less at  $\beta = 10\%$ . However, as the  $\beta$  range increases, the tasks probability to favour a fast core becomes higher. Therefore, *LLED-SP* and *MM-SP* give better allocations for higher values of  $\beta$ . Similar to the previous observation, the difference of *MM-SP* and *LLED-SP* is higher at low utilisation and decreases with an increase in the system utilisation.

Figure 5 demonstrates the effect of task-set size variation on the given allocation mechanism. In general a large task-set size increases the probability of the tasks to be allocated to their unfavourable core with *FF*. Therefore, energy consumption of the *LLED-SP* and *MM-SP* algorithms decreases with an increase in the task-set size. However, this saving reduces with an increase in the effective utilisation. In the beginning *LLED-SP* with the different task-set sizes do the same allocation but with an increase in effective system utilisation, the difference in allocation also increases. The same observations hold for the *MM-SP* as well. For small task-set size of 100, *FF* also performs well at low utilisation. However, this effect deteriorates with an increase in the effective system utilisation.

Processor types given in Table II have approximately similar ratio of  $P_a^x/P_a^y \approx \zeta^y/\zeta^x$ . We have generated a case where this

ratio is not the same and tasks always favour the same core i.e.  $P_a^x/P_a^y \neq \zeta^y/\zeta^x$ . This case allows us to evaluate a system, where all the tasks are competing for the best core types. For this experiment, we modified the heterogeneous platform given in Table II and generated an asimilar heterogeneous platform by changing the  $\eta^m$  values from 1, 0.5, 0.2, 0.1 to 1, 0.6, 0.45, 0.3. The average capacity of the asimilar platform is  $U_a = \frac{1}{1} + \frac{1}{0.6} + \frac{1}{0.45} + \frac{1}{0.3} = 8.22$ . The effective utilisation  $U$  is varied within a range of  $[0.5; 0.9] \times 8.22 = [4.11; 7.4]$ . Figure 6 presents the results for the asimilar platform. The energy consumption of *LLED-SP* and *MM-SP* is low at low utilisation and gradually increases towards high utilisation. All the algorithms attempt to allocate tasks in order from the slowest core to the fastest core. *LLED-SP* can rank tasks in an efficient way and saves more energy. Similarly, *MM-SP* also performs better when compared to *FF* as it also does some ranking of the tasks but *FF* does not prioritise the tasks to account for global energy benefits.

2) *Scenario 2*: In this scenario, we have modelled a system, in which the core types have large overheads of sleep transitions (time/energy). To generate such model, we have scaled the transition delays of all the sleep states by a factor of 12 and determined their BET accordingly. We have observed a very interesting result, which shows, it is not necessary that tasks assigned to their favourite core will always reduce the overall system energy consumption of the system. In this scenario, the overall energy consumption depends mostly on the characteristics of the core and it depends less on those of the tasks. This fact will be evident in the following experiments, in which we are comparing *LLED*, *MM*, *LLED-SP* and *MM-SP*. The base line is still the corresponding energy consumption of *FF*. Furthermore, the range of  $\zeta$  is increased to  $[0.4; 0.9]$  with a step size of 0.05 for this scenario.

Figure 7 shows the normalised total energy consumption of system for 4 core types. At low utilisation, though *LLED* and *MM* had a chance to allocate tasks to their favourite core but globally it is not energy efficient. The reason is that these algorithms are not accounting the effect of their allocation on the core sleep states. The *FF* algorithm which is also sleep state agnostic allocation mechanism surprisingly performs well compared to *LLED* and *MM*. It allocates the core from the slowest one and allows fast core to have empty space to use their efficient sleep state. However, our *LLED-SP* and *MM-SP* algorithms compare well to *FF* at low utilisations and compensate for the wrong allocation done by *LLED* and *MM* respectively. It is interesting to see that for low utilisations *LLED-SP* and *MM-SP* achieve substantial gains. For high utilisations, *LLED* and *MM* energy consumption reduces when compared to *FF*. Hence, a combination of initial first phase allocation (*LLED* or *MM*) with the second phase is a good choice for most of the system utilisations, except for some corner cases (at a utilisation of 9.9 in Figure 7). In the detailed analysis of utilisations between 7.2 and 9, we have observed that *FF* loses the efficient sleep states earlier than *LLED-SP* or *MM-SP*. Hence, the energy consumption of *LLED-SP* and *MM-SP* is dropped at  $U = 8.1$  when compared to *FF*. Figure 7 shows that the performance of the *LLED* algorithm is always dominant over the *MM* algorithm, and similarly, the performance of *LLED-SP* over *MM-SP*.

The variation in the characteristics factor  $\beta$  is demonstrated in Figure 8 and Figure 9. Similar to the results in scenario 1 (Figure 4), the performance of *LLED-SP* and *MM-SP* given in Figure 8 increases with an increase in the value of  $\beta$  and the similar trend is followed by *LLED* and *MM* in Figure 9. Figure 8 also shows that *LLED-SP* always dominates *MM-SP* and the same is true in Figure 9 for *LLED* and *MM*. The effect of variation in the task-set size is presented in Figure 10 and Figure 11. Unlike to Figure 5, in this scenario the task-set size does not make any difference on the performance of all the algorithms. To evaluate the platform, where all the tasks prefer similar core type, the same setup of Figure 6 is adopted. The results of this experiment are shown in Figure 12. All the algorithms follow the same race to allocate tasks to the slowest core. Furthermore, *LLED* performance dominated over *MM*, and towards high utilisations, it even consumes less energy compared to *MM-SP*. Overall, *LLED-SP* performs better for all utilisations. Figure 13 and Figure 14 present the execution times and the number of tasks migrations between different core types of the second phase of allocation respectively. To generate the results in Figure 13, we used a server with 8 Intel Xenon 1.60GHz processors and a memory size of 8GB. The allocation process of the second phase is very fast even for a large task-set size of 500. Figure 14 shows that the number of migrations (also execution time) decrease with an increase in effective utilisation as the tasks have less freedom to manoeuvre due to high utilisations. Less loaded systems ( $U = 7.2$ ) allow cores to use their more efficient sleep anyway. Therefore,  $U = 7.2$  has fewer number of migrations (executions time) when compared to  $U = 8.1$ .

## VI. CONCLUSION

Heterogeneous multicore platforms are becoming popular in industry. This trend demands an advancement in RT scheduling theory. We have explored the problem of task assignment with an objective to reduce the average-case energy consumption of the system, while satisfying RT constraints. This research effort demonstrates the importance of a realistic power model and its effect on the overall energy consumption. In the future, we have an intention to relax the assumption of single processing unit of each processor type and extend this work to allow the job migration to further reduce the energy consumption of the system.

## REFERENCES

- [1] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate on-line prediction of processor and memory energy usage under voltage scaling," in 7th EMSOFT, Salzburg, Austria, Oct 2007, pp. 84–93.
- [2] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in 23rd ECRTS, 2011, pp. 92–101.
- [3] A. Kandhalu, J. Kim, K. Lakshmanan, and R. Rajkumar, "Energy-aware partitioned fixed-priority scheduling for chip multi-processors," in 17th RTCSA, vol. 1, aug. 2011, pp. 93–102.
- [4] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in 13th RTCSA, aug. 2007, pp. 28–38.
- [5] Y. Yu and V. Prasanna, "Power-aware resource allocation for independent tasks in heterogeneous real-time systems," in Parallel and Distributed Systems, 2002, pp. 341–348.
- [6] T.-Y. Huang, Y.-C. Tsai, and E.-H. Chu, "A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem," in IPDPS, 2007, pp. 1–10.
- [7] J. Luo and N. K. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," in ASP-DAC, 2002.
- [8] J.-J. Chen and L. Thiele, "Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements," in 14th ICPADS, dec. 2008, pp. 161–168.
- [9] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo, "Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint," in 43rd DATE, 2006.
- [10] C.-M. Hung, J.-J. Chen, and T.-W. Kuo, "Energy-efficient real-time task scheduling for a dvs system with a non-dvs processing element," in 27th RTSS, 2006.
- [11] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in 46th DATE, 2009, pp. 694–699.
- [12] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in IPDPS, 2009, pp. 1–12.
- [13] G. B. Dantzig and M. N. Thapa, "Linear programming: 1: Introduction," in Springer Verlag, 1997.
- [14] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in 24th RTSS, Cancun, Mexico, Dec 2003, p. 396.
- [15] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, "Koala: A platform for OS-level power management," in 4th EuroSys Conf., Nuremberg, Germany, Apr 2009.
- [16] B. Nikolic, M. A. Awan, and S. M. Petters, "SPARTS: Simulator for power aware and real-time systems," in 8th IEEE Int. Conf. Emb. Softw. & Syst. Changsha, China: IEEE, Nov 2011, pp. 999–1004.
- [17] MPC8536E PowerQUICC III Integrated Processor Hardware Specifications, FreeScale Semiconductor, document Number: MPC8536EEC, Rev 3, Nov. 2010. [Online]. Available: [http://cache.freescale.com/files/32bit/doc/data\\_sheet/MPC8536EEC.pdf?fsp=1](http://cache.freescale.com/files/32bit/doc/data_sheet/MPC8536EEC.pdf?fsp=1)
- [18] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in IPDPS 2003, april 2003.
- [19] M. A. Awan and S. M. Petters, "Energy-conscious tasks partitioning onto a heterogeneous multi-core platform," Technical Report: 2012, <https://www.cister.isep.ipp.pt/people/Muhammad%2BAli%2BAwan/publications/>.