Technical Report

Energy-conscious tasks partitioning onto a heterogeneous multi-core platform

Muhammad Ali Awan

Stefan M. Petters

HURRAY-TR-121006

Version:

Date: 10/18/2012

Energy-conscious tasks partitioning onto a heterogeneous multi-core platform

Muhammad Ali Awan, Stefan M. Petters

IPP-HURRAY!

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, smp@isep.ipp.pt

http://www.hurray.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 very 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-conscious tasks partitioning 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 very 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 multiprocessors. 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, ERTH [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 steps. Firstly, the novel algorithm performs assignments with an objective to reduce the active energy consumption of the system by allocating tasks to their favourite processors, on which their active energy consumption is minimal compared to 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 stage 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 is an orthogonal issue as it depends on the properties of the tasks such as its minimum inter-arrival time and worst-case execution time. For instance, assume the task assignment is such that it generates large amount of idle intervals. 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.

The paper is organised as follows. Section II discusses the related work followed by the system model. Section IV presents the two phase approach to do the task assignment followed by the experimental setup and results. 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, Kandhalue et al. [3] recently presented a Single-clock domain multi-processor Frequency Assignment Algorithm (SFAA) for the 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 tasks 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 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, ignores static power consumption or considers it a constant factor while doing task allocation on such platforms.

III. 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. Each processor type π^m has z^m processing units of type m. The total number of processors in the system will be equal to $\sum_{i=1}^{M}(z^i)$. The utilisation of a given core is denoted as $U^m_{z^m}$.

B. Task Model

We assume sporadic task-model with ℓ independent tasks $\tau = \tau_1, \tau_2, \cdots, \tau_\ell$. Each task τ_i is represented as a quadruple $\langle C_i^m, D_i, T_i, \bar{E}_i^m \rangle$, where C_i^m is a vector of worst-case execution times of τ_i on M different processor types. D_i is the deadline and T_i is the minimum-inter arrival time. As derived value $U_i^m = C_i^m/T_i$. For the sake of simplicity, we assume implicit deadline meaning $D_i = T_i$. $\bar{E}_i^m$ is a vector of the average-case energy consumption of τ_i on M different processor types at their maximum speed. 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 τ_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 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, this 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 result in a different energy consumption. Therefore, two application 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. The power consumption of the processor type π^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^w 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 $2 * tr_n^m$.

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 core is set with respect to its ascending order of energy consumption. The most favourite core type for the task is the one where its energy consumption is minimal. Similarly, a core type is least preferred where the energy consumption of the 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 assignments 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. For simplicity sake, we assume in the discussion only a single core of each processor type and has a utilisation of U^m. Multiple cores per type essentially only increase the processing capacity, but do not provide new insights. Consequently 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 scenario, where more than one tasks 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 $ED_i^m = \bar{E}_i^m / T_i$ of a task τ_i on a core π^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 the task on the current core type from the next higher energy density value of the same task on another core. It can computed with the following expression $DD_i^m = min\{ED_i^k : k \neq m \wedge ED_i^k \geq ED_i^m\} - ED_i^m$. It defines how much extra energy it will consume, 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 task 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 are allocated to core. The intuition behind such 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. An allocated task 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 worst-case, the process is iterated over each core at most ℓ 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 τ_w on a core type π^q). To obtain the DD value of the task τ_w on its least preferred core type ($\max_{x=1,\cdots,M} ED_w^x$), 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 π^q, all tasks on π^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 π^q. For instance, τ_x is the current task on top of the sorted list with respect to DD values on core π^q. The algorithm attempts to allocate τ_x to π^q. If π^q can accommodate τ_x (line 17-18), it does not consider τ_x on other cores 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, τ_x is not considered for allocation on other core types where it consumes more or equal energy compared to this core type π^q. If the task τ_x was previously allocated to these higher energy consuming core types, it is deallocated on such cores (line 20). Once the allocation for τ_x is completed on π^q, LLED attempts to allocate the next task in the sorted list. If any of the task in the order cannot be allocated to π^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 ℓ times. Lines 13−26 in Algorithm 1 corresponds to these steps. Therefore, complexity of this algorithm in $O(\ell^2 \times M)$. The working of the algorithm is demonstrated with an example.

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

1: $U^m = 0$ for each core π^m
2: Compute ED_i^m for each τ_i on each core
3: **for** $q = 1$ to M **do** {/* For all processor types */}
4: **for** $w = 1$ to ℓ **do** {/* For all tasks */}
5: **if** $ED_w^q \neq \max_{x=1,\cdots,M} ED_w^x$ **then**
6: $ED_w^r = \min_{x=\{1,\cdots,M\}\setminus q \&\& ED_w^x \geq ED_w^q} ED_w^x$
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 ℓ **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 τ_w to π^q
19: $\forall_{x\in[1,\cdots,M]\setminus q}$ remove $MT_w^x \; iff(\bar{E}_w^x \geq \bar{E}_w^q)$
20: $\forall_{x\in[1,\cdots,M]\setminus q} \; U^x - = U_w^x \; iff(\bar{E}_w^x \geq \bar{E}_w^q \&\& \tau_w$ is assigned)
21: **else**
22: Break;
23: **end if**
24: **end for**
25: **end for**
26: **end for**

a) Example: Assume, we have 4 tasks and 3 core types. The tasks specifications are given in Figure 1(a). Entries under each core type specifies $C_i, \bar{E}_i, ED_i^m$ for τ_i. The DD values are computed for all tasks and presented in Figure 1(b). As an example, the DD value of τ_1 in π^1 is computed by an expression $ED_1^2 - ED_1^1$. We start from the first core type π^1 and sort the tasks in descending order of DD values as presented in the first column of Figure 1(c). τ_4 can be allocated to π^1, therefore, its entry that consumes more energy compared to this core type is deleted in π^3 type. τ_2 cannot be allocated, therefore we move to π^2 and sort the task-set according. In core type π^2, τ_1 and τ_4 can be allocated. τ_1's entry in π^3 and τ_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 π^3, τ_2 and τ_3 can be allocated to π^3. Therefore, τ_2's entry in π^2 and τ_3's entry in π^2, π^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 π^1, the τ_4's entry is deleted, so it is not considered for allocation and system attempts to allocated next task in the order (τ_2). The rest of the process is similar to the first iteration. The end result of 2^{nd} 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 τ_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 this difference (line 5). The MM algorithm picks a task from the top of the list and assigns to its favourite core type. If

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

(a) $C_i/\bar{E}_i/ED_i^m$ Values

	π^1	π^2	π^3	T_i
τ_1	4.5/16.5/1.65	3/17.2/1.72	7/52.5/5.25	10
τ_2	8/37.65/2.51	10/65.1/4.34	8/57/3.80	15
τ_3	18/84/2.80	12/78.9/2.63	10/75.9/2.53	30
τ_4	60/259.2/2.16	35/210/1.75	80/649.2/5.41	120

(b) Density Difference (DD) in MT

	π^1	π^2	π^3
τ_1	0.07	3.53	−5.25
τ_2	1.29	−4.34	0.54
τ_3	−2.8	0.17	0.10
τ_4	3.25	0.41	−5.41

(c) 1^{st} Iteration

π^1	π^2	π^3
~~τ_4~~	τ_1	τ_2
τ_2	τ_4	τ_3
τ_1	~~τ_3~~	~~τ_1~~
~~τ_3~~	~~τ_2~~	~~τ_4~~

(d) 2^{nd} Iteration

π^1	π^2	π^3
~~τ_4~~	~~τ_1~~	~~τ_2~~
τ_2	τ_4	τ_3
τ_1	~~τ_3~~	~~τ_1~~
~~τ_3~~	~~τ_2~~	~~τ_4~~

Algorithm 2 Alternative First Phase: MaxMin (MM)

1: $U^m = 0$ for each core π^m
2: Compute ED_i^m for each τ_i on each core
3: $\forall \tau_i$: Find $ED_i^{max} = \max_{x=1,\cdots,M} ED_w^x$
4: $\forall \tau_i$: Find $ED_i^{min} = \min_{x=1,\cdots,M} ED_w^x$
5: Sort task-set with respect to $\left(ED_i^{max} - ED_i^{min}\right)$ in descending order
6: **for all** Tasks $i = 1$ to ℓ **do**
7: Sort cores with respect to the energy consumption of τ_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 τ_i to π^j
11: $U^j += U_i^j$
12: Break
13: **end if**
14: **end for**
15: **end for**

the favourite core cannot accommodate this task, an allocation attempt is made for 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 Level of Optimisations

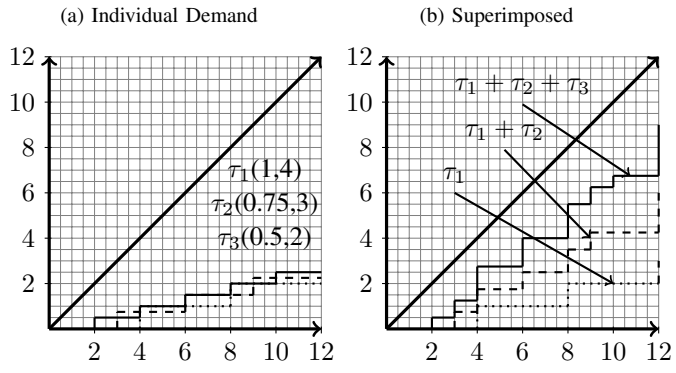
While, the first phase of allocations is derived with an objective to optimise an individual task's active energy consumption in the system, it ignores its effect on the mechanism to reduce the static power consumption. For instance, a core may have less active energy consumption but some small group 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. (Note: we use a single core per type to simplify notations.)

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 tasks to miss their deadlines 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 π^m, $th^m = \min_{\forall L \in L^*} (L - dbf(L))$, 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 π^m. ERTH initiates a sleep transition online when system is idle or has sufficient slack. The length of th^m defines which sleep state can be used online on core π^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, for the obvious reason that a sleep state saves energy when used for greater than BET_n^m time interval.

ERTH selects a sleep state for a core π^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 give a small value of th^m and restrict usage of those sleep states with $BET_n^m > T_i - C_i$. 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, algorithm must ensure that such migration reduces the overall average energy consumption of a system.

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 in descending order. Consider one of the core type π^m and assume ℓ^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 the sorted list (w.r.t $T_i - C_i$) of tasks allocated to π^m. This value is denoted as th_1^m and computed through DBF. As there is just one task, therefore, $th_1^m = T_i - C_i$. Now we superimpose the next task on the current DBF and new maximum-feasible-sleep-threshold $th_2^m = \min_{\forall L \in L^*} (L - dbf(L))$ is computed. Similarly, a third task is superimposed and correspondingly th_3^m is computed. This

Fig. 2: Demand Bound Function



process is repeated for all sorted tasks allocated to π^m and at the end we have a set of maximum-feasible-sleep-threshold values called $\rho^m = \{th_1^m, th_2^m, th_3^m, \cdots, th_{\ell^m}^m\}$. As the tasks are superimposed in the descending order of $T_i - C_i$, therefore, one of the property of ρ^m is that $th_1^m \geq th_2^m \geq th_3^m \geq \cdots \geq th_{\ell^m}^m$. Moreover, $th^m = th_{\ell^m}^m$. To illustrate the computation of ρ^m set, lets consider an example. Assume, we have three task $\tau(C_i, T_i) \Rightarrow \tau_1(1,4), \tau_2(0.75,3), \tau_3(0.5,2)$ sorted in the descending order of $T_i - C_i$ and allocated to π^m. Individual demands of these tasks are shown in Figure 2(a). Firstly, th_1^m with τ_1 is computed, i.e. 3 units. Then τ_2 is superimposed on τ_1 and th_2^m is computed, which is equal to 2.25. Finally, τ_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 ρ^m is equal to ℓ^m. Each element in ρ^m gives the maximum sleep interval with the corresponding number of tasks. A core π^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 π^m) for all element of ρ^m using the following expression $\{\forall x \in \rho^m$, find $S_n^m : S_n^m$ minimises $(x * P_a^m + Es_n^m)\}$. We know, ρ^m holds the property that $th_1^m \geq th_2^m \geq th_3^m \geq \cdots \geq th_{\ell^m}^m$. Therefore, th_2 cannot get a better sleep state when compared to th_1^m and so on.

After computing the sleep states for each ρ^m element we group the tasks that allows 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, τ_1 is added to a set of its computed sleep state. Similarly, τ_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 ρ^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 complete step is given in line 4 of 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 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 π^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 π^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, $\bar{E}_{old} = (th_{old}^m - tr_{n1}^m) * P_{n1}^m + Es_{n1}^m$ and $\bar{E}_{new} = (th_{new}^m - tr_{n2}^m) * P_{n2}^m + Es_{n2}^m$ 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, $\{\pi^1, \pi^2, \cdots, \pi^m\}$ represent the cores in descending order of $\Delta\bar{E}^m$. Initially, we select π^1. G_{top}^1 are the tasks in the top least efficient sleep state group of π^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 π^o excluding π^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 τ_j's energy density on π^o plus the energy consumption per unit of time in idle period with τ_j on π^o minus the energy consumption per unit of time in idle period without τ_j on π^o. Where, th_{new}^o and th_{old}^o are the maximum-feasible-sleep-thresholds with and without including τ_j respectively on π^o. The sleep states corresponding to th_{new}^o and th_{old}^o are determined. $\bar{E}_{new}$ and $\bar{E}_{old}$ are the energy consumption of a single sleep transition with or without τ_j respectively. The algorithm sort all the core types in ascending order of $LC_{\tau_j}^o$ to move τ_j.

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

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

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

The algorithm attempts to assign it to a core type with the least migration cost provided it is schedulable on that core. This

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 Processors M do
8:      for all Tasks in a group do
9:        Compute the local cost of migration on energy consump-
          tion of this task for all other cores
10:       Sort other cores by decreasing order of cost
11:       for all Cores except the core of the currently assigned
          task do
12:         if  Feasible on core then
13:           Assign to a core
14:           Success = True
15:           Break
16:         end if
17:       end for
18:       if !Success then
19:         Feasible = FALSE
20:         Break
21:       end if
22:     end for
23:     if Feasible && Energy New < Energy Old then
24:       Break
25:     else
26:       Undo all Assignments
27:     end if
28:   end for
29: until Previous Assignment == Current Assignment
```

process is repeated $\forall \tau_j \in G^1_{top}$. Line 8 to 22 in Algorithm 3 corresponds to this step. In case any of the tasks $\tau_j \in G^1_{top}$ 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^1_{top} 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. 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. While the theoretical complexity of the second phase is exponential as we need to find the combinations of all tasks in a group with all other groups on other cores, the actual computation time in our experiments is very fast. The execution time and the number of migrations are discussed in Section V-B(2).

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 ξ 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 π^D. The task-set is initially generated for π^D type. The WCET time C^D_i of τ_i is deemed to be $U^D_i \times T_i$, where U^D_i is the utilisation of τ_i on the default core type π^D.

TABLE I: Overview of Simulator Parameters

Parameters	Specifications
Task-set sizes |T|	$\{100, \underline{200}, 500\}$
Inter-arrival time T_i for RT tasks	$[30ms, 50ms]$
Inter-arrival time T_i for BE tasks	$[50ms, 200ms]$
Sporadic delay limit Υ	$\{10\%\}$
Best-Case execution-time limit C^b	$\{10\%\}$
Share of RT/BE tasks ξ	$\langle 30\%, 70\% \rangle$
Characteristic Factor β	$\{10\%, \underline{20\%}, 40\%\}$

TABLE II: Device Power Model Parameters

π^m	P^m_a	P^m_i	η^m	S^m_0	S^m_1	S^m_2	S^m_3
π^0	1.0	0.39	1.0	0.31	0.21	0.12	0.05
π^1	2.2	0.86	0.5	0.67	0.47	0.27	0.11
π^2	6.0	2.33	0.2	1.83	1.29	0.74	0.30
π^3	13.0	5.05	0.1	3.98	2.79	1.61	0.64
π^4	12.1	4.7	0.15	3.70	2.60	1.50	0.6

The average system capacity U_a of the given platform is computed through the average speed-up-factor η^m. The speed-up-factor defines a ratio of the clock cycle of a core π^m with reference to π^D. Suppose speed-up-factor of a core type π^m is η^m, then the average capacity of the system will be $U_a = z^1/\eta^1 + z^2/\eta^2 + \cdots + z^m/\eta^m$. However the effective utilisation U of the task-set in the experiments is controlled through a helper variable ζ, and $U = U_a \times \zeta$. The range of ζ is $(0; 1]$. In our experiments ζ is varied from 0.5 to 0.9 with a step size of 0.05. Individual utilisation of τ_i on each π^m is a random number within a range of $U_i = [(1-\beta); (1+\beta)] \times \eta^m \times U^D_i$, where β 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 Δ^s_i in the interval $[0, \Upsilon * T_i]$ and on the best-case execution time C^b_i in the interval $[C^b * C^m_i, C^m_i]$. The second level varies the behaviour of individual jobs of the same task. The interested reader is referred to [16] for details. Each set point of parameters is

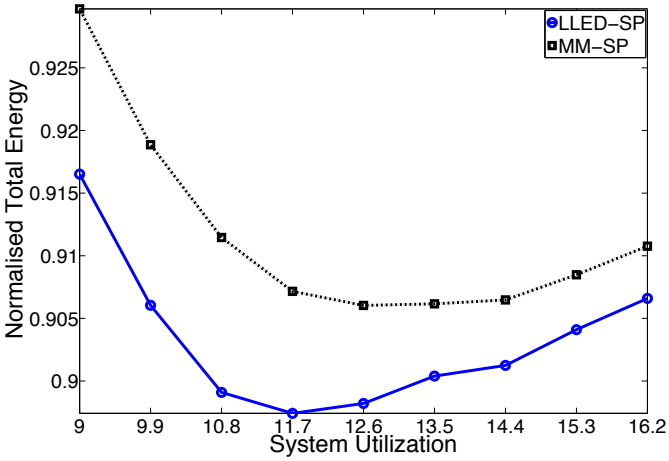


Fig. 3: 4 Core Types (S1)

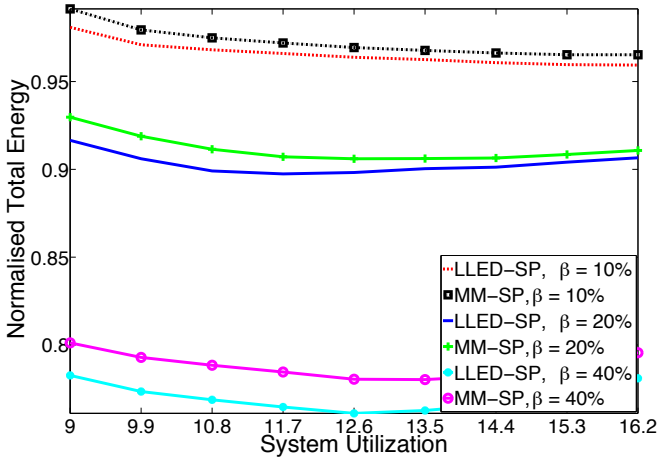


Fig. 4: Variation in β (S1)

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 FreeScalePowerQUICC III core specifications are given in Table II under $m = 4$. The 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 $\{S_x^m : x \in 0, 1, 2, 3\}$ 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 ζ in an interval of $[0.5; 0.9]$, therefore, the effective utilisation of the system U is within a range of $[0.5; 0.9] * 18 = [9; 16.2]$. The energy consumption of a task is, however, not a mere function of its execution time. As such the values of E_i^m are computed using the average execution time $\bar{C}_i^m$ and a random value similar to the utilisation conversion $E_i^m = [1 - \beta; 1 + \beta] \times P_a^m \times \bar{C}_i^m$.

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, initially, we compare our approached against the FF algorithm. Later on, the experiments of WFD are also presented and compared against the FF algorithm. 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 $LLED\text{-}SP$ and $MM\text{-}SP$ 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 overhead 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, $MM\text{-}SP$ saves in some cases energy but it is fairly minimal. Therefore, for this scenario, we compare the energy consumption of $MM\text{-}SP$ and $LLED\text{-}SP$.

Firstly, we analyse the performance of $LLED\text{-}SP$ and $MM\text{-}SP$ for different number of core types. Figure 3 shows the normalised energy consumption of the system for only 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 $LLED\text{-}SP$ and $MM\text{-}SP$ increases but then starts to shrink towards the higher utilisations. The reason for this behaviour is quite obvious that $LLED\text{-}SP$ and $MM\text{-}SP$ has more chance at low utilisation to allocate task to their favourite core. However, towards, high utilisation, this flexibility decreases along with their difference. In best-case, $LLED\text{-}SP$ consumes 10% less energy consumption when compared to FF, while $MM\text{-}SP$ saves energy slightly under 10%.

We evaluate the effect of variation in the characteristic factor β on the normalised total energy consumption of the system.

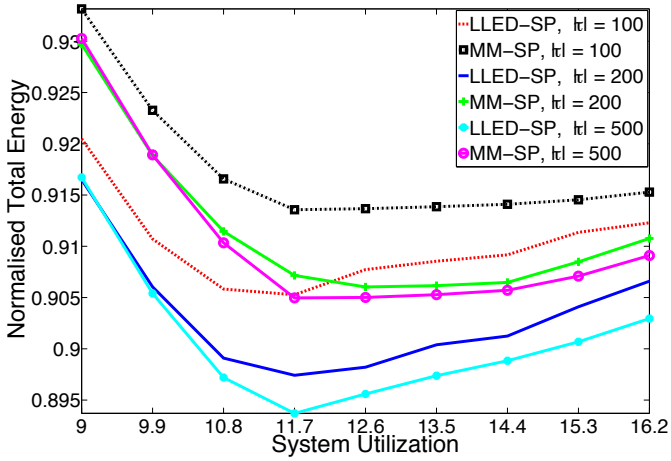


Fig. 5: Variation in Task-set Size (S1)

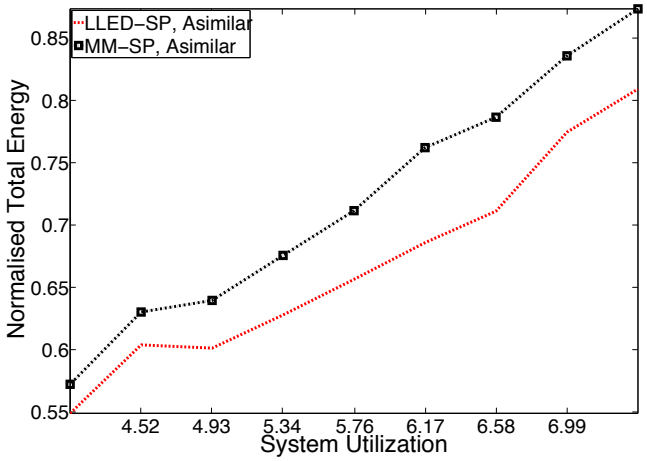


Fig. 6: Asimilar Platform (S1)

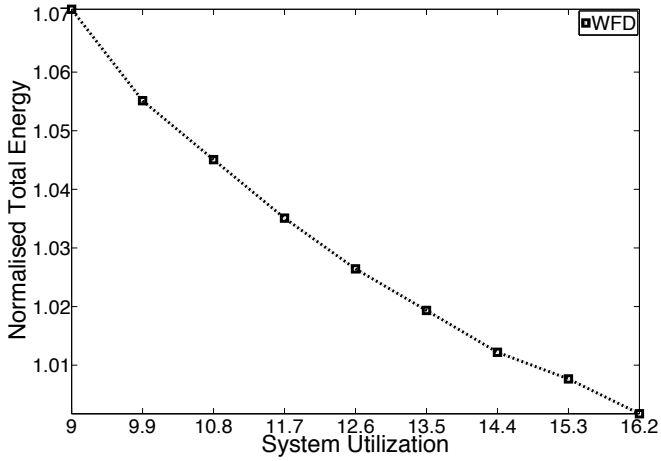


Fig. 7: 4 Core Types (S1, WFD)

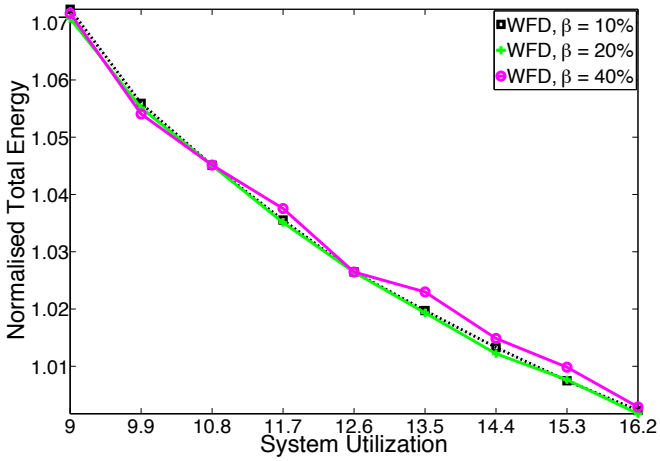


Fig. 8: Variation in β (S1, WFD)

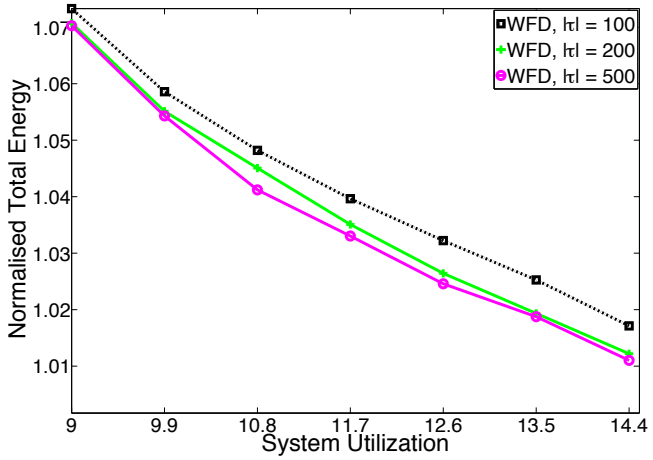


Fig. 9: Variation in Task-set Size (S1, WFD)

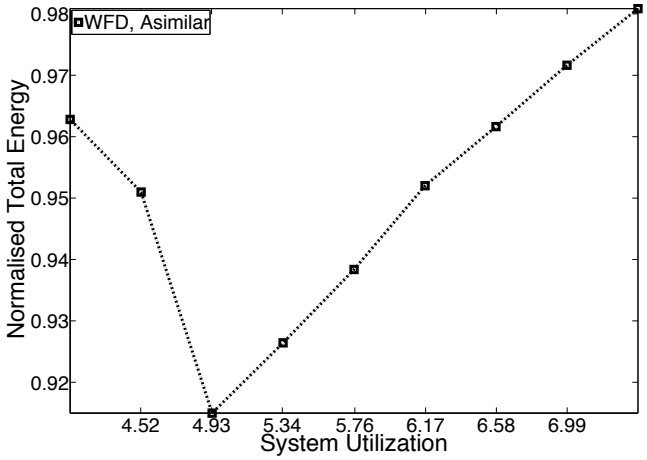


Fig. 10: Asimilar Platform (S1, WFD)

β 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 β. The developed power model on average favours the slow core. However, this factor (β) 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 $LLED\text{-}SP$ and $MM\text{-}SP$ are less at $\beta = 10\%$. However, as the β range increases, the tasks probability to favour a fast core becomes higher. Therefore, $LLED\text{-}SP$ and $MM\text{-}SP$ give better allocations for higher values of β. Similar to the previous observation, the difference of $MM\text{-}SP$ and $LLED\text{-}SP$ 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 allocations 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 $LLED\text{-}SP$ and $MM\text{-}SP$ algorithms decreases with an increase in the task-set size. However, this saving reduces with an increase in the effective utilisation. In the beginning $LLED\text{-}SP$ with the different task-set sizes do the same allocations but with an increase in effective system utilisation, the difference in allocation also increases. The same observations hold for the $MM\text{-}SP$ 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.

Processors 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 η^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 $[0.5; 0.9] * 8.22 = [4.11; 7.4]$. Figure 6 presents the results for the asimilar platform. The energy consumption of $LLED\text{-}SP$ and $MM\text{-}SP$ is low at low utilisation and gradually increases gradually towards high utilisation. All the algorithms, attempt to allocate tasks in order from the slowest core to the fastest core. $LLED\text{-}SP$ can rank tasks in an efficient way and saves more energy. Similarly, $MM\text{-}SP$ 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.

a) Comparison With Worst Fit Decreasing (WFD): We have simulated the WFD algorithm and compared it against the FF algorithm. All values are normalised to the corresponding results of FF. Figure 7 demonstrate the energy consumption of WFD for four different cores types. It is evident that WFD performs worse when compared to FF algorithm. Its performance slightly increases with an increase in the system utilisation and the difference of energy consumption with FF decreases. WFD follows the similar trend for different value of β as shown in Figure 8. One interesting observation is that change of β has very similar effect on both WFD and FF decreasing algorithms. As all the values of WFD are normalised to the corresponding values of FF, therefore, the results of different β values are similar.

We have also compared the effect of different task-set sizes on the WDF algorithm as shown in Figure 9. The performance of the WFD algorithm increases with an increase in the task-set size. Moreover, the utilisation of the system is only varied upto 14.4 in this experiment because the WFD algorithm was not able to schedule most of the task-sets on higher utilisations. For the asimilar platform, Figure 10 shows WFD performs better than FF. However, its performance is a way more worse when compared to our algorithms.

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 sleeps 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 the $LLED$, MM, $LLED\text{-}SP$ and $MM\text{-}SP$ algorithms. The base line is still the corresponding energy consumption of FF. Furthermore, we have increased the range of ζ to $[0.4; 0.9]$ with a step size of 0.05 for this scenario.

Figure 11 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 reasons is that these algorithms are not accounting the effect of their allocations 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 $LLED\text{-}SP$ and $MM\text{-}SP$ algorithm compares well to FF at low utilisations and compensates for the wrong allocations done by $LLED$ and MM respectively. It is interesting to see that for low utilisation $LLED\text{-}SP$ and $MM\text{-}SP$ achieves substantial gain. 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 11). In the detailed analysis of utilisations between 7.2 and 9, we have observed that FF loses the efficient sleep states earlier than $LLED\text{-}SP$ or $MM\text{-}SP$. Hence, the energy consumption of $LLED\text{-}SP$ and $MM\text{-}SP$ is dropped at $U = 8.1$ when compared to FF It is also evident from Figure 11 that the performance of $LLED$ is always dominant over MM, and similarly, the performance of $LLED\text{-}SP$ over $MM\text{-}SP$.

The variation in the characteristics factor β is demonstrated in Figure 12 and Figure 13. Similar to the results in scenario

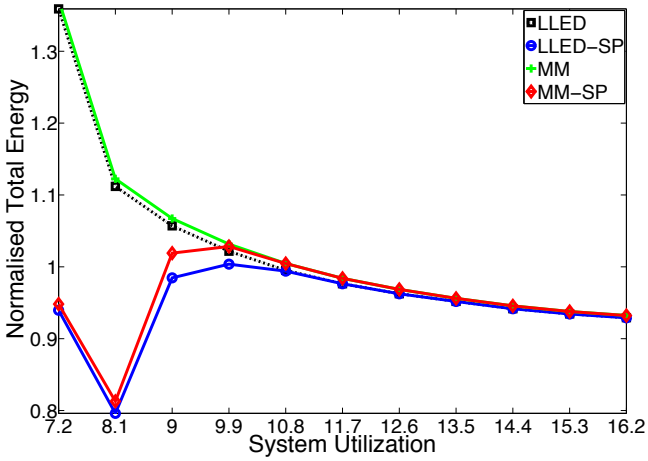


Fig. 11: 4 Core Types (S2)

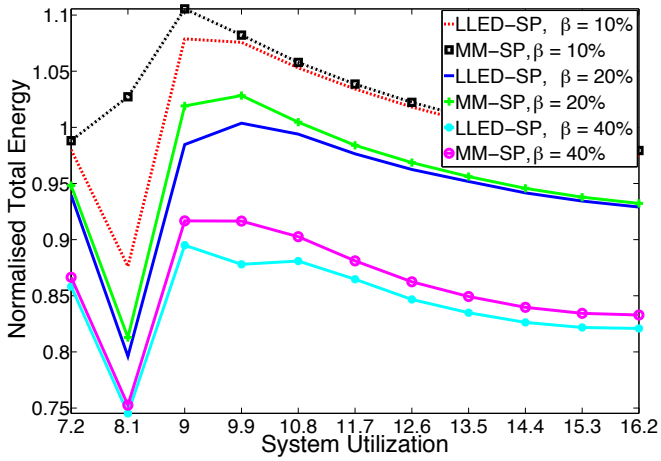


Fig. 12: Variation in β (S2)

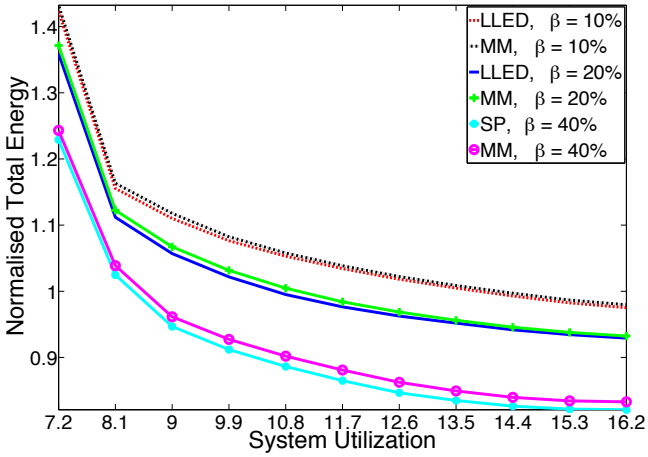


Fig. 13: Variation in β (S2)

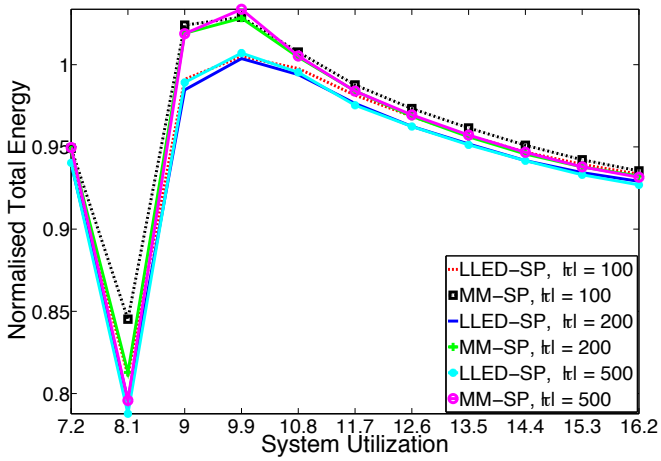


Fig. 14: Variation in Task-set Size (S2)

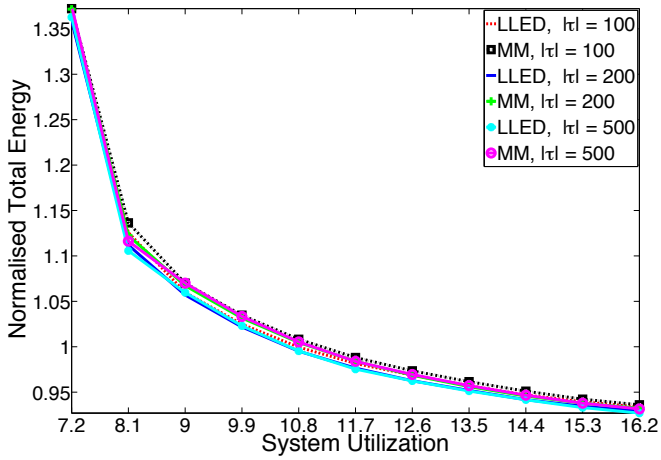


Fig. 15: Variation in Task-set Size (S2)

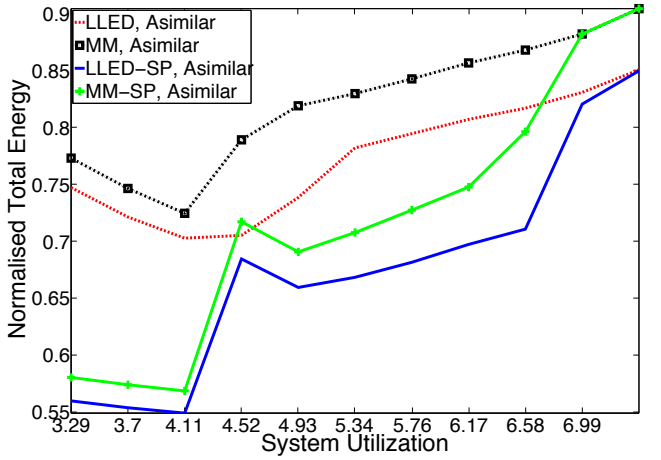


Fig. 16: Asimilar (S2)

1 (Figure 4), the performance of $LLED$-SP and MM-SP given in Figure 12 increases with an increase in the value of β and the similar trend is followed by $LLED$ and MM in Figure 13. Another observation which is clear from Figure 12 is that $LLED$-SP always dominates MM-SP and the same is true in Figure 13 for $LLED$ and MM.

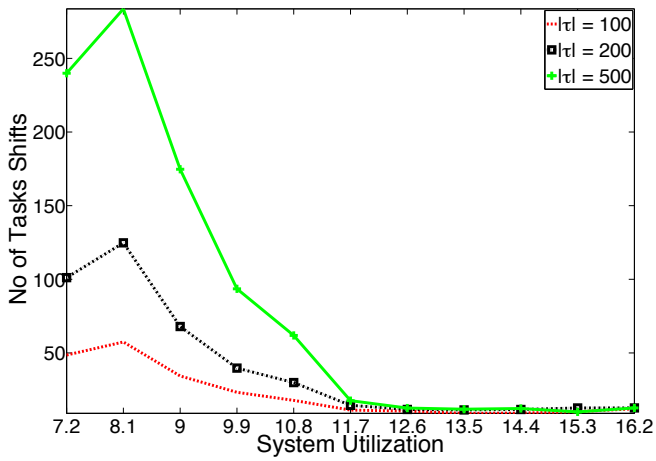


Fig. 17: Decisions (S2)

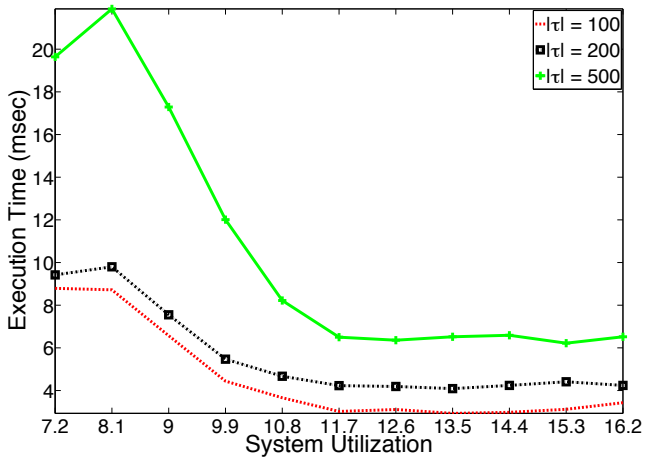


Fig. 18: Time Calculation (S2)

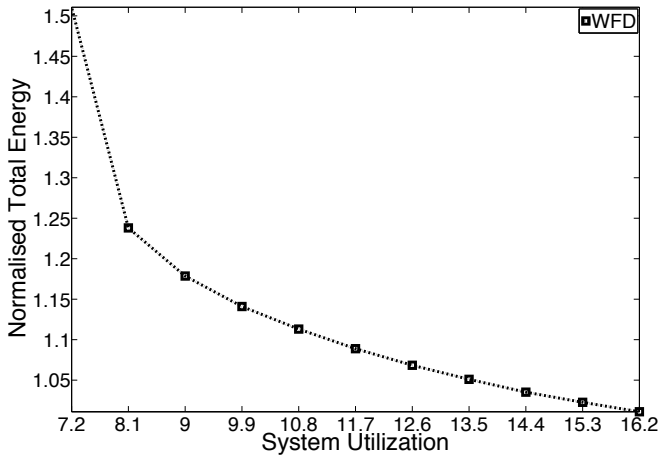


Fig. 19: 4 Core Types (S2, WFD)

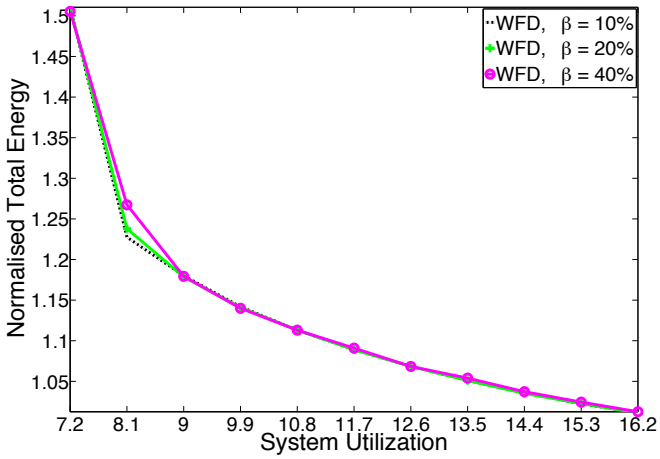


Fig. 20: Variation in β (S2, WFD)

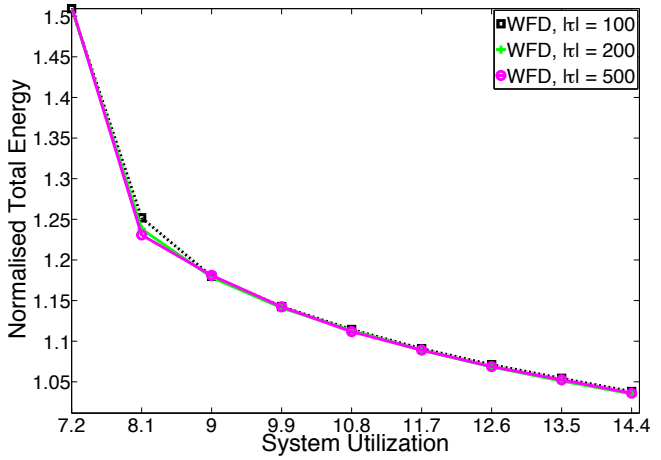


Fig. 21: Variation in Task-set Size (S2, WFD)

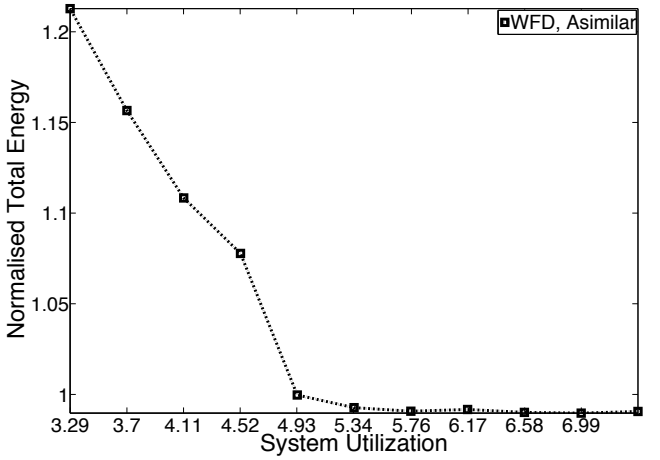


Fig. 22: Asimilar Platform (S2, WFD)

The effect of variation in the task-set size is presented in Figure 14 and Figure 15. 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 16.

All the algorithms follow the same race to allocate tasks to the slowest core. Furthermore, $LLED$ performance dominated over MM, and towards high utilisation, it even consumes less energy compared to MM-SP. Overall, $LLED$-SP performs better for all utilisations.

Figure 17 and Figure 18 present the execution time and the number of tasks migrations between different core types of the second phase of allocation respectively. Its allocation process is very fast even for a large task-set size of 500. Figure 17 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 $U = 8.1$.

a) Comparison With Worst Fit Decreasing (WFD): Similar to the previous scenario, we have also compared WFD with FF in the second scenario. All the values of the WFD algorithm are normalised to the corresponding values of FF for consistency. For four core types, normalised energy consumption of the WFD algorithm is shown in Figure 19. The shape of the curve is similar to the previous scenario (Figure 7). The difference of energy consumption between FF and WFD is higher in this scenario when compared to first scenario. The variation in β and task-set sizes also have the similar effects when compared to previous scenario. The normalised energy consumption for different values of β and different sizes are shown in Figure 20 and Figure 21 respectively. In this scenario, the energy consumption of WFD is higher for the asimilar platform as shown in Figure 22.

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 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?fpsp=1

[18] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in IPDPS 2003, april 2003.