2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications

Energy-Efficient Scheduling of Real-Time Tasks on Heterogeneous Multicores Using Task Splitting

Di Liu, Jelena Spasic, Peng Wang, Todor Stefanov
Leiden Institute of Advanced Computer Science
Leiden University, Leiden, The Netherlands
Emails: {d.liu, j.spasic, p.wang, t.p.stefanov}@liacs.leidenuniv.nl

***Abstract*—In this paper, we investigate the problem of using the state-of-the-art C=D task-splitting approach to energy efficiently schedule real-time tasks on a single-ISA heterogeneous multicore system. We first extend the existing task-splitting approach for heterogeneous multicore systems. Based on our extension, we propose an algorithm, called ASHM, to allocate and split real-time tasks on a heterogeneous multicore system. The experimental results demonstrate the effectiveness of our proposed ASHM algorithm compared to existing allocation approaches in terms of energy savings.**

I. Introduction

Multicore systems are widely adopted to satisfy the increasing computational demands of applications and at the same time to reduce energy consumption. Homogeneous multicore systems which consist of identical cores are ubiquitous in modern electronic systems spanning from mobile devices to supercomputing systems. However, the emergence of multicore systems brings a new issue, called the dark silicon problem [1], i.e., not all cores on the chip can be powered on at the same time due to power density issues. Several solutions [2], [3] have been proposed in the recent years to mitigate the dark silicon issue. Heterogeneous multicore systems [4] have been considered to be one of the promising solutions for the dark silicon problem and a good alternative to homogeneous multicore systems.

A single-ISA heterogeneous multicore system [5] is a special heterogeneous multicore system, where the cores on the chip have the same instruction set architecture (ISA) but differentiate with each other in terms of power and performance. Typical single-ISA heterogeneous multicore systems usually consist of two types of cores; The 'big' cores with complex micro-architecture, e.g., deep pipeline and wider issue width, are designed for high performance and the 'little' cores with simple micro-architecture, e.g., shallow pipeline and narrower issue width, are optimized for low power execution. Several leading semiconductor companies have mass-produced their own single-ISA heterogeneous multicore systems for commodity products, e.g., Qualcomm Snapdargon 810 and 808, Samsung's Exynos 5 Octa series [6], and Nvidia's Tegra X1 [7]. In the remainder of this paper, when we refer to heterogeneous multicore systems, we mean single-ISA heterogeneous multicore systems.

The energy efficiency of heterogeneous multicore systems also attracts attentions when designing real-time systems. Several works [8]–[10] consider to schedule real-time tasks on heterogeneous multicore systems in an energy-efficient manner. Real-time scheduling on multicore systems falls into two categories, partitioned scheduling and global scheduling. Partitioned scheduling algorithms which statically allocate tasks to fixed cores are easy to implement and can use existing uniprocessor scheduling algorithms to schedule tasks on each core, but they suffer from the capacity loss, i.e., resource utilization inefficiency [11]. On the other hand, global scheduling algorithms allow tasks to migrate at runtime in order to achieve higher resource utilization. However, global scheduling suffers from high scheduling overheads, e.g., the task migration overhead and cache coherency problems.

In order to overcome the above scheduling problems, an alternative, called semi-partitioned scheduling/task-splitting [12]–[15], is proposed to improve schedulability and resource utilization. In semi-partitioned/task-splitting approaches, most of the tasks are statically allocated to cores and only a few tasks are permitted to migrate between cores. Among these proposed semi-partitioned/task splitting approaches, the C=D task-splitting [12] has been shown in [14] to outperform others in terms of schedulability. Moreover, the C=D approach provides a practical paradigm in terms of implementation on real platforms. Burns et al. in [12] provide an implementation guide for the Ada language or on Linux systems. More details about the C=D are given in Section III-D.

The task-splitting approach has been investigated by Lu and Guo [16] to energy-efficiently schedule real-time tasks under fixed-priority scheduling on homogeneous multicore systems. However, there is no work investigating the task-splitting approach on heterogeneous multicore systems to optimize the energy efficiency of a system. Therefore, in this paper, we investigate how to adopt the task-splitting approach with dynamic priority scheduling to better utilize the resources on heterogeneous multicore systems for energy efficiency. We select the C=D approach [12], discussed above, to split tasks among heterogeneous multicores. We extend the C=D approach for heterogeneous multicore systems and propose an allocation algorithm to schedule real-time tasks with C=D task-splitting on heterogeneous multicore systems. Formally, our novel technical contributions are summarized as follows:

- We analyze the properties of the C=D task-splitting and extend it for heterogeneous multicore systems. We present a new definition, namely '*valid split*', for the C=D task-splitting on heterogeneous multicore systems. This analysis is presented in Section V;
- Based on the analysis of the C=D task-splitting and the characteristics of heterogeneous multicore systems, we propose an energy-efficient algorithm, called ASHM, to allocate and split real-time tasks on heterogeneous multicore systems. Algorithm ASHM is presented in

978-1-5090-2479-7/16 $31.00 © 2016 IEEE
DOI 10.1109/RTCSA.2016.40

Section VI;

- Since the existing methods to compute the minimum operational frequency for each core cannot work with the C=D approach, we propose a new approach based on Quick convergence Processor-demand Analysis (QPA) [17] to compute the minimum frequency for each core in the system. This proposed approach is presented in Section VI-D

The reminder of this paper is organized as follows. Section II discusses the related work. Section III gives the preliminaries. Section IV shows an example to motivate the application of the task-splitting approach on heterogeneous multicore systems for energy reduction. Then in Section V, we analyze the C=D task-splitting and extend it for heterogeneous multicore systems. Based on Section V, Section VI presents the proposed algorithm, ASHM, to allocate and split tasks on heterogeneous multicore systems. Finally, Section VII evaluates the proposed algorithm in comparison to existing approaches and Section VIII concludes this paper.

II. Related Work

Energy-efficient scheduling for real-time systems has been widely explored in the past two decades. Chen and Kuo in [18] comprehensively reviewed most of the papers addressing energy-efficient real-time scheduling problems before 2007. An updated review for energy-efficient real-time scheduling is provided by Bambagini et al. in [19]. We can see from [18], [19] that most of the works consider homogeneous systems, whereas in this paper we consider heterogeneous multicore systems which are more energy-efficient but more difficult to effectively schedule the tasks.

A few works consider heterogeneous systems. Chen and Thiele in [20] proposed a polynomial algorithm to energy efficiently schedule periodic tasks on heterogeneous systems but the systems they considered had only two cores. In contrast, we consider a more general system model where the system has two types of cores, and for each core type we can have any number of cores which can be seen on many real commercial processors. Chen et al. [21] developed two polynomial-time algorithms to energy efficiently allocate real-time tasks on a more general system model that can have different types of processors and different number of processors for each type like we consider in our work. However, in their work, they do not take dynamic voltage/frequency scaling (DVFS) into account, whereas we consider DVFS as a crucial technique to improve the energy efficiency. With the consideration of DVFS, we can further minimize the energy consumption of the heterogeneous system. In [22], Huang et al. proposed an allocation algorithm to schedule frame-based real-time tasks on heterogeneous multicore systems, where a non-preemptive scheduling is considered. The main difference compared to our work is: (1) they consider frame-based real-time task model, whereas we consider the periodic task model which is more general; (2) non-preemptive scheduling they consider is known to be NP-hard in strong sense even on uniprocessor [23]. In contrast, we consider preemptive scheduling.

Recently, more interests have risen for energy efficient real-time scheduling on single-ISA heterogeneous multicore systems. Liu et al. [8] consider an optimal cluster scheduling to schedule real-time tasks on cluster heterogeneous multicore systems. However, from practical perspective the optimal cluster scheduling suffers from a very high overhead caused by frequent context switching and task migration. When the overhead is taken into account, the achieved resource utilization may be quite low in practice [15]. In contrast, the C=D task-splitting, we consider, has a limited number of migrations and on each core a normal EDF scheduler is used to schedule real-time tasks, hence it significantly reduces the context-switching and task migration overhead and makes it more practical for real implementation. Colin et al. [9] and Elewi et al. [10] adopt the partitioned EDF scheduling to schedule real-time tasks on heterogeneous multicore systems, where both consider energy minimization as the objective. Due to the capacity loss of partitioned scheduling, the proposed approaches from [9] and [10] do not fully utilize 'little' cores on a heterogeneous multicore system and thus possibly lose some opportunities to further reduce the energy consumption. Contrarily, in our work, we adopt the state-of-the-art C=D task-splitting approach to exploit the energy efficiency of a heterogeneous multicore system. Our experimental results on randomly generated task sets demonstrate the merit of the task-splitting on heterogeneous multicore systems.

A few works study the task migration/splitting approaches for energy-efficient real-time multicore system. Chen et al. [24] address the energy-efficient scheduling problem on homogeneous multicore systems with task migration, in which all tasks have the same release time and a common deadline. In our work, we consider a more general and widely-used periodic task model and instead of homogeneous multicore systems, we consider heterogeneous multicore systems which are more energy efficient. Lu and Guo [16] adopt the task-splitting approach proposed by Guan et al. [13] on homogeneous multicore systems to achieve energy efficiency. The main difference between [16] and our work is twofold:1) they consider fixed priority scheduling, whereas dynamic priority scheduling, i.e., earliest deadline first (EDF) [25], is adopted in our work. It is known that EDF can achieve better resource utilization than fixed-priority scheduling; 2) they consider homogeneous multicore systems, whereas we target heterogeneous multicore systems which are more energy-efficient.

III. Background

In this section, we present the system model, task model, and energy model used in this paper. Then, we give a brief description of the C=D task-splitting approach [12] and the schedulability analysis technique [26] [17] which we use in this paper.

A. System Model

We consider a heterogeneous multicore system M which consists of two types of cores, the 'big' core for performance and the 'little' core for low power. In our paper, we use PE and EE to denote a 'big' core and a 'little' core, respectively. We use M_{EE} and M_{PE} to denote the sets consisting of all EE cores and all PE cores, respectively. The power consumption of one core can be computed by the following equation,

$$P(f) = \alpha f^b + s \tag{1}$$

where α and $b \in [2,3]$ are technology-based parameters [18], f is the operational frequency. For different types of cores, α and b are different. The first term of Eq. (1) is the frequency-related power consumption, i.e., the dynamic power consumption. s denotes the power consumption unrelated to the frequency, i.e., the static power consumption. Each core executes independently from the others and has a discrete frequency set at which the core can run. Let $\overline{f_j} = \{f_1, \cdots, f_l\}$ denote the frequency set of core j. Without loss of generality, we assume that the frequencies in the set are sorted in increasing order, i.e., $f_k < f_{k+1}$.

B. Task Model

Consider a task set Γ which consists of n independent real-time tasks. We adopt the widely-used periodic task model, where a periodic task which might produce an infinite number of jobs is characterized by a tuple of parameters $\tau_i = \{C_i^{EE}, C_i^{PE}, D_i, T_i\}$.

- C_i^{EE} and C_i^{PE} are the worst-case execution times (WCETs) of task τ_i executing on the EE core and PE core at the maximum frequency, respectively;
- D_i is the relative deadline of task τ_i;
- T_i is the period of task τ_i. In this work, we consider that each task has an implicit deadline, i.e., $D_i = T_i$.

C. Energy Model

With the system and task models discussed above, we explain how to compute the energy consumption for the system. After all tasks are allocated to cores, the energy consumption for each core can be computed as follows:

$$E_j = hp\left(\alpha_j f_j^{b_j} \frac{f_{max}}{f_j} \sum_{\forall i \in \Gamma_j} \frac{C_i}{T_i} + s_j\right) \quad (2)$$

where Γ_j is the task set containing all tasks allocated to core j and hp is the hyper-period of task set Γ. The hyper-period is the least common multiple (LCM) of all tasks' periods. Every hyper-period has the same workload and thus we compute the energy consumption within one hyper-period. The energy consumption of the whole system is the summation of the energy consumption E_j of all cores.

D. C=D Task-Splitting

In this work, we adopt the C=D task-splitting to schedule real-time tasks on a heterogeneous multicore system. Burns et al. in [12] proposed the C=D approach to split real-time tasks on homogeneous systems. They use a preemptive earliest deadline first (EDF) scheduling [25] to schedule the tasks on each core. The tasks are first allocated to cores according to a certain allocation algorithm. If task τ_i cannot be integrally allocated to a core, the C=D approach splits unassigned task τ_i into two parts/subtasks, τ_i^1 and τ_i^2. The split procedure is as follows:

- Find a processor x and then compute the maximum computation time C_i^1 for subtask τ_i^1 which ensures the schedulablility of subtask τ_i^1 on processor x. For subtask τ_i^1, its deadline D_i^1 is set to be equal to C_i^1 and its period T_i^1 is equivalent to its original period T_i, i.e., $\tau_i^1 = \{C_i^1, D_i^1 = C_i^1, T_i^1 = T_i\}$. Then, subtask τ_i^1 is allocated to processor x;
- According to subtask τ_i^1, we can obtain the second subtask τ_i^2. The WCET C_i^2 of τ_i^2 is computed as $C_i^2 = C_i - C_i^1$, its deadline D_i^2 is computed as $D_i^2 = D_i - D_i^1$ and its period T_i^2 equals to its original period, $T_i^2 = T_i$, i.e., $\tau_i^2 = \{C_i^2 = C_i - C_i^1, D_i^2 = D_i - D_i^1, T_i^2 = T_i\}$. Subtask τ_i^2 is allocated to a processor which has enough space to schedule subtask τ_i^2 and is different from processor x on which subtask τ_i^1 is allocated.

In the remainder of this paper, we call subtask τ_i^1 the first subtask and subtask τ_i^2 the second subtask.

The C=D task-splitting permits each core to have only one first subtask τ_i^1. This means that the whole system has at most m split tasks, where m is the number of cores. This task-splitting scheme can be realized by using task migration. τ_i^1 completes its execution on the allocated core. Then it migrates to the core where τ_i^2 is assigned and continues the execution of subtask τ_i^2. From the experimental results in [14], the C=D task-splitting outperforms other existing semi-partition/task-splitting approaches in terms of schedulability.

Migration Overhead: Like [12], in our work the migration overhead is assumed to be negligible. An extensive number of experiments on real hardware systems [15] have shown that with cache coherence among cores the task migration overhead is at the similar order of magnitude as the normal context switching. The cache coherence hardware architecture, like CoreLink CCI-400 Cache Coherent Interconnect [27], has been adopted by the big.LITTLE multicore systems to maintain the cache coherence between cores. Therefore, the migration overhead is accounted for in the WCET of a task.

E. Schedulability Analysis for EDF

For implicit deadline tasks on one core, the total utilization $U \leq 1$ is an exact test, i.e., necessary and sufficient, to ensure the schedulability of a task set under EDF [25]. However, after adopting the C=D task-splitting, the split subtasks become constrained deadline tasks where $D_i < T_i$ and the exact test for such tasks is more complicated which is based on the computation of the demand bound function (DBF) [26]. To better deal with the complexity of the exact test, Zhang and Burns [17] proposed a new exact test, namely Quick convergence Processor-demand Analysis (QPA) [17], to quickly test the schedulability of a task set under EDF scheduling. The extensive experimental results in [17] demonstrate the efficiency of QPA in terms of reducing the time complexity of testing the schedulability. Therefore, in our work, we use QPA to test the schedulability when there are subtasks (constrained deadline tasks) on a core. We refer interested readers to [17] for more details about QPA.

IV. Motivational Example

In this section, we use an example to motivate the application of the C=D task-splitting approach on heterogeneous multicore systems for energy efficiency purpose. For simplicity, assume that we have a multicore system with one PE core and one EE core. The PE core and EE core have different power parameters α and b (see Eq. (1) and (2)). In this example, we use the parameters from [8], where they obtained these parameters based on measurements on the real hardware

Core type	$\alpha(W/Mhz^b)$	b	$s(W)$
PE	3.03×10^{-9}	2.621	0.155
EE	2.62×10^{-9}	2.12	0.022

TABLE I: Power parameters for different core types

	$C^{PE}(ms)$	$C^{EE}(ms)$	D(ms)	T(ms)
τ_1	55	110	100	100
τ_2	20	40	100	100
τ_3	20	40	100	100
τ_4	15	30	100	100

TABLE II: The original task set

platform ODROID XU-3 [28]. Table I gives these parameters.

Suppose to have four tasks with the parameters given in Table II. As far as the deadlines can be ensured, we strive to partition/allocate as many tasks as possible to the EE core in order to save energy consumption. However, since scheduling τ_1 on the EE core will violate the deadline guarantee, only τ_2, τ_3, and τ_4 are eligible to be scheduled on the EE core. But we cannot schedule τ_2, τ_3, and τ_4 together on the EE core, because a total utilization of $1.1 > 1$ leads to infeasibility. One task has to be scheduled on the PE core along with τ_1. Then, we obtain a fully partitioned allocation for the given task set, where τ_1 and τ_4 are scheduled on the PE core and τ_2 and τ_3 are scheduled on the EE core. In contrast to the above fully partitioned allocation, we adopt the C=D task-splitting (explained in Section III-D) to schedule the tasks on the multicore system. In the splitting case, τ_1 is scheduled on the PE core while τ_2 and τ_3 are scheduled on the EE core. But τ_4 is split into two subtasks, τ_4^1 and τ_4^2, and then we schedule τ_4^1 on the EE core and τ_4^2 on the PE core. The parameters for the subtasks are shown in the following table,

	$C^{PE}(ms)$	$C^{EE}(ms)$	D(ms)	T(ms)
τ_4^1	10	20	20	100
τ_4^2	5	10	80	100

TABLE III: Split subtasks

With the given allocation and the power parameters, we can compute a minimum frequency for each core such that the energy consumption can be minimized by using DVFS while deadlines are still ensured. Table IV shows the allocation, the minimum operational frequency of each core, and the energy consumption of the multicore system. We can see that the splitting approach saves energy consumption by 32% compared to the partitioned approach because it can effectively utilize the EE core to save energy and at the same time it can reduce the workload allocated to the PE core. As a result, the PE core in the splitting approach executes at a lower frequency compared to the partitioned approach.

From the example, we see the advantage of the C=D task-splitting approach on heterogeneous systems in terms of energy efficiency. In the subsequent sections, we will introduce our novel approach to exploit the C=D task-splitting on heterogeneous multicore systems for minimizing the energy consumption.

Mapping	PE	EE	f^{PE}	f^{EE}	Energy(mJ)
Partitioned	τ_1, τ_4	τ_2, τ_3	$1.4GHz$	$1.2GHz$	5.42
Splitting	τ_1, τ_4^2	τ_2, τ_3, τ_4^1	$1.2GHz$	$1.4GHz$	3.69

TABLE IV: Energy consumption

	C^{PE}	C^{EE}	D	T
τ_1	60	120	100	100
τ_1^1	25	50	50	100
τ_1^2	35	70	50	100
τ_1^1	41	82	82	100
τ_1^2	19	38	18	100

TABLE V: Split Example

V. C=D Task-Splitting on Heterogeneous Multiprocessor Systems

In [12], the C=D task-splitting is devised for homogeneous multiprocessor systems. However, in our work, we target heterogeneous multicore systems [4] [29] which have been emerging as an alternative of the conventional homogeneous multicore systems. In this section, we investigate how to adopt the C=D task-splitting on a heterogeneous system.

A. Task Splitting

Since, on heterogeneous multicore systems, a task's WCET is varying upon the allocated core, the splitting on the heterogeneous multicore system should pay more attention to the varying WCET and the relation between the obtained two subtasks. First, the deadline of the first subtask τ_i^1 is set according to where the first subtask is allocated. For instance, assume that a subtask τ_i^1 has its $C_i^{PE} = 5$ and $C_i^{EE} = 10$. If it is allocated to a PE core, its deadline D_i^1 equals to $C_i^{PE} = 5$, otherwise $D_i^1 = C_i^{EE} = 10$ if allocated to an EE core. Moreover, in some cases an improper split might cause a deadline miss for the second subtask τ_i^2. The following example demonstrates this issue:

Example 1. *Suppose to have task τ_1 given in row 2 of Table V. Let us assume that we split τ_1 into two subtasks τ_1^1 and τ_1^2 and allocate τ_1^1 and τ_1^2 to an EE core and a PE core, respectively. We assume that there is no constraint on the split. We give two different splits for τ_1 shown in rows 3,4 and 5,6. For the first split shown in rows 3,4, there is no problem to schedule the subtasks. However, for the second split, although the execution time on the EE core is maximized, it causes a deadline miss for subtask τ_1^2 due to $C^{PE} > D$, seen in the last row with red color.*

From the above example, we observe the potential split issue on a heterogeneous multicore system. Thus, we give the following property to ensure that a proper split on heterogeneous multicore systems is obtained:

Property 1. *On a heterogeneous multicore system, the following inequality must hold for a split task τ_i,*

$$T_i - C_i^1 \geq C_i^2 \tag{3}$$

where C_i^1 and C_i^2 are the WCETs of subtasks τ_i^1 and τ_i^2, depending on which type of core the subtasks have been allocated.

This property is to ensure enough space to execute the second subtask τ_i^2 on a heterogeneous system. We can see that for subtask τ_i^2 it must have,

$$D_i^2 \geq C_i^2 \tag{4}$$

Since $D_i^2 = D_i - D_i^1 = T_i - D_i^1$ and $D_i^1 = C_i^1$, see Section III-D, we obtain

$$T_i - C_i^1 \geq C_i^2 \tag{5}$$

Thus, the property is observed. Based on this property, we give the following definition,

Definition 1 (valid split)**.** *If two subtasks τ_i^1 and τ_i^2 obtained by splitting task τ_i satisfy Property 1, we call such split a valid split.*

If the split is not a *valid split*, then the second subtask cannot meet its deadline.

B. Subtask Allocation

In Section V-A, we discussed how to find a *valid split* for a task on a heterogeneous multicore system. Here, we continue to discuss the allocation of subtasks. Before proceeding to the discussion, we distinguish tasks in two categories and give their definitions as follows,

Definition 2. *If a task can be integrally scheduled on an EE core, we call such task an* ***eligible task*** *(E-task).*

Definition 3. *If a task* ***cannot*** *be integrally scheduled on an EE core, we call such task a* ***non-eligible task*** *(NE-task).*

If we look at the motivational example in Section IV-Table II, τ_2, τ_3, and τ_4 are E-tasks and τ_1 is NE-task. Now, we discuss the possible allocation destinations for these two categories of tasks.

1) E-task: When an E-task is selected to be split, any split is a *valid split* regardless of which type of core the subtasks are allocated. Therefore, for an E-task, the two subtasks can be allocated to any type of core, as long as the schedulability of the system is ensured. Thus, we can have three possible combinations to allocate the two subtasks of an E-task:

- Allocate the two subtasks to two EE cores;
- Allocate the two subtasks to one EE core and one PE core; and
- Allocate the two subtasks to two PE cores.

2) NE-task: When a NE-task is about to be split, we need to ensure that the obtained split is a valid split by satisfying Property 1. For a NE-task, we cannot allocate the two subtasks to two EE cores, because Property 1 will be violated and then it leads to an invalid split. Excluding the invalid combination, we have two possible combinations to allocate the two subtasks of a NE-task:

- Allocate the two subtasks to one EE core and one PE core; and
- Allocate the two subtasks to two PE cores.

With the above possible allocation destinations for the two categories of tasks, in the next section, we will use this information to devise an energy-efficient allocation strategy for each category of tasks.

VI. Allocation and Split on Heterogeneous Multicore Systems (ASHM)

In [20], Chen and Thiele have shown that allocating real-time tasks onto two different processors is an NP-hard problem. Their problem is just a subset of our problem, so our problem is also an NP-hard problem. Hence, we propose a heuristic algorithm to energy-efficiently schedule real-time tasks on heterogeneous multicore systems with task-splitting. We call this algorithm ASHM. ASHM first handles all E-tasks and then all NE-tasks. For the sake of clarity, we first explain the different parts in the ASHM algorithm and after that we explain the whole ASHM algorithm. Before proceeding to the detailed discussion, we introduce the following property for the core with first subtask τ_i^1 allocated on it,

Property 2. *A core must run at the maximum frequency if first subtask τ_i^1 of a split task τ_i is assigned to it.*

It is trivial to see this property because the first subtask of a split task has its WCET equal to the deadline. Scaling down the frequency leads to a deadline miss. This property is useful to determine the allocation of the subtasks.

A. Allocation and splitting of E-tasks

ASHM first starts to allocate and split E-tasks. The procedure to allocate and split E-tasks is summarized as follows:

1) Use a bin-packing algorithm, first-fit-decreasing (FFD) [30], to integrally allocate E-tasks to EE cores;
2) Split unallocated E-tasks on the platform. For a given unallocated E-task τ_i, we use the following allocation and splitting order,
   a) Split τ_i among two EE cores. If it fails, try step b);
   b) Split τ_i among one EE core and one PE core. If fails, try step c);
   c) Allocate τ_i integrally to one PE core. If it fails, try step d);
   d) Split τ_i among two PE cores. If it fails, the system is unschedulable on the platform with $M = \{M_{EE}, M_{PE}\}$.

For the first step, we use FFD to integrally allocate EE tasks to EE cores because FFD is proven to be the resource efficient bin-packing algorithm [31]. By using FFD we could leave some EE cores with a lot of free capacity. This could later benefit the NE-tasks for energy saving.

After some E-tasks are integrally allocated to EE cores, we might have some E-tasks left unallocated. The next step is to split and allocate them on the system. The allocation and split order summarized above prioritizes the EE cores to explore the energy-efficient potential on the EE cores. Therefore, we first try to allocate the subtasks of a split E-task to two EE cores. If the task cannot be split among two EE cores, this means that there is no enough space on EE cores. So, we try one EE core and one PE core. Since, a PE core consumes much more power than an EE core and Property 2 indicates the maximum frequency requirement, it is not favorable to allocate the first subtask to a PE core. Therefore, we constrain ourself to allocate the first subtask to an EE core and the second subtask to a PE core. For the selection of the PE cores, we use the approach proposed in [9] which selects the core

Algorithm 1: E-task Allocation and Split (EAS)

```
input : All E-tasks Γ_E and the heterogeneous multicore
        platform M = {M_EE, M_PE}
output: Allocation for all E-tasks
1  M_EE ← using FFD to allocate tasks from Γ_E
2  Γ_un ← unallocated tasks from Γ_E
3  for ∀τ_i ∈ Γ_un in order of decreasing U do
4      for ∀x ∈ M_EE in order of increasing U do
5          τ_i^1, τ_i^2 = SPLIT(τ_i, x)
6          if τ_i^1 ≠ ∅ then
7              x ← τ_i^1
8              y ← mpwr(M = {M_EE, M_PE}, τ_i^2)
9              if y = ∅ then
10                 x ← x − τ_i^1
11     if τ_i is not allocated successfully then
12         x ← mpwr(M_PE, τ_i)
13         if x ≠ ∅ then
14             x ← τ_i
15         else
16             for ∀x ∈ M_PE in order of decreasing U do
17                 τ_i^1, τ_i^2 = SPLIT(τ_i, x)
18                 if τ_1^1 ≠ ∅ then
19                     x ← τ_i^1
20                     y ← mpwr(M_PE, τ_i^2)
21                     if y = ∅ then
22                         x ← x − τ_i^1
23                     else
24                         y ← τ_i^2
25     if τ_i is not allocated successfully then
26         return Unschedulable
27 return Allocation of ∀τ_i ∈ Γ_E
```

with the smallest energy cost contribution to the whole system when the task is allocated to it. If the combination of one EE core and one PE core still fails, we need to find an allocation among PE cores.

On PE cores, we first try to integrally allocate the E-task to one PE core because if we split an E-task among two PE cores, Property 2 requires that one PE core must execute at the maximum frequency which leads to a very high power consumption. Hence, we prefer to integrally allocate the E-task to one PE core than split it among two PE cores. We also use the approach from [9] to select the energy-efficient core for the task. If it still fails, we try the final step to split it on two PE cores in order to ensure its schedulability.

Algorithm 1 presents the pseudo-code to allocate and split E-tasks, called EAS, following the procedure explained above. EAS takes as inputs task set Γ_E consisting of all E-tasks and the heterogeneous multicore platform consisting of EE core set M_{EE} and PE core set M_{PE} and outputs the allocation of all E-tasks. At Line 1, we first use FFD to allocate E-tasks to EE cores integrally. If there are some unallocated E-tasks, we follow the steps introduced above to split unallocated E-tasks among two EE cores or one EE core and one PE core - see Line 3-10. We use function mpwr() to represent the core selection approach from [9], where the inputs of mpwr() are a core set and a task and the output is a core which can schedule the task and has the smallest contribution to the energy consumption. However, if the task is not allocated successfully, we have to try to allocate or split the task among PE cores - see Line 11-24. From Line 12-14, the integral allocation on one PE core is first tried. If it fails, from Line 15-24 EAS splits τ_i among two PE cores. Function SPLIT in Algorithm 1 finds the first subtask τ_i^1 with the maximum WCET which is schedulable on core x and also gives the corresponding τ_i^2. We will explain SPLIT in details later in Section VI-C.

B. Allocation and Splitting of NE-tasks

After all E-tasks are allocated, we proceed towards allocating and splitting NE-tasks on the system. The procedure to allocate and split NE-task τ_i is summarized as follows:

1) Split τ_i among one EE core and one PE core. If it fails, try step 2);
2) Allocate τ_i integrally onto one PE core. If it fails, try step 3);
3) Split τ_i among two PE cores. If it fails, it is unschedulable.

Since, after the allocation of E-tasks, EE cores might have some free space to execute parts of NE-tasks, we first try to split a NE-task among one EE core and one PE core in order to utilize EE cores for energy saving. Since the first subtask needs a maximum operational frequency (Property 2), we constrain the first subtask to the EE core and allocate the second subtask to a PE core for ensuring the schedulability. However, when we maximize the execution time of the first subtask on an EE core, it might bring a negative effect on the second subtask. Maximizing the execution of the first subtask will reduce the slack time for the second subtask, see Example 1, i.e., $D_i^2 - C_i^2$. As a consequence, the reduced slack time leaves a little space to scale down the frequency of the PE core which might compromise the energy saving from the EE core. Hence, in order to provide an energy-efficient split, we set the following constraint for splitting a NE-task on one EE core and one PE core.

$$\frac{C_i^2}{D_i^2} \leq \frac{C_i}{T_i} \tag{6}$$

Constraint (6) can guarantee that after the split the slack ratio of the second subtask is not smaller than before. Therefore, it would not require to run at a higher frequency. If the task cannot be split on one EE core and one PE core, we integrally allocate NE-task τ_i to one PE core. For the integral allocation, we try to allocate task τ_i to the PE core given by function mpwr(). If task τ_i cannot be allocated to a PE core, then we split it among two PE cores in order to ensure its schedulability.

Algorithm 2 presents the pseudo-code to allocate and split NE-tasks, where we call this algorithm NEAS. The inputs for NEAS are all NE-tasks and the platform. From Line 2-13 NEAS splits task τ_i among one EE core and one PE core. For this combination NEAS selects the EE core with the smallest utilization and the PE core given by function mpwr() to split task τ_i in order to save the energy consumption. At Line 5-7 constraint (6) is checked. If the combination of one EE core and one PE core fails to allocate task τ_i, then from Line 14-17 NEAS tries to integrally allocate task τ_i to one PE core

Algorithm 2: NE-task Allocation and Split (NEAS)

input : All NE-tasks Γ_{NE} and the heterogeneous multicore platform $M = \{M_{EE}, M_{PE}\}$
output: Allocation for all NE-tasks

1 **for** $\forall \tau_i \in \Gamma_{NE}$ *in order of decreasing* U **do**
2   **for** $\forall x \in M_{EE}$ *in order of increasing* U **do**
3   τ_i^1,τ_i^2 = SPLIT(τ_i, x)
4   **if** $\tau_i^1 \neq \emptyset$ **then**
5   **while** $\frac{C_i^2}{D_i^2} > \frac{C_i}{T_i}$ **do**
6                 $C_i^1 \leftarrow C_i^1 - 1$
7   Recompute C_i^2 according to the new C_i^1 (see Section III-D)
8             $x \leftarrow \tau_i^1$
9             $y \leftarrow$ mpwr(M_{PE}, τ_i^2)
10   **if** $y = \emptyset$ **then**
11                 $x \leftarrow x - \tau_i^1$
12   **else**
13                 $y \leftarrow \tau_i^2$
14   **if** τ_i *is not allocated* **then**
15         $y \leftarrow$ mpwr(M_{PE}, τ_i^2)
16   **if** $y \neq \emptyset$ **then**
17             $y \leftarrow \tau_i$; break
18   **for** $\forall pe \in M_{PE}$ *in order of increasing* U **do**
19   τ_i^1,τ_i^2 = SPLIT(τ_i, pe)
20   **if** $\tau_i^1 \neq \emptyset$ **then**
21                 $x \leftarrow \tau_i^1$
22                 $y \leftarrow$ mpwr(M_{PE}, τ_i^2)
23   **if** $y = \emptyset$ **then**
24                     $pe \leftarrow pe - \tau_i^1$
25   **if** τ_i *is not allocated successfully* **then**
26   **return** Unschedulable
27 **return** Allocation of $\forall \tau_i \in \Gamma_{NE}$

which can schedule τ_i and has the minimum contribution to the energy consumption. If it does not successfully allocate τ_i to one PE core, NEAS splits τ_i among two PE cores from Line 18-24. In this case, it finds the PE core with the largest utilization to schedule the first subtask τ_i^1. Because τ_i^1 requires the maximum frequency to guarantee the schedulability and the PE core with the largest utilization should execute at a high frequency compared to others, allocating τ_i^1 to the PE core would not increase the frequency too much which in turn does not lead to a lot of extra energy consumption for the task allocated to the PE core. For τ_i^2, we still use function mpwr() to find the candidate core. If splitting among two PE cores fails, NEAS returns a failure.

C. Split function

In this section, we present the SPLIT function used in EAS and NEAS discussed above. Algorithm 3 presents the pseudo-code for SPLIT. The concept behind the SPLIT algorithm is based on the approach proposed in [12] and the properties of the C=D approach on heterogeneous multicore systems identified and discussed in Section V. The inputs for SPLIT are a task τ_i and a core x while the output is two subtasks τ_i^1 and τ_i^2. The objective of function SPLIT is to find the

Algorithm 3: SPLIT

Input : τ_i and one processor x
Output: subtasks τ_i^1, τ_i^2

1 $C_i^1 = D_i^1 = (0.999 - U_x)T_i$;
2 Compute subtask τ_i^2 according to the parameters of τ_i^1 (see Section III-D)
3 **while** $C_i^2 > T_i - C_i^1$ *and* τ_i *is a* NE*-task and* x *is an EE core* **do**
4     $C_i^1 \leftarrow C_i^1 - 1$
5   Recompute C_i^2 according to the new C_i^1 (see Section III-D)
6 $\Gamma_x \leftarrow \Gamma_x + \tau_i^1$;
7 **while** *True* **do**
8   **if** $C_i^1 < 1$ **then**
9   **return** $\tau_i^1 = \tau_i^2 = \emptyset$
10   **if** *QPA(Γ_x) reports unschedulable* **then**
11         $t \leftarrow$the failure point from QPA
12   **while** *True* **do**
13             $I = (t - \text{dbf}(\Gamma_x - \tau_i^1, t))/\lfloor\frac{t+T_i-(C_i^1-1)}{T_i}\rfloor$
14   **if** $I \neq C_i^1$ **then**
15                 $C_i^1 = C_i^1 - 1$
16   **else**
17   Break;
18   **else**
19   Compute parameters for subtask τ_i^2 (see Section III-D)
20   **return** τ_i^1, τ_i^2

maximum WCET of τ_i^1 which can satisfy the schedulability on core x. The procedure is as follows:

- Initialize the parameters of subtasks. For τ_i^1 let $C_i^1 = D_i^1 = (0.999 - U_x)T_i$ (Line 1) and configure subtask τ_i^2 according to subtask τ_i^1 (Line 2), as explained in Section III-D, where U_x denotes the total utilization of processor x;
- If τ_i is a NE-task and x is an EE core (Line 3-5), ensure valid split according to Property 1;
- Use QPA [17] to test whether subtask τ_i^1 can be allocated onto core x. If it is schedulable, return the subtasks τ_i^1 and τ_i^2 (Line 10, 18-20);
- If QPA reports 'unschedulable', recompute the WCET for subtask τ_i^1. In this case, we use the recurrence approach from [12] to make sure that

$$C_i^1 = (t - \text{dbf}(\Gamma_x - \tau_i^1, t))/\lfloor\frac{t + T_i - (C_i^1 - 1)}{T_i}\rfloor \quad (7)$$

  where t is the failure point returned by QPA, i.e., the time instance $dbf(\Gamma_x, t) > t$ and $\text{dbf}(\Gamma_x - \tau_i^1, t)$ represents the demand of tasks on core x excluding subtask τ_i^1. The recurrence equation in Eq. (7) computes a maximum value for C_i^1 such that $\text{dbf}(\Gamma_x, t) \leq t$ which ensures the schedulability of task set Γ_x at time instant t. If Eq. (7) is satisfied, the recurrence procedure stops and returns C_i^1 for subtask τ_i^1. Otherwise, it decrements C_i^1 by 1 and repeats the previous procedure (Line 10-17);
- Return failure if it cannot split task τ_i on core x (Line 8-9).

Note that we use 0.999 instead of 1 to initialize a subtask at Line 1, because if utilizing 1 would result in that QPA uses the hyper-period of all tasks as bound to test the schedulability.

```
Algorithm 4: Compute Minimum Frequency (CMF)
  Input  : core x and task set Γ_x
  Output : the minimum operating frequency for core x
1 if x has a first subtask then
2     return f_max
3 else
4     Compute a minimum achievable frequency f_crit based on U_x
5     f̄ ← {∀f_i | f_i ≥ f_crit} and sort f̄ in order of increasing frequency
6     for ∀f_i ∈ f̄,   i = {1, 2, ..., k} do
7         if QPA(γ_x, f_i) reports schedulable then
8             return f_i
9     return f_max
```

Then, QPA would be very complex and time-consuming.

D. Computing the minimum frequency

We use DVFS to scale down the frequency of each core so that the energy consumption is further reduced. However, next to implicit deadline tasks (unsplit), we might have some subtasks obtained by splitting on some cores which are constrained deadline tasks. In such case, we cannot simply use the utilization-based approach [18] [19] to compute the minimum frequency. Hence, we integrate the frequency into QPA [17] to efficiently compute the minimum frequency for a core.

Algorithm 4 (CMF) presents the pseudo-code to compute the minimum operational frequency for each core. The inputs are one core x and a task set Γ_x which includes all tasks allocated to core x. The output is the minimum operational frequency for core x. If the core has first subtask τ_i^1, its frequency will be set to the maximum frequency according to Property 2 - see Line 1-2. Otherwise, we compute a minimum operational frequency for the core from Line 4-8. First, we compute a frequency called f_{crit} based on utilization U_x of core x [18]. Frequency f_{crit} can be deemed as the lower bound of the operational frequency of core x. If the operational frequency is lower than f_{crit}, the system is not schedulable. Then, we select all frequencies from the core's frequency set which are greater than f_{crit} and let these frequencies form a frequency set $\overline{f}$ sorted in order of increasing frequency - see Line 5. We start with the smallest frequency f_i in frequency set $\overline{f}$ and use QPA to test whether the task set is schedulable at this frequency - see Line 7. If it is schedulable, CMF returns frequency f_i as the operational frequency. Otherwise, we take frequency f_{i+1} and use QPA to test whether the task set is schedulable at this frequency.

E. ASHM Algorithm

Given all algorithms explained earlier, we present our complete Allocation and Split algorithm ASHM using the pseudo-code in Algorithm 5. We first divide all tasks into two task sets Γ_E and Γ_{NE}, one for all E-tasks Γ_E and another for all NE-tasks Γ_{NE}. Then, we use EAS (Algorithm 1) to allocate all E-tasks - see Line 2. If all E-tasks are successfully allocated, we proceed to allocate all NE-tasks by using NEAS (Algorithm 2) - see Line 3. Finally, we apply CMF (Algorithm 4) to compute the minimum frequency for each core - see Line 4-5.

```
Algorithm 5: ASHM
  Input  : all tasks Γ and the platform M = {M_EE, M_PE}
  Output : the allocation for all tasks and the minimum
           operational frequency for each core on the platform
1 Γ_E ←all E-tasks, Γ_NE ←all E-tasks
2 M ← EAS(Γ_E, M)
3 M ← NEAS(Γ_NE, M)
4 for ∀x ∈ M do
5     f_x ← CMF(x, Γ_x)
```

Complexity Analysis: In the worst case, EAS, NEAS, SPLIT and CMF are all pseudo-polynomial algorithms due to QPA. Although QPA has shown its efficiency in [17], its complexity is still pseudo-polynomial in the worst case. This worst-case scenario happens when the utilization U equals to 1. However, in function SPLIT, we strive to avoid the worst-case scenario to occur by setting the utilization bound as 0.999 - see Line 1 of Algorithm 3. Therefore, in practice, our algorithms can be executed very efficiently.

VII. EVALUATION

In this section, we present extensive experimental results to show the effectiveness of our ASHM algorithm in terms of energy consumption compared to two widely-used bin-packing algorithms [30] and two existing related approaches [9] [10]. We do not compare with [8], because when we use their approach on per-core DVFS system (per-cluster DVFS is considered in [8][1]) their approach is very similar to [9]. We do not compare with [21], because they do not take DVFS into account. Therefore, our approach will always save more energy consumption than [21]. Since the authors in [9] have shown that their approach outperforms the allocation approach proposed in [22], we do not compare our ASHM to [22].

A. Experimental Setup

1) **Task Generation:** To evaluate the effectiveness of ASHM, we adopt the widely-used random task generator based on *UUnifast-discard* [32]. *UUnifast-discard* enables the generation of unbiased task sets. It takes as inputs the number of tasks n and the total utilization U and generates utilization u_i for n tasks. The generation procedure is summarized as follows:

- For each task, utilization u_i is generated using *UUnifast-discard*;
- Period T_i is generated using a log-uniform distribution with a factor of 100 difference between the minimum and maximum possible task period. This presents a range of task periods from 10ms to 1s in real-time applications [12] [32];
- C_i^{PE} is computed as $C_i^{PE} = u_i \cdot T_i$; and
- C_i^{EE} is computed as $C_i^{EE} = C_i^{PE} \cdot ce_i$, where ce_i is selected from a uniform random distribution in the range $[1.8, 2.3]$ which represents the variance of the execution time on different types of cores [33].

[1]Here, the per-core DVFS system can be considered as each cluster with one core. The approach proposed in [8] actually becomes a partitioned approach.

2) **Platforms:** We have two types of cores (PE and EE) in the platforms and the core's power parameters are shown in Table I taken from [8]. In this experiment, we evaluate the effectiveness of our ASHM algorithm mainly on platforms with limited number of resources because on a platform with more resources our approach will always perform good, especially with more EE cores. Therefore, we conduct experiments on following three limited platforms:

1) Platform 1: 2 PE cores and 2 EE cores
2) Platform 2: 2 PE cores and 3 EE cores
3) Platform 3: 3 PE cores and 2 EE cores

On the three platforms, we experiment with task sets with different U and a different number of tasks.

3) **Comparison approaches:** We compare our proposed ASHM algorithm with the following approaches in terms of energy consumption:

- FFD: Allocate E-tasks and NE-tasks to EE cores and PE cores, respectively, using FFD [30]. If E-tasks cannot be allocated to EE cores, then they are allocated to PE cores using FFD;
- WFD: Similar to FFD, but instead of FFD we use WFD [30] to allocate tasks;
- EFD: The allocation algorithm proposed in [10];
- m-pwr: The allocation algorithm proposed in [9];

4) **Comparison Metric:** In the experimental results, we show the energy saving by using our ASHM compared to the above four reference approaches. The energy saving is computed as follows:

$$\text{Energy saving} = \frac{E_{ref} - E_{ASHM}}{E_{ref}} \cdot 100[\%] \tag{8}$$

where E_{ref} is the energy consumption of one of the four approaches given above and E_{ASHM} is the energy consumption of our proposed ASHM.

B. Experimental Results

All the experimental results are plotted in Figure 1 and 2. For each point in the figures, we generate 100 random task sets and compute an average energy saving. Note that only when all reference approaches can schedule the generated task set we compute an energy saving using Eq. (8). Our ASHM always can schedule more task sets than the other approaches because ASHM uses task-splitting. Since the schedulability advantage of the task-splitting approach has been reported in [12], we do not compare the number of schedulable task sets in this work.

1) **Impact of the Utilization:** In this experiment, we fix the number of tasks for different platforms and then vary the total utilization to evaluate the effectiveness of ASHM. In order to have both NE-tasks and E-tasks in the generated task set, the number of tasks is fixed to 7 for all platforms. The results are plotted in Figure 1 where the y-axis is the energy saving computed using Eq. (8) and the x-axis is the variable utilization. We can see that our ASHM outperforms all allocation approaches in terms of energy efficiency. From the experimental results, we observe:

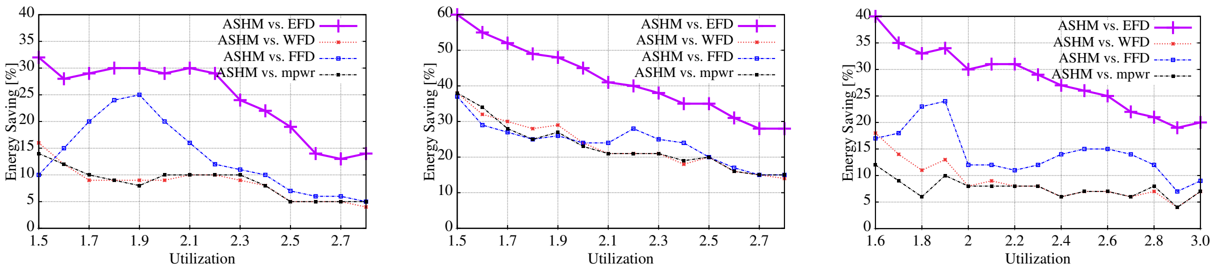
- The average energy saving by ASHM decreases as the total utilization increases. In the comparison between ASHM and WFD, EFD and m-pwr, this trend is easy to be observed although there is some variation due to the randomness of the generated task sets. The reason is that when we increase the total utilization, the slack space on the EE cores is reduced such that the task set cannot benefit from our ASHM too much. However, for FFD in Platform 1 and 3, see Figure 1(a) and 1(c), the energy saving increases until a point and then gradually decreases. The reason is that when we have task sets with a low utilization, FFD always tries to use the smallest number of cores to schedule tasks which might cause the PE cores to execute at a high frequency. The high frequency in turn leads to a high energy consumption.
- ASHM saves more energy consumption on a platform with more EE cores, see Figure 1(b). The advantage of ASHM is to effectively utilize EE cores on the platform to achieve energy efficiency. More EE cores provide more space to split tasks and thus ASHM reduces more the energy consumption.

2) **Impact of the Number of Tasks:** In this experiment, we fix the utilization for different platforms and then vary the number of tasks to evaluate the effectiveness of ASHM. Since larger total utilization leads to smaller number of schedulable task sets, we fix the utilization to 2 for all platforms in order to compare our ASHM to the reference approaches on more schedulable task sets. We ensure that the number of tasks is greater than the number of cores, so we start with 4 tasks for Platform 1 and 5 tasks for Platform 2 and 3. The results are plotted in Figure 2.

Compared to the well-performed allocation approaches WFD and m-pwr, we can see that the the energy saving is decreasing with the increasing number of tasks. The reason is that when the number of tasks increases with a fixed utilization, the tasks in the set become lighter, i.e., with a smaller utilization. Therefore, these tasks are easy to be allocated among the cores and then EE cores might be completely fulfilled or just have a little space for splitting of tasks. Therefore, ASHM cannot save too much in this case. However, as can be seen in Figure 2(a) and 2(c), compared to FFD, the energy saving by ASHM increases gradually. Since we have more tasks with a low utilization, FFD might allocate all tasks onto one core which will execute at a high frequency. However, since the dynamic power consumption still dominates the total power consumption, executing on two PE cores with lower frequencies is more energy-efficient than on one PE core with a high frequency.

VIII. Conclusion

The state-of-art C=D task-splitting [12] is studied in this work to energy efficiently schedule real-time tasks on heterogeneous multicore systems. We analyze and extend the C=D task-splitting for heterogeneous multicore systems. With our analysis and extension, we propose the ASHM algorithm to allocate and split real-time tasks on a heterogeneous multicore system. In contrast to fully partitioned allocation approaches, our proposed ASHM algorithm can effectively utilize EE cores to achieve more energy saving. The experimental results show the effectiveness of our proposed ASHM in terms of energy saving, where the maximum energy saving by ASHM compared to previous approaches is up to 60%.

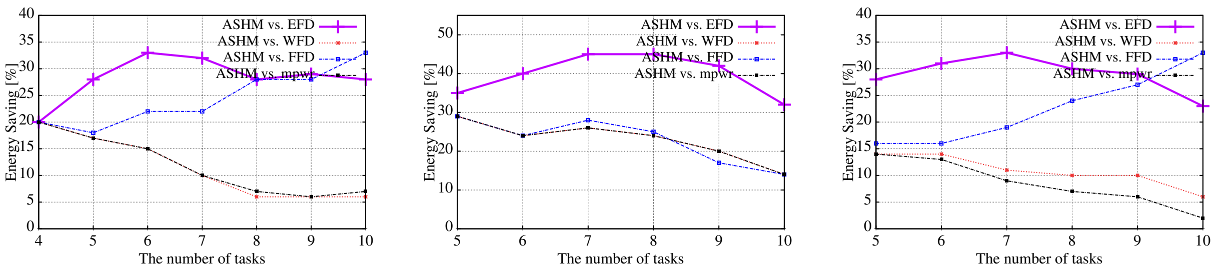


(a) 2 PE cores and 2 EE cores

(b) 2 PE cores and 3 EE cores

(c) 3 PE cores and 2 EE cores

Fig. 1: Varying U on different platforms



(a) 2 PE cores and 2 EE cores

(b) 2 PE cores and 3 EE cores

(c) 3 PE cores and 2 EE cores

Fig. 2: Varying the number of tasks on different platforms

REFERENCES

[1] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *Proceedings of ISCA*, 2011, pp. 365–376.

[2] H.-Y. Cheng *et al.*, "Core vs. uncore: The heart of darkness," in *Proceedings of DAC*, 2015, pp. 121:1–121:6.

[3] J. Henkel *et al.*, "New trends in dark silicon," in *Proceedings of DAC*, 2015.

[4] T. Mitra, "Heterogeneous multi-core architectures," *Information and Media Technologies*, vol. 10, no. 3, pp. 383–394, 2015.

[5] R. Kumar *et al.*, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings of MICRO*, 2003.

[6] Samsung Exynos, "http://www.samsung.com/."

[7] L. Gil, "Nvidias tegra x1 crushes the competition," Retrieved February 23 2016. [Online]. Available: http://liliputing.com/2015/02/nvidias-tegra-x1-crushes-the-competition.html

[8] D. Liu, J. Spasic, G. Chen, and T. Stefanov, "Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs," in *Proceedings of ESTIMedia*, Oct 2015, pp. 1–10.

[9] A. Colin *et al.*, "Energy-efficient allocation of real-time applications onto heterogeneous processors," in *Proceedings of RTCSA*, 2014.

[10] A. Elewi, M. Shalan, M. Awadalla, and E. M. Saad, "Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2s, pp. 71:1–71:27, Jan. 2014.

[11] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, p. 35, 2011.

[12] A. Burns, R. I. Davis, P. Wang, and F. Zhang, "Partitioned edf scheduling for multiprocessors using a c=d task splitting scheme," *Real-Time Syst.*, vol. 48, no. 1, pp. 3–33, Jan. 2012.

[13] N. Guan, M. Stigge, W. Yi, and G. Yu, "Fixed-priority multiprocessor scheduling with liu and layland's utilization bound," *Proceedings of RTAS*, vol. 0, pp. 165–174, 2010.

[14] J. Santos, G. Lima, K. Bletsas, and S. Kato, "Multiprocessor real-time scheduling with a few migrating tasks," in *Proceedings of RTSS*, Dec 2013, pp. 170–181.

[15] A. Bastoni, B. Brandenburg, and J. Anderson, "Is semi-partitioned scheduling practical?" in *Proceedings of ECRTS*, July 2011, pp. 125–135.

[16] J. Lu and Y. Guo, "Energy-aware fixed-priority multi-core scheduling for real-time systems," in *Proceedings of RTCSA*, vol. 1, 2011, pp. 277–281.

[17] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.

[18] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in *Proceedings of RTCSA*, 2007.

[19] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 1, pp. 7:1–7:34, Jan. 2016.

[20] J. J. Chen and L. Thiele, "Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements," in *Proceedings of ICPADS*, Dec 2008, pp. 161–168.

[21] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proceedings of IPDPS*, 2009, pp. 1–12.

[22] T. Y. Huang, Y. C. Tsai, and E. T. H. Chu, "A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem," in *Proceedings of IPDPS*, March 2007.

[23] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *Proceedings of RTSS*, 1991.

[24] J.-J. Chen *et al.*, "Multiprocessor energy-efficient scheduling with task migration considerations," in *Proceedings of ECRTS*, June 2004, pp. 101–108.

[25] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, 1973.

[26] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of RTSS*. IEEE Computer Society Press, 1990, pp. 182–190.

[27] ARM, "http://www.arm.com."

[28] ODROID, "http://www.hardkernel.com/."

[29] S. Mittal, "A survey of techniques for architecting and managing asymmetric multicore processors," *ACM Computing Surveys*, 2016.

[30] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation algorithms for np-hard problems," D. S. Hochbaum, Ed., 1997, ch. Approximation Algorithms for Bin Packing: A Survey.

[31] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of IPDPS*, 2003, p. 113.

[32] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Syst.*, vol. 47, no. 1, pp. 1–40, Jan. 2011.

[33] S. Chiang, "Advances in big.little technology for power and energy savings," Retrieved February 23 2016. [Online]. Available: http://www.armtechforum.com.cn/2012/13