INTEGRATION the VLSI journal 61 (2018) 114–124

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Mixed-criticality scheduling on heterogeneous multicore systems powered by energy harvesting

Yi Xiang[a], Sudeep Pasricha[b,*]

[a] Synopsys Inc., United States
[b] Colorado State University, Fort Collins, United States

ARTICLE INFO

Keywords:
Energy harvesting
Task scheduling
Real-time systems
Mixed-criticality
Heterogeneous platforms

ABSTRACT

In this paper, we address the scheduling problem for single-ISA heterogeneous multicore processors running hybrid mixed-criticality workloads with a limited and fluctuating energy budget provided by solar energy harvesting. The hybrid workloads consist of a set of firm-deadline timing-centric applications and a set of soft-deadline throughput-centric multithreaded applications. Our framework exploits traits of the different types of cores in heterogeneous multicore systems to service timing-centric workloads with a few big out-of-order cores, while servicing throughput-centric workloads with many smaller in-order cores clocked in the energy-efficient near-threshold computing (NTC) region. Guided by a novel timing intensity metric, our mixed-criticality scheduling framework creates an optimized schedule that minimizes overall miss penalty for a time-varying energy budget. Experimental results indicate that our framework achieves a 9.5% miss penalty reduction with the proposed timing intensity metric compared to metrics from prior work, a 13.6% performance improvement over a state-of-the-art scheduling approach for single-ISA heterogeneous platforms, and a 23.2% performance benefit from exploiting platform heterogeneity.

1. Introduction

Recent years have seen billions of embedded systems deployed around the world to support a variety of different applications domains. For an increasing number of embedded applications, there is a critical need for energy autonomous devices that can utilize ambient energy from the environment to perform computations without relying on an external power supply or frequent battery charges. Solar energy has been considered as one of most important source of ambient energy in research on management of harvesting-aware embedded systems [1–6], as the advancement of photovoltaic energy harvesting technologies at various scales have paved the way to deploy such embedded system types more widely [53–55].

Embedded computing systems that include timing behavior as part of their performance or correctness criteria are called real-time embedded systems. In such real-time systems, a deadline is called *firm* if missing it results in an immediate performance penalty, otherwise the deadline is considered to be *soft*. If critical system failure can happen after a deadline miss, the deadline is considered to be a *hard* deadline [7]. Due to the variable nature of solar radiation intensity, the most suitable role of embedded systems with solar energy harvesting as the only energy source is to host applications without strict real-time requirements. Thus it may not be desirable to consider such systems for real-time applications with *hard* deadlines, such as for life-support systems, automotive system control, aircraft navigation, etc., for which any deadline miss results in a critical system failure that may have catastrophic consequences. Instead, it is more practical to deploy such systems without energy guarantees, for best-effort execution of applications, where a *firm* or *soft* deadline miss is not considered a failure of the entire system.

Consider such a best-effort embedded system powered by energy harvesting, deployed for continuous data collection, data post-processing, and data transmission at a remote location. For each operation interval (e.g., every few minutes), a data point can be recorded from sensor modules by executing certain control tasks, for which a missed deadline results in inaccuracy in the averaged values of data features. Such tasks can be considered to be *timing-centric with firm deadlines*. On the other hand, post-processing of raw data and data transmission tasks can be delayed somewhat as the system can buffer a certain amount of data or clients can accept a lower rate of transmitted data. Such tasks are generally *throughput-centric with soft deadlines*. In this paper, we represent such applications (with different levels of real-time constraints) as mixed-criticality workloads that consist of a mix of timing-centric tasks with firm deadlines and throughput-centric tasks with soft deadlines [8,9].

* Corresponding author.
E-mail addresses: yi.xiang@synopsys.com (Y. Xiang), sudeep@colostate.edu (S. Pasricha).

https://doi.org/10.1016/j.vlsi.2017.11.007
Received 28 April 2017; Received in revised form 26 September 2017; Accepted 27 November 2017
Available online 06 December 2017

Recent years have also seen the rise of multicore processing in low-power embedded devices [10]. Due to the increasing demand for performance in emerging embedded applications, we are starting to see manycore platforms with greater core counts while maintaining energy-efficiency. As an example, MediaTek's Helix ×30 for mobile embedded applications released in 2017 [12] is a heterogeneous architecture with 10 heterogeneous ARM cores. For mixed-criticality applications, ARM is rolling out Cortex-R52 [52], a new generation of embedded processor with up to 4 cores and improved performance to address mixed-criticality workloads. Multiple Cortex-R52 processors can be integrated into a SoC, together with other ARM processor variants, for greater performance in embedded applications.

Multicore processors with heterogeneous cores have also been shown to provide substantial improvements in energy-efficiency and performance for energy-constrained systems [11]. Thus the pairing of manycore and heterogeneous computing is not a privilege for large-scale high-performance systems anymore, whose adoption to embedded processors can be seen in the big. LITTLE architecture from ARM that is continuously evolving with increased core counts [10]. With the rise in computing capabilities of emerging heterogeneous multicore processors, run-time workload distribution and energy-management in these architectures are becoming crucial steps towards minimizing the overall system energy consumption while maximizing achievable application performance. Heterogeneous platforms are particularly well-suited to execute mixed-criticality workloads as different types of cores can be utilized to better match specific criticality requirements of different types of tasks.

In addition to multiprocessing and heterogeneous computing, a new design paradigm has emerged to further help minimize energy in contemporary chip designs, called near-threshold computing (NTC) [13–18]. In NTC, the supply voltage is set just slightly higher than threshold voltage, and execution at this NTC mode achieves several times better energy-efficiency than conventional super-threshold computing (STC) [16] operation modes. NTC is thus a very effective strategy to minimize energy for energy-constrained embedded systems. However, as NTC mode operation typically sacrifices performance in favor of energy-efficiency, it is not straightforward to use it for mixed-criticality real-time systems with timing constraints.

In this paper, we focus on the important problem of *design and management of STC/NTC capable heterogeneous multicore platforms powered by solar energy harvesting and running mixed-criticality workloads, to optimize cost, performance and energy efficiency of such systems*. We propose a novel mixed-criticality scheduling framework (*McSF*), that for the first time, addresses the problem of allocating and scheduling workloads with different degrees of criticality on a heterogeneous multicore embedded system powered by energy harvesting and supporting NTC operation. Our framework employs NTC for *throughput-centric tasks* with loose timing constraints and a high degree of parallelism (DoP), maintaining their computation throughput by executing their threads concurrently on many cores in an energy-efficient manner. By improving the energy-efficiency for throughput-centric tasks, more energy budget becomes available for *timing-centric tasks*, which are allocated with awareness of harvested energy fluctuations. The novel contributions of our work can be summarized as follows:

1. Unlike any prior work, we formulate and solve the challenging problem of scheduling mixed-criticality, real-time applications on heterogeneous energy-harvesting embedded system platforms;
2. A hybrid mapping and scheduling framework is proposed to offload scheduling complexity of timing-centric task graphs to a comprehensive design-time methodology so that only lightweight adjustments are required at run-time (e.g., selecting among a small set of schedule templates, core operation modes, and task DoPs) to cope with changing energy harvesting scenarios over time;
3. For efficient execution of throughput-centric tasks, we utilize near-threshold computing (*NTC)* on several small cores to maintain high throughput levels without sacrificing energy efficiency of the computation;
4. A new energy-aware priority metric, *timing intensity-aware penalty density*, is proposed to dynamically measure the importance of instances of different task criticality types within a mixed-criticality workload.
5. Our experiments analyze the proposed mixed-criticality scheduling framework and show notable improvements in miss penalty reduction and overall performance improvement compared to a state-of-the-art scheduling approach for single-ISA heterogeneous platforms.

2. Related work

In this section, we review prior work that is relevant to our contribution in this paper. First we discuss scheduling algorithms for single-core energy harvesting embedded systems, followed by scheduling algorithms for multi-core energy harvesting embedded systems. Next we discuss work on near-threshold computing (NTC) to save energy in emerging computing platforms. Lastly, we discuss efforts to schedule mixed-criticality workloads on various platforms.

Several prior efforts have explored workload scheduling for embedded systems with solar energy harvesting, e.g., [2–6,19–22]. An early work by Moser et al. proposed the lazy scheduling algorithm that executed tasks as late as possible, reducing task deadline miss rates compared to the classical earliest deadline first algorithm [2]. However, for low-power systems with frequency scaling capabilities, delaying task execution can lead to higher frequency requirements and thus lower energy-efficiency. Liu et al. [3] used frequency scaling to slow down the execution speed of arriving tasks as evenly as possible, to save energy. In [4], a utilization-based technique was proposed for periodic task scheduling in energy-harvesting embedded systems. Zhang et al. [5] combined task scheduling with a power model of a DC-DC converter to adjust the workload for higher charging efficiency. Chetto [19] proposed a semi-online EDF-based scheduling algorithm that is theoretically optimal. *However, these works only consider single-core systems and simple independent task models with no inter-task dependencies*.

Xiang et al. [6,21] proposed a framework to manage task execution on energy harvesting powered homogeneous multicore systems with a hybrid battery/supercapacitor for energy storage. The work addressed the scheduling problem for independent periodic tasks, with awareness of the on-chip temperature profile and process variations. Xiang et al. [22] also proposed a framework to schedule task graphs on homogeneous multicore embedded systems and react to soft errors at run-time to reduce their impact on performance. Zhang et al. [20] introduced a deadline-aware scheduling algorithm with energy migration strategies specifically designed to manage distributed supercapacitors in sensor networks. *None of these prior works on scheduling for embedded systems with solar energy harvesting consider the scheduling problem for heterogeneous multicore systems, utilize the NTC execution paradigm, or support mixed-criticality workloads.*

The high energy-efficiency achievable with near-threshold computing (NTC) and its design challenges are discussed in [13]. In [14] NTC is explored as a way to address the power density problem for 3D-stacked chips. As NTC systems tend to be more sensitive to process variations with their lower supply voltage, a few recent works propose novel management techniques for NTC to alleviate the performance impact of process variations [15–17]. More recently, Karpuzcu et al. [18] proposed Accordion, a framework that executes workloads with adjustable problem sizes and fault resilience on NTC-enabled cores. Chen et al. [23] studied the impact of NTC on architectural design of processors by analyzing resulting shifts in performance bottlenecks. But to the best of our knowledge, *no prior work has addressed the scheduling problem for NTC-enabled cores powered by energy harvesting. Moreover prior work has also not considered allocation of mixed criticality workloads on heterogeneous NTC-capable platforms.*

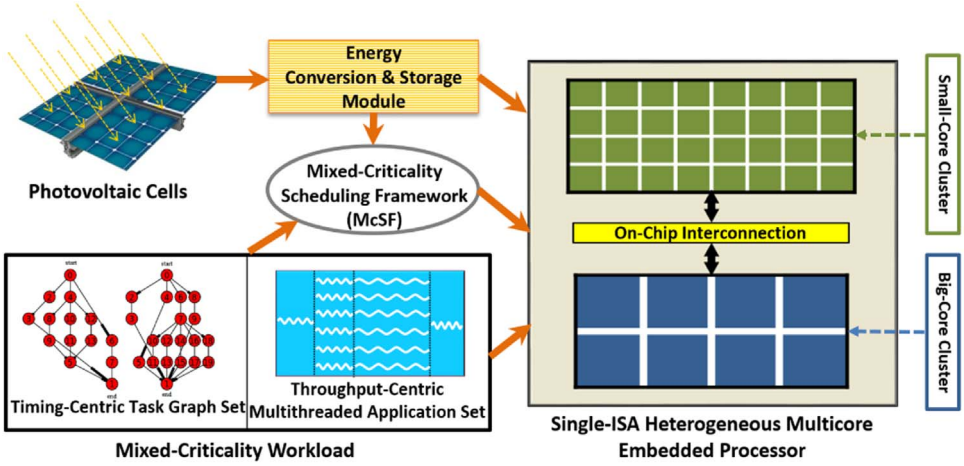


Fig. 1. Overview of our proposed McSF framework, for a single-ISA heterogeneous multicore embedded system platform with solar energy harvesting and near threshold computing (NTC) capability.

Mixed-criticality workloads are becoming pervasive in many embedded systems today. These workloads consist of applications with different timing or reliability requirements. Systems designed to support such workloads are often referred to as mixed-criticality platforms. The problem of managing a mixed-criticality workload on a single physical platform has attracted a lot of attention in recent years. An early work by Vestal [24] studied schedulability analysis and preemptive fixed priority scheduling for tasks with different criticalities. Mollison et al. [25] brought this problem to multicore systems by proposing a global mixed-criticality scheduling algorithm that can redistribute slack among tasks while maintaining isolation for tasks of different criticality levels. Giannopoulou et al. [26] proposed a time-triggered mixed-criticality scheduling approach with barrier synchronization to resolve resource sharing conflicts between applications with different criticality levels. Saraswat et al. [27] studied the topic of fault-tolerance for mixed-criticality systems. Their proposed framework tackles soft errors using checkpointing-based rollback recovery and tolerates permanent core failures by task migration. Huang et al. [28] studied fault-tolerant mixed-criticality scheduling in the presence of transient faults in the system to provide safety guarantees to tasks with different criticality levels according to established safety standards. The applicability of the proposed scheduling technique was verified for a flight management system application. Huang et al. [29] also suggested a "run and be safe" strategy that boosts processor frequency temporarily to satisfy timing requirements of critical tasks without degrading service for other tasks. A few works have also focused on the mapping and partitioning of mixed-criticality applications on multicore architectures [30–32]. *However, none of these works consider heterogeneous multicore processors as the target platform for mixed-criticality scheduling, or the added complexity due to solar energy harvesting.*

Tamas-Selicean et al. [33] explored optimization for mixed-criticality real-time applications on distributed heterogeneous nodes, but not for heterogeneous multicores integrated on a single processor chip. In [34], although heterogeneous multicore processors are initially considered as the hardware platform, the platform is virtualized to behave as a symmetric multi-processor (SMP). Craeynest et al. [35] proposed the performance impact estimation (PIE) scheduling and allocation framework for thread scheduling in single-ISA heterogeneous systems. However, it did not consider applications with mixed-criticality constraints. *Unlike any of these prior works, this paper is the first to specifically address the mixed-criticality scheduling problem for a unique computing platform that consists of a heterogeneous multiprocessor powered by solar energy harvesting.*

3. Problem formulation

Fig. 1 shows an overview of our system model that consists of a single-ISA heterogeneous multicore processor with NTC operation mode capability, an energy harvesting/ storage/conversion module, and our mixed-criticality scheduling framework (McSF) to schedule mixed-criticality workloads on to cores. In the following subsections we describe the various components and assumptions of our system model before presenting our problem objective.

3.1. Mixed-criticality workload model

We differentiate the criticality dynamics of real-time periodic tasks based on the *(m,k)* model proposed in [9]. By definition, a task in a system with an *(m,k)* deadline needs to finish at least *m* task instances out of each *k* consecutive instances to avoid system service quality degradation. However, limited by energy harvesting availability, the system cannot always guarantee full service and some tasks may have to be dropped. In our *(m,k)* model, the dropping of a task instance comes with a user-defined cost, which is called *miss penalty*. Miss penalty of a task will be applied to the system whenever an *(m,k)* deadline miss is detected. Our mixed-criticality workload is composed of tasks classified into two categories: the first is *timing-centric* real-time tasks with *(1,1)-firm* deadline constraints; the other is a set of *throughput-centric* tasks with *(m,k)-soft* deadline constraints. The criticalities of tasks of both types can be compared based on combinations of their miss penalties and *(m,k)* constraints.

Timing-centric workloads represent lightweight real-time periodic tasks in the application domain of control, sensing, communication, etc., that require a response before a specified deadline (*(1,1)*-firm). In this paper, we assume that these workloads come with highly customized and fixed degree of parallelism (DoP) adapted for efficient execution and, thus, can be best modeled as periodic task graphs [36].

Throughput-centric workloads represent general-purpose-applications such as soft real-time tasks in the domain of image processing, data mining, etc., that can tolerate some delay between samples (with *(m,k)-soft* deadline). We model these workloads as barrier-synchronized multithreaded applications [37,38] with flexible DoP, which forks/joins threads between sequential and parallel phases. Even though timing constraints for these workloads are not as stringent, they require more computing resources and support high DoPs, making it essential to exploit parallelism in order to achieve high throughput for such workloads.

In the rest of this paper, we refer to these two types of workloads as *timing-centric task graphs* (*timing-graphs*) and *throughput-centric*

Table 1
Characteristics of mixed-criticality workloads.

Workload Type	timing-graphs	throughput-apps
Criticality Type	timing-centric	throughput-centric
Structure Model	task graphs	multithreaded applications
Parallelism	highly customized	barrier-synchronized
Execution Time	few seconds	few minutes
Period	tens of seconds	tens of minutes
Deadline Model	*(1,1)-firm*	*(m, k)-soft*
Execution Rate	related to period	related to *(m, k)* and period

multithreaded applications (throughput-apps). Table 1 summarizes the differences between these two types of workloads. In this paper, we assume that all sets of periodic workloads have their execution times, periods, and deadline models profiled and defined at design-time, and are known to our mixed-criticality scheduling framework.

3.2. Heterogeneous multicore computing platform

We consider a single-ISA heterogeneous multicore platform to service mixed-criticality workloads. Similar to ARM's big. LITTLE [10], our platform combines one cluster of big cores and one cluster of small cores. In our work, both types of cores (big, small) are based on the ×86 instruction set architecture (ISA). The big-core-cluster has several high performance out-of-order cores with per-core dynamic voltage and frequency scaling (DVFS) capability that allows execution at several discrete frequency-voltage levels [45]. The small-core-cluster has several power-efficient in-order cores, all of which are clocked with uniform frequency in the NTC region to maximize energy-efficiency.

We assume no simultaneous multithreading support in the processors, thus each core only executes one task (thread) exclusively at a time. The high performance big-core-cluster is mainly, but not exclusively, utilized to execute timing-centric tasks graphs, while the small-core-cluster executes parallel phases for throughput-centric multithreaded applications. While it is common to rely on high performance cores for high throughput tasks on general-purpose computing platforms, an energy harvesting-powered system, however, obtains maximum throughput over time when energy-efficiency is prioritized. This is because the dominant throughput bottleneck for energy harvesting systems is their highly limited energy budget. Our decision to prioritize the small-core-cluster for throughput-centric applications (note that these applications can also utilize the big-core-cluster with lower priority) allows for better energy efficiency in our highly energy-constrained platform.

3.3. Energy harvesting, storage, and budgeting

A photovoltaic (PV) system is used as the power source for our multicore embedded system, converting ambient solar energy into electric power. Naturally, the amount of harvested power varies over time due to changing environmental conditions. To cope with the unstable nature of the solar energy source, we assume an energy harvesting subsystem with maximum power point tracking to extract the maximum amount of energy possible from the PV system [46]. We also assume a hybrid supercapacitor-battery storage to bridge the PV system with our embedded system efficiently [21]. Note that our run-time scheduler can operate normally as long as it can query and get a feasible energy budget from the energy storage subsystem for the upcoming schedule window; thus our approach can work with diverse energy storage solutions, e.g., battery-only or supercapacitor-only solutions.

As solar harvesting power can vary abruptly within a very short period, it is important to filter out the *noise* from incoming power so that scheduling decisions can be made and executed based on a stable and reliable energy supply. Thus, we use an energy budget assignment scheme called *energy budget window shifting* from [22] (see Fig. 2). This scheme partitions time into *schedule windows* of identical length, which in our work is the least common multiple of all timing-centric task graphs' periods. Then the energy harvested within each schedule window, combined with unused energy, is applied as the energy budget for the next schedule window. Although utilization of harvested energy is delayed for a short period of time in this scheme, it provides the run-time scheduler with a known and stable energy budget at the beginning of each window, making it easier to split the energy budget between timing-centric and throughput-centric workloads.

3.4. Problem objective

As solar energy harvesting does not guarantee energy sufficiency, our system is positioned as a soft real-time system that ensures best-effort operation adapted to a given level of energy supply available at run-time. Our main objective is to allocate and schedule mixed-criticality workloads composed of multiple timing-centric task graphs with firm-deadlines (*timing-graphs*) and throughput-centric multi-threaded applications with soft-deadlines (*throughput-apps*) running

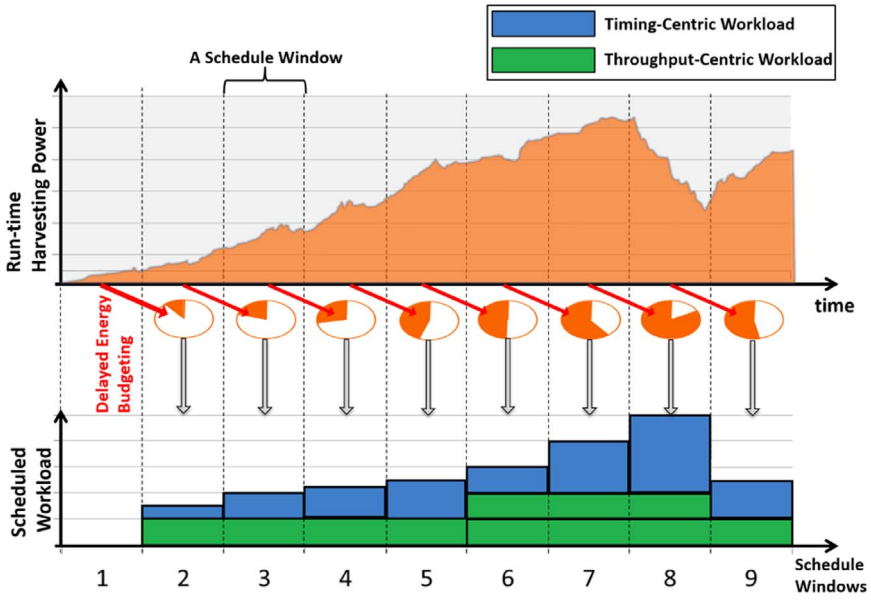


Fig. 2. Illustration of energy budgeting and workload scheduling across schedule windows over time.

simultaneously at run-time, such that the total miss penalty for the entire system is minimized, under a varying and abruptly changing harvested energy budget, as shown below:

$$\textbf{Minimize} \sum_{timing-graphs} misspenalty + \sum_{throughput-apps} misspenalty$$

Subject to harvested energy budget, soft/firm deadlines

4. McSF scheduling framework

In this section, we give an overview of our mixed-criticality scheduling framework (McSF), which consists of both design-time and run-time components.

As illustrated in Fig. 2, for each schedule window, our run-time scheduler dispatches a mix of timing-centric and throughput-centric workloads for execution, given the available energy budget and computing resources. At the system level, the scheduler intelligently enables a balanced distribution of energy budget between the two types of workloads while aiming to minimize overall system miss penalty. Due to the different characteristics and needs of these two types of workloads, each type of workload is scheduled with a specifically designed approach, as discussed next.

Timing-centric task graphs in a schedule window can be executed without considering other schedule windows, as the length of a schedule window is the least common multiple of their periods. The general problem of scheduling a task graph under optimization goals and constraints is known to be NP-complete [47]. Thus our scheduling scheme for timing-centric task graphs is designed to offload their scheduling complexity to design-time by offline generation of schedule templates that can be quickly selected for each schedule window at run-time based on the energy budget and cores made available for them after the system-level resource distribution.

In contrast, instances of *throughput-centric multithreaded applications* require execution times that can span multiple schedule windows, and thus their execution has to be scheduled dynamically. However, as the execution phases of throughput-centric multithreaded applications are barrier-synchronized, their scheduling complexity is much lower than that of timing-centric task graphs.

The following sections discuss our scheduling approach in more detail. Section 5 describes our *run-time heuristic* for penalty-aware workload filtering and scheduling for mixed-criticality workloads, while Section 6 describes the *design-time heuristic* to generate schedule templates for timing-centric task graphs.

5. Run-time mixed-criticality scheduling

In this section we describe our run-time mixed-criticality scheduling heuristic for scheduling timing-centric task graphs and throughput-centric multithreaded applications to minimize total miss penalty in the system. First, we define a priority metric to represent the impact of each task instance on system miss penalty with consideration of *(m,k)* soft deadline constraints. Then we propose a heuristic to dynamically select and schedule high-priority instances of timing-centric and throughput-centric workloads.

5.1. Soft deadline-aware priority metric

As we consider best-effort execution under insufficient solar energy harvesting conditions, it is necessary to dynamically create priorities and rank the instances of both timing-centric task graphs and throughput-centric multithreaded applications, to compare their impact on system miss penalty per unit energy. Based on this guideline, we define a new *penalty density* metric that depends on miss penalty, energy requirement, and timing intensity of a task instance, as shown below:

$$penaltydensity = \frac{misspenalty \times timingintensity}{energyrequirement}$$

Among the three components in the equation, *miss penalty* of each instance is user defined and assumed to be known at design time and *energy requirement* can be obtained by profiling applications under different operating frequency levels. However the *timing intensity* of an instance can change dynamically at run-time based on its *(m,k)* constraint and finish/miss history of previous instances. Hamdaoui et al. [9] proposed a *distance-to-failure* metric to characterize timing intensity of task instances. However, that metric only considers the next nearest instance failure in the worst case while we would like to consider all upcoming instances affected by recent execution history, to enable minimization of overall system miss penalty. Thus in this paper we propose a new and more comprehensive way to characterize the timing intensity of a task instance, as shown below:

$$timingintensity = \sum_{p=0}^{k-1} \frac{m - m_p'}{(k-p)^2} \quad , \quad m \geq 1, k \geq 1 \quad and m < k$$

where, m_p' is the total number of deadlines met (instances finished) in the last p periods, and the values of m and k are based on the user-defined *(m,k)* constraint of the task instance. We refer to every k instances as an *evaluation window*. A finish or miss of an upcoming task instance affects the results for the k upcoming evaluation windows. The *timing intensity* of an upcoming instance is essentially the accumulation of its importance factors with respect to these k evaluation windows. For an evaluation window consisting of p previous instances and $k - p$ future instances, as m_p' instances have already finished, $m - m_p$' out of $k - p$ upcoming task instances should be finished to avoid a miss penalty, resulting in a finish rate requirement of $(m-m_p')/(k-p)$. As the upcoming instance is only one of the future $k - p$ instances to contribute to this finish rate, we divide finish rate by $k - p$ to get $(m-m_p')/(k-p)^2$ as the importance factor. This definition also applies to task graphs with *(1, 1)-firm* deadlines, which is a special case with $m= 1$, $k=1$, $p=0$, $m_p'=0$ that always results in an instance timing intensity value of 1.

Fig. 3 shows an example of a *(2,5)-soft* constraint workload execution under three different scenarios. To calculate timing intensity of the upcoming instance in case *(a)*, 5 (k=5) evaluation windows are involved, as shown in Fig. 3(a). Considering the first (leftmost) evaluation window, as 2 instances have already finished, 0 out of 1 instances in the future are require to finish to meet the *(2,5)-soft* constraint, resulting in an importance factor of $0/1^2$. For the fourth evaluation window, only 1 instance has already finished. Thus 1 additional instance should be finished in the remaining 4 future instances, resulting in an importance factor of $1/4^2$. In all, the upcoming instance in case *(a)* has timing intensity of 0.143, which is

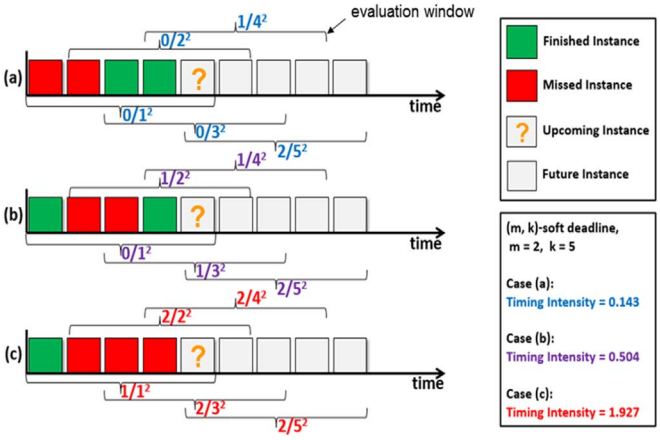


Fig. 3. Illustration of timing intensity for (2, 5)-soft deadline case.

calculated by accumulating importance factors of all the 5 evaluation windows involved.

Case *(b)* also has 2 out of 4 previous task instances finished, similar to case *(a)*. However, the first finished instance only affects the importance factor for the first evaluation window. Consequently, the other 4 evaluation windows all have higher importance factors compared to case *(a)*, causing the timing intensity of the upcoming instance to be much higher (= 0.504). *Thus, for previously finished instances, not only is their number but also their distribution affects timing intensity of the upcoming instance.* Case *(c)* shows that an upcoming instance can have a timing intensity greater than 1, as it not only must be finished to avoid miss penalty in the current period, similar to *(1,1)-firm* instances, but it also affects timing intensities of future instances.

5.2. Dynamic workload filtering and scheduling

As the proposed timing intensity-aware penalty density metric supports both *(m,k)-soft* and *(1,1)-firm* deadline constraints, it makes co-optimization of both timing-centric task graphs (*timing-graphs*) and throughput-centric multithreaded applications (*throughput-apps*) possible. In this section, a dynamic workload filtering and scheduling heuristic is proposed to perform uniform resource allocation for mixed-criticality workloads, based on the energy budget assigned or predicted in the current and future schedule windows, with the goal of minimizing overall system miss penalty. The input and output of the proposed heuristic is outlined in Fig. 4. Our heuristic progressively compares and accepts instances of *timing-graphs* and *throughput-apps* at the beginning of each schedule window. Recall (from Section 4) that *throughput-apps* require execution times that can span multiple schedule windows, while execution times for *timing-graphs* are always less than a schedule window. Also, *timing-graphs* execute on the big-core-cluster; whereas *throughput-apps* execute their parallel phases on the small-core-cluster and their sequential phases on the big-core-cluster. Our core type-aware allocation scheme optimally matches the characteristics of the input mixed-critically workload with the given target heterogeneous multicore platform. Based on this design choice, the major focus of our heuristic is to dynamically balance energy budget and core allocation wisely between two very different workload types, while ensuring that workload instances with better return per watt can get prioritized, to minimize overall system miss penalty. The heuristic procedure is shown in Algorithm 1.

First of all, the dynamic instance priorities (i.e., penalty densities) of the two types of applications are assigned on-the-fly by using two different approaches:

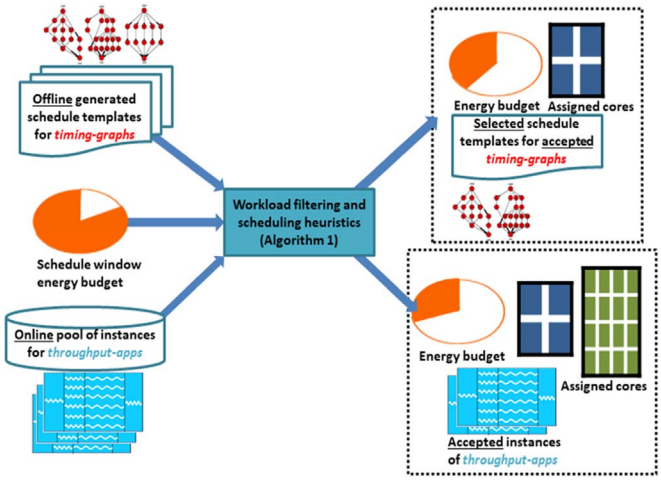


Fig. 4. Illustration of dynamic workload filtering and scheduling.

(i) *For throughput-app* instances, priorities are updated individually at the beginning of each schedule window (**step 1**). The priority of a task instance will evolve and be different from that in previous windows as timing intensity keeps changing with respect to the *(m,k)* constraint (Section 5.1). For a task instance already in execution, its priority will increase with time because the more energy it has already consumed, the less energy it requires to finish. This mechanism encourages the heuristic to resume application instances in progress so that the effort already invested in execution can be preserved.

Algorithm 1. Dynamic workload filtering and scheduling

Inputs	Definition
EGY_BGT	energy harvested and unused during last schedule window
EGY_PRD	averaged harvested energy prediction for relevant future schedule windows using mechanism from [3]
TH_pool	all instances of *throughput-apps* that have arrived or in execution
TMPL_set	Set of offline-generated *schedule templates* for scheduling *timing-graphs* (see Section 6)

Triggered at the beginning of each schedule window:
1 update priorities (i.e., penalty densities) for all instances of *throughput-apps* in *TH_pool;*
2 while (there are unscheduled task instances in *TH_pool*) **and** (*EGY_BGT* != 0) {
3 select instance *TH* ϵ *TH_pool,* with highest priority, $density_{TH}$
4 find a schedule template (*new_temp*) in *TMPL_set* that allows for the most *timing-graphs* to finish for available *EGY_BGT*;
5 calculate priority of *timing-graphs* added in *new_temp*: $density_{TI} \leftarrow \Delta penalty/\Delta energy$; †
6 if ($density_{TH} < density_{TI}$) {
7 *current_template* = *new_temp*;
8 } **else if** ($density_{TH} > density_{TI}$) **and** (*EGY_BGT, EGY_PRD* are sufficient to finish execution of *TH*) {
9 if sequential phase detected for *TH* in this schedule window {
10 steal one big core from *timing-graphs;* } //**end if**
11 start/resume execution of *TH* on lowest number of small cores*;*
12 *EGY_BGT* = *EGY_BGT* – estimated energy for *TH* in window;
13 *current_template* = schedule template in *TMPL_set* that allows the most *timing-graphs* to finish for the available *EGY_BGT*;
14 } //**end if**
15 remove *TH* from *TH_pool;*
16 } //**end while**
17 if (*EGY_BGT* != 0) {
18 *EGY_BGT* = *EGY_BGT* + energy of *current_template*;
19 *current_template* = schedule template that allows for the most *timing-graphs* to finish for the *EGY_BGT*; } //**end if**
Output: Execution schedule for *throughput-apps*; selected current schedule template for *timing-graphs*

† Δ values are based on comparison between current and new template from step 4

(i) *For timing-graph* instances, as their *(m,k)* timing intensity is always equal to 1 (Section 5.1), their dynamic priorities only change with varying energy requirements for different frequencies assigned in different schedule templates. We assume the availability of offline-generated schedule templates for different discrete combinations of number of big cores and energy budget levels (discussed in more detail in Section 6). These templates are sorted in increasing order of instance finish rates. From this available sorted template list, we select a template that enables finishing the most *timing-graphs* within the energy budget for the schedule window (step 4). Unlike the case of *throughput-apps*, here our heuristic evaluates the total priority of extra task instances that can be accepted if a new schedule template is used instead of the currently selected schedule template. This is deduced by comparing the new template's miss penalty and energy requirement to those of the currently selected schedule template (steps 4, 5). Section 6 discusses how these templates are generated at design time for *timing-graphs*.

In each iteration of the while loop (steps 6–16), priorities of candidate instances from *timing-graphs* and *throughput-apps* are compared to decide which ones to accept for execution. During this workload filtering, the execution schedules of any accepted instances of *timing-graphs* and *throughput-apps* are also decided. For an accepted *throughput-app* instance, the execution schedule is dynamically deduced in steps 8–11 by always allocating the parallel phase of each instance with the lowest possible number of small cores just sufficient to finish the instance before its deadline. In this way the system's execution effort can be distributed as evenly as possible (i.e., load balanced) over time to avoid spikes in the number of busy or idle cores.

While parallelizable phases of a *throughput-app* instance are executed on the small-core-cluster with the least number of cores clocked at an energy-efficient frequency in the NTC region, sequential phases are prioritized to steal big cores from *timing-graph* instances (steps 9–10), leaving more time to spread execution effort in the small-core-cluster evenly over parallelizable phases. This improves energy-efficiency of the system in two ways: *(i)* an even execution scheduling minimizes the number of small cores required for each parallel phase, reducing multithreading energy-overhead which typically increases with thread count [48]; and *(ii)* as this scheduling method distributes energy consumption of multithreaded applications more evenly across multiple schedule windows, timing-centric task graphs in these windows also tend to get a more even energy budget among them, resulting in better overall energy-efficiency. In prior work, strategies to create an even execution schedule have been shown to result in high energy efficiency for systems with DVFS capability, due to the well-known convex power-frequency relationship [49]. For the same reason, our scheduler does not consider shutting down cores as energy saved will not justify the efficiency loss of the resulting uneven schedule. When a sequential phase of *throughput-apps* steals a big core, a new schedule template for *timing-graphs* is selected to execute with one less core available (step 13). Lastly, if there is energy left over after handling all *throughput-app* instances, we repeat schedule template selection for *timing-graphs,* to fully utilize the energy budget for the current schedule window (steps 17–19).

6. Schedule template generation

As discussed in Section 4, our proposed mixed-criticality scheduling framework (McSF) relies on schedule templates generated at design-time to guide the execution of *timing-centric task graphs*. We utilize a design-time method derived from [22] to generate scheduling templates for timing-centric task graphs under different energy budget levels and number of available cores, called *snapshot-learn-and-rewind* (SLR).

Fig. 5 illustrates the SLR method. The main idea in SLR is to perform iterative learning via step-by-step simulation to update a tentative execution schedule with better choices. Each time an energy-inefficient event is detected, SLR makes informed updates to the execution schedule and then rewinds for another round of learning. SLR saves time during the template generation process by rewinding to a system snapshot taken previously, jumping over simulation steps not affected by updates in scheduling.

The three main components of SLR are outlined below:

1) Firstly, an initial tentative schedule for SLR is generated based on a given target energy budget level, which drops task graphs with lower penalty densities that cannot be supported by the given energy budget, assuming uniform execution frequency for all task nodes. The generated initial schedule conservatively rules out some obviously sub-optimal portions of the solution space and reserves headroom for the upcoming iterative learning process.
2) During recursive learning, the tentative schedule decides the amount of workload to accept and execution frequencies of task nodes, which are taken by a simulation engine that updates the status of task graphs and processing cores step-by-step to detect possible energy inefficient events (discussed later). At each step, the engine checks core utilization and ready task nodes, to perform *execution order scheduling* (to decide execution order of tasks) and *task node-to-core allocation* (to decide execution core of each task). In *execution order scheduling*, the execution order of task nodes from different task graphs is deduced using a metric called *implicit deadline*, which is defined as the latest time to finish a task node so that it is still possible to finish all remaining task nodes of the parent task graph before its deadline. Thus the earlier the implicit deadline is, the more urgent it is to finish the task node to avoid a deadline miss for the entire task graph. As shown in Fig. 6, implicit deadlines of all task nodes are calculated by using a nested function to back-traverse the entire task graph starting from the end task node, which has its implicit deadline equal to the deadline of its parent task graph. As a result, task nodes with earlier implicit deadlines should be scheduled first for execution. In *task node-to-core allocation*, we use a heuristic based on a first-fit decreasing algorithm for the bin-packing problem [50], which sorts task nodes in decreasing order of their execution times and then iteratively allocates the task node with highest execution times to cores with lowest accumulated workload, for execution.
3) Lastly, the core of the SLR heuristic is a snapshot-based iterative learning method, as shown in Fig. 5. Every time a new task graph arrives for execution, SLR saves a snapshot of the current system status so that the simulation can always rewind to this snapshot that was saved before the schedule for the new task graph takes effect. During step-by-step simulation, two types of energy inefficient events may be detected due to sub-optimal decisions in tentative schedule: *(i) task node deadline miss*, which is caused

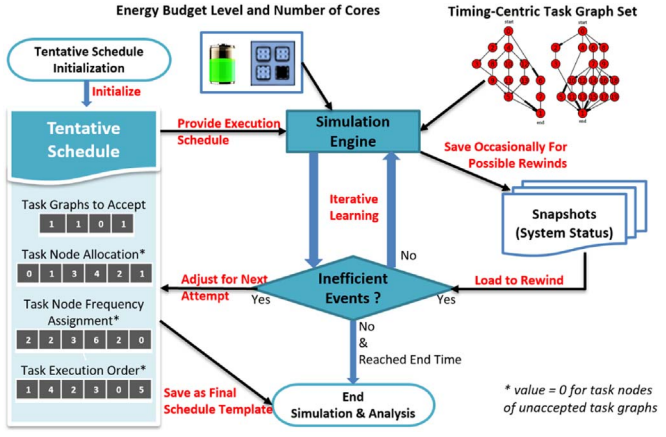


Fig. 5. Snapshot-learn-and-rewind (SLR) technique for generating schedule templates for timing-centric task graphs.

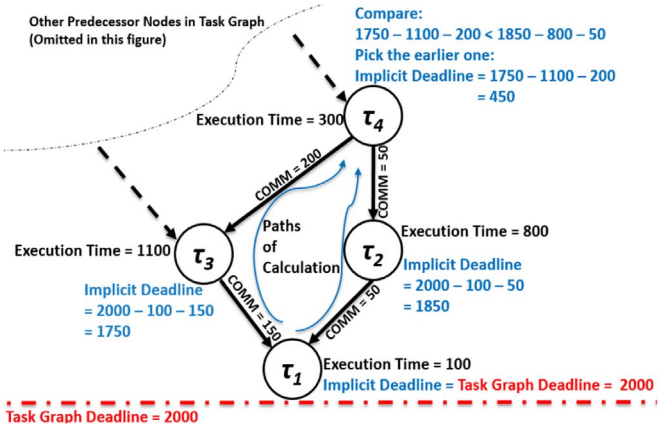


Fig. 6. An example of priority (implicit deadline) assignment.

by conservative frequency assignment in the tentative schedule and is addressed by boosting frequency for nodes along the corresponding critical path; and *(ii) energy depletion*, due to higher than supportable workload accepted by the tentative schedule, which is addressed by dropping the task graph with lowest penalty density. When simulation steps finish without any energy inefficient events, SLR concludes and the updated schedule is saved as a schedule template for the specified energy budget and number of cores.

At design time, the SLR heuristic is executed multiple times with different energy budget levels and number of cores to generate a set of schedule templates for the run-time scheduler (described in Section 5) to select from.

7. Experimental results

7.1. Experiment setup

Our experiments use real-world energy harvesting profiles based on historical weather data provided by the Measurement and Instrumentation Data Center (MIDC) of the National Renewable Energy Laboratory (NREL) [51]. We evaluate system performance over a span of 750 min, from 6:00 A.M. to 6:30 P.M. in a day. We assume peak energy harvesting power to be equal to maximum power required by the system to execute all workload instances.

For timing-centric task graphs, we select applications from various domains: *automotive, networking*, and *telecom* from the Embedded System Synthesis Benchmark Suite (E3S) [36]. Task graphs are assigned with periods uniformly distributed from 10 to 60 s. For throughput-centric multithreaded applications, we select a set of barrier-synchronized parallel applications, including *fft*, *cholesky*, *bodytrack*, *vips*, and *blackscholes*, from the SPLASH-2 [37] and PARSEC [38] benchmark suites, which have different degrees of parallelism (4, 8, or 16), periods (in range of 10–20 min), and *(m, k)-soft* constraints (m in range of 1–3, k in range of 5–10) assigned.

To acquire power and performance metrics for mixed-criticality workloads on different types of cores, we use Sniper [39], an ×86 multicore simulator, and the McPAT [40] power model extended to support V_{dd} in the NTC region for the 22 nm technology node. Table 2 shows the configuration of our platform with big-core-clusters and small-core-clusters. For intra-cluster transfers, a 2D-mesh network-on-chip (NoC) and XY routing over conflict-free TDMA virtual channels is assumed. For inter-cluster communication, we assume delay in the range of hundreds of microseconds to cross clusters [41].

To fit with near-threshold computing's main objective, of achieving high power efficiency, we assumed aggressive low-voltage/power techniques at both device and architecture level to be adapted in our target computing platform. In [43] the tunnel field-effect transistor (TFET) was modeled and simulated, and shown to exhibit benefits with ultra-low operation voltage and low energy-delay product. The authors experimented with a processor critical path abstraction using a ring-oscillator chain of NAND gates and FO4 loads and reported 500 MHz operation frequency with near-threshold voltage as low as 0.3 V. In this paper, we adopt this new device technology and shorter critical path for our cores compared to Intel Atom processors and set the target NTC operation frequency, f^{nth}, to be 500 MHz. Besides, we assumed threshold voltage V_{th} of 0.289 V [42] for the 22 nm technology node and ran multiple iterations of simulation using VARIUS-NTC, a manycore system process variation model extended to NTC in [15], to set the NTC supply voltage V_{dd}^{nth} to 0.4 V, which not only achieves high energy-efficiency but also keeps a safe margin with V_{th} to avoid errors due to the impact of process variations.

Table 2
Configuration of heterogeneous multicore processor.

Architectural Parameters		
Core Types	**Big Cores**	**Small Cores**
Execution	Out-of-Order	In-Order
Issue Width	4	2
Reorder Buffer Size	128	N/A
Cache	64KB, 4-way	16KB, direct
Core Area	15.7 mm^2	4 mm^2
Cluster Parameters		
Cluster Type	**Big-Core-Cluster**	**Small-Core-Cluster**
Core Count	8	32
Frequency Control	Per-Core DVFS	Uniform Frequency
f, V_{dd} Range	0.5–1.2 GHz, 0.4–1 V	f^{nth}, V_{dd}^{nth}
Technology Parameters		
Technology Node	22 nm with TFET	
V_{th}	0.289 V	
V_{dd}^{nth}, f^{nth}	0.4 V, 500 MHz	

7.2. Design-time template generation analysis

Our mixed-criticality scheduling framework (McSF) executes timing-centric task graph applications based on schedule templates generated at design time using the snapshot-learn-and-rewind method (SLR). With scheduling granularity set to 1 ms, our first set of experiments compared SLR with a mixed integer linear programing (MILP) approach similar to [44], which generates optimal solutions mathematically. As shown in Table 3, although the MILP approach generates optimized schedule templates, we found it to be not scalable for larger problem sizes in terms of both template generation time and memory footprint, which may not be practical even at design time. In contrast, the SLR approach stays viable for large problem sizes with a slight sacrifice in solution optimality.

For McSF, a total of 99 schedule templates are generated with different energy budget levels evenly distributed from none to sufficient for different number of (big) cores ranging from 0 to 8. The per-schedule-window miss penalties of the generated template set are shown in Fig. 7, which shows decreasing penalty when more energy budget and cores are made available for an execution schedule. It should be noted that some templates are ignored in our scheduling, e.g., those highlighted in the upper-left and bottom-right regions of Fig. 7 enclosed by blue lines. For example, for the 2 core case, looking at the highlighted region on the bottom-right, increasing the energy budget level beyond 3 does not reduce miss penalty. Similarly, for energy budget level 2, increasing the number of cores beyond 5 does not improve miss penalty. Thus templates in these two regions can be safely ignored.

7.3. Timing intensity metric evaluation

Our next set of experiments tested if our proposed timing intensity metric (from Section 5.1) can accurately characterize the importance of application instances with respect to the (m,k) constraint in a mixed-criticality workload. Suppose that the proposed timing intensity metric was inaccurate, then an alternative timing intensity metric could return

Table 3
MILP and SLR on task graph sets with different sizes.

Method	**Task Graph Set Complexity**		**Memory Footprint**	**GenerationTime**	**Total Miss Penalty**
	Num. of Nodes	**Num. of Edges**			
SLR	36	44	42 MB	0.1 h	1797
MILP			257 MB	6.5 h	1770
SLR	150	193	61 MB	1 h	1695
MILP			7693 MB	492 h	1660

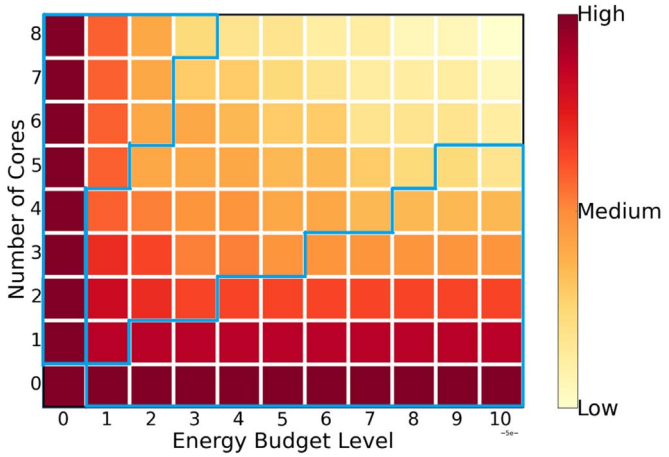


Fig. 7. Miss penalties for generated schedule templates.

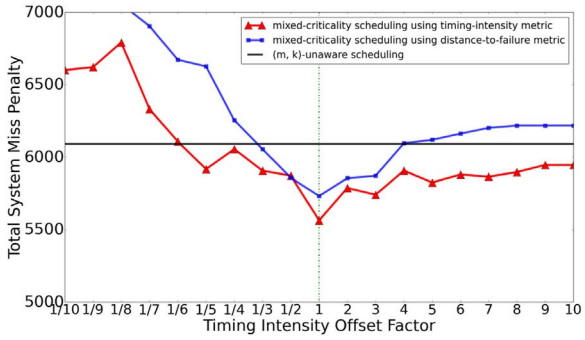


Fig. 8. System miss penalties under different intensity scale factors.

a better result that has less miss penalty in total. To mimic alternative metrics, we offset the resulting timing intensity values with different offset factors ranging for 1/10 to 10. In our experiments for each offset factor, timing intensity values calculated for instances of throughput-centric multithreaded applications were multiplied by each offset factor. For timing-centric task graph instances, the timing intensity was still fixed to 1 by definition. The results in Fig. 8 (red line) show that keeping the original calculated timing intensity (of 1) minimizes overall system miss penalty, while offsetting timing intensity to higher or lower values leads to more miss penalties. Thus it can be concluded that our defined timing intensity metric can accurately evaluate importance of instances to achieve the best balance between throughput-centric and timing-centric tasks to minimize miss penalty of the entire mixed-criticality workload.

We also compared our timing intensity metric with the *distance-to-failure* metric proposed in Hamdaoui's work [9], which also finds its peak when no offset is applied (Fig. 8). We found that it results in up to 9.5% higher miss penalties, compared to our timing intensity-based priority assignment method, as the distance-to-failure metric only considers the next nearest timing failure in the worst case. Besides, both metric evaluation methods outperform the *(m,k)*-unaware scheduling method that assumes firm deadlines for all application instances (see black horizontal line in Fig. 8).

7.4. McSF performance evaluation

As ours is the first framework to address the scheduling and allocation problem for mixed-criticality workloads on heterogeneous systems powered by energy harvesting, there is no prior work to directly compare the overall system performance against. However, we did adapt the performance impact estimation (*PIE*) methodology as an exemplar state-of-art thread scheduling technique for single-ISA heterogeneous systems from [35] (even though it does not support

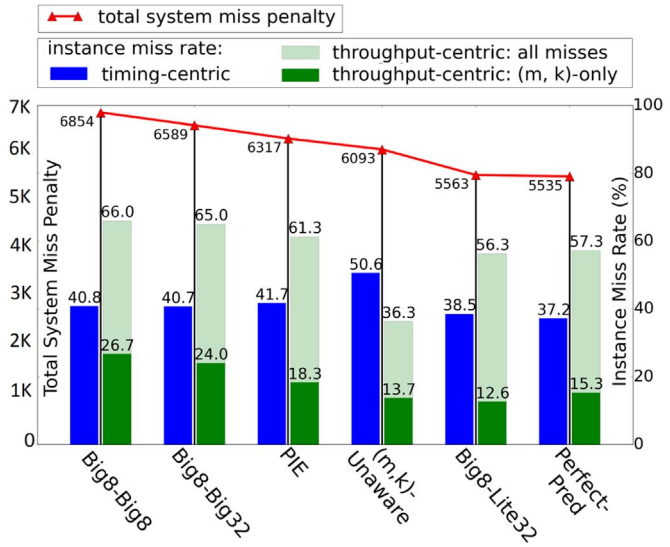


Fig. 9. Miss penalties and instance miss rates across configurations.

energy harvesting). To fit into the experimental setup of this paper, our version of *PIE* estimates the performance benefit of mapping each phase in throughput-centric applications to big cores and the scheduler dynamically selects one phase with the most benefit to share bigger cores with timing-centric task graphs for each schedule window.

We additionally compare the performance of our proposed mixed-criticality scheduling framework (McSF) across four different setups: 1) *Big8-Lite32*, the default configuration with 8 big cores and 32 small cores, which adapts the *moving average* solar energy prediction method used in Liu's work [3]; 2) *Perfect-Pred*, a setup with identical core configuration as the default one, but with the assumption of perfect energy harvesting prediction; 3) *Big8-Big32*, a configuration that replaces the default 32 small cores with 32 big ones; and 4) *Big8-Big8*, a configuration that replaces the default 32 small cores with 8 big cores, to keep overall area footprint the same as *Big8-Lite32* (Table 2). Fig. 9 shows the results of our comparison study. For throughput-centric applications the total miss rate (*throughput-centric: all*) represents all the instances that are dropped. However, because of the (*m,k*)-soft deadline constraint in these applications, some dropped instances do not violate the constraint. Therefore the effective miss rate (*throughput-centric: (m, k)-only*) is much lower.

From Fig. 9, it can be observed that the default *Big8-Lite32* configuration only suffers slight increase in system miss penalty compared to *Perfect-Pred* that has ideal energy prediction, showing the ability of McSF to mitigate the performance impact of energy harvesting mispredictions. Compared to *Perfect-Pred*, *Big8-Lite32* has higher miss rate for timing-centric tasks graph instances and lower miss rate for multithreaded application instances. This is because *Big8-Lite32* accepts higher than optimal multithreaded application instances, without awareness of hard-to-predict instantaneous drops in harvesting power. Then our dynamic workload filtering framework (Section 5.2) allocates fewer resources to timing-centric task graphs to sustain the energy supply for those extra throughput-centric instances already in execution to minimize energy wasted due to misprediction. As a result, miss penalty increases slightly because the balance between the two types of workloads is affected during this process.

For the *(m, k)-Unaware* setup, which utilizes the same core-configuration as *Big8-Lite32* but has no awareness of *(m, k)* constraints, the result shows much lower total miss rate for throughput-centric instances as this approach considers all instances as necessary for penalty avoidance. However, the actual *(m, k)*-miss rate increases as *(m, k)-Unaware* allocates energy to instances that are less important for *(m, k)* constraints. Besides, it also leads to higher miss rate for timing-centric task graphs as the balance between the two types of

workload is notably affected. Thus *(m, k)-Unaware* has higher overall system miss penalty compared to *Big8-Lite32*.

Fig. 9 also shows a comparison with the performance impact estimation (PIE) scheduling and allocation framework from prior work [35] for thread scheduling in single-ISA heterogeneous systems, It can be seen that *PIE* has 13.6% higher miss rate and penalty compared to *Big8-Lite32*, *as it does not focus on energy efficiency but rather on throughput performance, causing more workload to be allocated to big cores for less overall efficiency*.

Comparing *Big8-Lite32* with *Big8-Big32*, it can be seen that although *Big8-Big32* provides better computing capability, it leads to much higher overall miss penalty due to a decrease in energy efficiency. On average, big cores bring performance speedup of approximately 3×, with an average jump of 7× in power consumption, ending up with a 2× degradation in energy efficiency. Moreover *Big8-Big32* also has a much higher area footprint than *Big8-Lite32*, given that the area of big cores is close to 4× that of small cores.

It is also interesting to note the results for *Big8-Big8,* which is a multicore configuration with the same chip area footprint as *Big8-Lite32*. *Big8-Big8* suffers even higher miss penalty than *Big8-Big32,* as it not only has lower energy efficiency than *Big8-Big32* but also possesses lower computation throughput than *Big8-Big32*. *Big8-Lite32* outperforms *Big8-Big8* by 23.2% miss penalty reduction, *highlighting the importance of core heterogeneity to improve energy-efficiency and performance in energy-constrained multicore computing platforms*.

8. Conclusions

In this paper, we addressed the scheduling problem for single-ISA heterogeneous multicore processors running hybrid mixed-criticality workloads with a limited and fluctuating energy budget provided by solar energy harvesting. We modeled a mixed-criticality workload by combining timing-centric real-time task graphs with firm deadlines and throughput-centric multithreaded applications with soft deadlines, with different associated miss penalties. We utilized a single-ISA heterogeneous platform design to fulfill requirements for this mixed-criticality workload. To achieve a balance that minimizes overall system miss penalty, we proposed a novel timing intensity estimation method, based on which we can allocate resources dynamically to different types of workloads according to energy harvesting conditions. In our experiments, the proposed mixed-criticality scheduling framework was shown to achieve a 9.5% miss penalty reduction with the proposed timing intensity metric compared to metrics from prior work, a 13.6% performance improvement over a state-of-the-art scheduling approach for single-ISA heterogeneous platforms, and a 23.2% performance benefit from exploiting platform heterogeneity.

Acknowledgements

This research is supported in part by grants from NSF (CCF-1252500, ECCS-1646562), and endowments from the Monfort Family and Rockwell-Anderson foundations.

References

[1] Chao Li, Wangyuan Zhang, Chang-Burm Cho, Tao Li, SolarCore: solar energy driven multicore architecture power management, in: Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA '11). IEEE Computer Society, Washington, DC, USA, 205-216.

[2] C. Moser, D. Brunelli, L. Thiele, L. Benini, Lazy scheduling for energy-harvesting sensor nodes, Model-Driven Design to Resource Management for Distributed Embedded Systems 2006, Springer US, 2006, pp. 125–134.

[3] Shaobo Liu, Jun Lu, Qing Wu, Qinru Qiu, Harvesting-aware power management for real-time systems with renewable energy. IEEE Trans. Very Large Scale Integr. Syst. 20, 8, August 2012, 1473–1486, 2012.

[4] Jun Lu and QinruQiu. Scheduling and mapping of periodic tasks on multi-core embedded systems with energy harvesting, in: Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC '11). IEEE Computer Society, Washington, DC, USA, 1-6.

[5] Yukan Zhang, Yang Ge, Qinru Qiu, Improving charging efficiency with workload scheduling in energy harvesting embedded systems, in: Proceedings of the 50th Annual Design Automation Conference (DAC '13). ACM, New York, NY, USA, Article 57, 8 pages, 2013.

[6] Yi Xiang, Sudeep Pasricha, Run-time management for multicore embedded systems with energy harvesting. IEEE Trans. Very Large Scale Integr. March 2015, 2015.

[7] Raimund Kirner, Ingredients for the specification of mixed-criticality real-time systems, in: Proceedings of the 2014 IEEE 17th International Symposium on Object/Component-Oriented Real-Time Distributed Computing (ISORC '14). IEEE Computer Society, Washington, DC, USA, 269-275, 2014.

[8] Lui Sha, Resilient mixed-criticality systems, Cross.: J. Def. Softw. JDS (2009) (2009).

[9] M. Hamdaoui, P. Ramanathan, A dynamic priority assignment technique for streams with (m, k)-firm deadlines, IEEE Trans. Comp. 44 (2012) 1443–1451 (12, December 2012).

[10] Peter Greenhalgh. big. LITTLE processing with ARM cortex-a15 and cortex-a7. ARM White Paper 2011, 2011.

[11] Nvidia. The benefits of multiple CPU cores in mobile devices. Retrieved from ⟨http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf⟩.

[12] MediaTek Helio-x30. ⟨https://www.anandtech.com/show/11166/mediatek-helio-x30-10-cores-on-10nm⟩, 2017.

[13] RonaldG. Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, Trevor Mudge, Near-threshold computing: reclaiming moore's law through energy efficient integrated circuits, in: Proceedings of the IEEE. vol. 98, no. 2, February, 253-266, 2010.

[14] David Fick, RonaldG. Dreslinski, Bharan Giridhar, Gyouho Kim, Sangwon Seo, Matthew Fojtik, Sudhir Satpathy, Yoonmyung Lee, Daeyeon Kim, Nurrachman Liu, Michael Wieckowski, Gregory Chen, Trevor Mudge, David Blaauw, Dennis Sylvester, Centip3De: a 3930DMIPS/W configurable near-threshold 3D system with 64 ARM Cortex-M3 cores, in: Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC '12). IEEE Computer Society, Washington, DC, USA, 190-192, 2012.

[15] UlyaR. Karpuzcu, KrishnaB. Kolluru, NamSung Kim, Josep Torrellas, VARIUS-NTV: a microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages, in: Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '12). IEEE Computer Society, Washington, DC, USA, 1-11, 2012.

[16] TimothyN. Miller, Xiang Pan, Renji Thomas, Naser Sedaghati, Radu Teodorescu, Booster: reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips, in: Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA '12). IEEE Computer Society, Washington, DC, USA, 1-12, 2012.

[17] UlyaR. Karpuzcu, Abhishek Sinkar, NamSung Kim, Josep Torrellas, EnergySmart: toward energy-efficient manycores for Near-Threshold Computing, in: Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA '13). IEEE Computer Society, Washington, DC, USA, 542-553, 2013.

[18] UlyaR. Karpuzcu, Ismail Akturk, NamS. Kim, Accordion: toward soft near-threshold voltage computing, in: Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA '14). IEEE Computer Society, Washington, DC, USA, 72-83, 2014.

[19] Maryline Chetto, Optimal scheduling for real-time jobs in energy harvesting computing systems. IEEE Trans. Emerg. Topics in Comput. 2, 2, June 2014, 122-133, 2014.

[20] Daming Zhang, Yongpan Liu, Xiao Sheng, Jinyang Li, Tongda Wu, ChunJason Xue, Huazhong Yang, Deadline-aware task scheduling for solar-powered nonvolatile sensor nodes with global energy migration, in: Proceedings of the 52nd Annual Design Automation Conference (DAC '15). ACM, New York, NY, USA, Article 126, 6 pages, 2015.

[21] Yi Xiang, Sudeep Pasricha, Harvesting-aware energy management for multicore platforms with hybrid energy storage, in: Proceedings of the 23rd ACM International Conference on Great lakes symposium on VLSI (GLSVLSI '13). ACM, New York, NY, USA, 25-30, 2013.

[22] Yi Xiang, Sudeep Pasricha, Fault-aware application scheduling in low-power embedded systems with energy harvesting, in: Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES '14). ACM, New York, NY, USA, Article32, 10 pages, 2014.

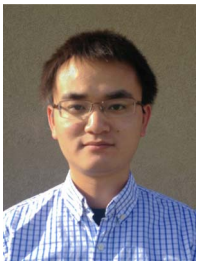
[23] Hu Chen, Dieudonne Manzi, Sanghamitra Roy, Koushik Chakraborty, Opportunistic turbo execution in NTC: exploiting the paradigm shift in performance bottlenecks, in: Proceedings of the 52nd Annual Design Automation Conference (DAC '15). ACM, New York, NY, USA, Article 63, 6 pages, 2015.

[24] Steve Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time tssurance, in: Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07). IEEE Computer Society, Washington, DC, USA, 239-243, 2007.

[25] MalcolmS. Mollison, JeremyP. Erickson, JamesH. Anderson, SanjoyK. Baruah, JohnA. Scoredos, Mixed-criticality real-time scheduling for multicore systems, in: Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology (CIT '10). IEEE Computer Society, Washington, DC, USA, 1864–1871, 2010.

[26] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele, Scheduling of mixed-criticality applications on resource-sharing multicore systems, in: Proceedings of the Eleventh ACM International Conference on Embedded Software (EMSOFT '13). IEEE Press, Piscataway, NJ, USA, Article 17, 15 pages, 2013.

[27] Prabhat Kumar Saraswat, Paul Pop, Jan Madsen, Task migration for fault-tolerance in mixed-criticality embedded systems, SIGBED Rev. (2009) (6, 3, Article 6, October 2009, 5 pages).
[28] Pengcheng Huang, Hoeseok Yang, Lothar Thiele, On the scheduling of fault-tolerant mixed-criticality systems, in: Proceedings of the 51st Annual Design Automation Conference (DAC '14). ACM, New York, NY, USA, Article 131, 6 pages, 2014.
[29] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, Lothar Thiele. Run and be safe: mixed-criticality scheduling with temporary processor speedup, in: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15). San Jose, CA, USA, 1329–1334.
[30] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele. Mapping mixed-criticality applications on multi-core architectures, in: Proceedings of the conference on Design, Automation & Test in Europe (DATE '14). European Design and Automation Association, 3001 Leuven, Belgium, Belgium, Article 98, 6 pages.
[31] Chuancai Gu, Nan Guan, Qingxu Deng, Wang Yi. Partitioned mixed-criticality scheduling on multiprocessor platforms, in: Proceedings of the Conference on Design, Automation & Test in Europe (DATE '14). European Design and Automation Association, 3001 Leuven, Belgium, Belgium, Article 292, 6 pages.
[32] Shin-Haeng Kang, Hoeseok Yang, Sungchan Kim, Iuliana Bacivarov, Soonhoi Ha, Lothar Thiele. Reliability-aware mapping optimization of multi-core systems with mixed-criticality, in: Proceedings of the conference on Design, Automation & Test in Europe (DATE '14). Leuven, Belgium, Article 327, 4 pages.
[33] Domitian Tamas-Selicean, Paul Pop, Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures, in: Proceedings of the 2011 IEEE 32nd Real-Time Systems Symposium (RTSS '11). IEEE Computer Society, Washington, DC, USA, 24-33, 2011.
[34] Salvador Trujillo, Alfons Crespo, Alejandro Alonso, MultiPARTES: Multicore virtualization for mixed-Criticality systems, in: Proceedings of the Conference on Digital System Design (DSD '13). Los Alamitos, CA, USA, 260-265, 2013.
[35] KenzoVan Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, Joel Emer, Scheduling heterogeneous multi-cores through Performance Impact Estimation (PIE), in: Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12). IEEE Computer Society, Washington, DC, USA, 213-224, 2012.
[36] Dick Robert. Embedded system synthesis benchmarks suites (E3S) Retrieved from ⟨http://ziyang.eecs.umich.edu/~dickrp/e3s/⟩.
[37] StevenC. Woo, Moriyoshi Ohara, Evan Torrie, JaswinderPal Singh, Anoop Gupta, The SPLASH-2 programs: characterization and methodological considerations, in: Proceedings of the 22nd Annual International Symposium on Computer architecture (ISCA '95). ACM, New York, NY, USA, 24-36, 1995.
[38] Christian Bienia, Benchmarking Modern Multiprocessors. Ph.D. Dissertation. Princeton University, Princeton, NJ, USA. Advisor(s) Kai Li. AAI3445564, 2011.
[39] TrevorE. Carlson, Wim Heirman, Lieven Eeckhout, Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11). ACM, New York, NY, USA, Article52, 12 pages, 2011.
[40] Sheng Li, JungHo Ahn, RichardD. Strong, JayB. Brockman, DeanM. Tullsen, NormanP. Jouppi, McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures, in: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '09). ACM, New York, NY, USA, 469-480, 2009.
[41] Matthew DeVuyst, Ashish Venkat, DeanM. Tullsen, Execution migration in a heterogeneous-ISA chip multiprocessor, in: Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12). ACM, New York, NY, USA, 261-272, 2012.
[42] A.H. Maheran, P.S. Menon, I. Ahmad, Z. Yusoff, Threshold voltage optimization in a 22nm high-k/silicide PMOS device, in: Proceedings of the 2013 IEEE Regional Symposium on Micro and Nanoelectronics (RSM '13). IEEE Computer Society, Washington, DC, USA, 126-129, 2013.
[43] Vinay Saripalli, Device and architecture co-design for ultra-low power logic using emerging tunneling-based devices. Vinay Saripalli, Device and architecture co-design for ultra-low power logic using emerging tunneling-based devices. Ph.D. Dissertation. Pennsylvania State University, Dec, 2011, 2011.. Pennsylvania State University, Dec, 2011, 2011.
[44] Vivy Suhendra, Chandrashekar Raghavan, Tulika Mitra, Integrated scratchpad memory optimization and task scheduling for MPSoC architectures, in: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06). ACM, New York, NY, USA, 401-410, 2006.
[45] MeetaS. Wonyoung Kim, Gu-yeon Gupta, Wei, David Brooks, System level analysis of fast, per-core DVFS using on-chip switching regulators, in: Proceedings of the 2008 IEEE 14th International Symposium on High Performance Computer Architecture (HPCA '08). IEEE Computer Society, Washington, DC, USA, 123-134, 2008.
[46] Mummadi Veerachary, Tomonobu Senjyu, Katsumi Uezato, Maximum power point tracking of coupled inductor interleaved boost converter supplied PV system, in: Proceedings of the IEE Proceedings Electric Power Applications (EPA '03). IEEE ComputerSociety, Washington, DC, USA, 71-80, 2003.
[47] Yu-Kwong Kwok, Ishfaq Ahmad. Benchmarking the task graph scheduling algorithms, in: Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium (IPPS '98). IEEE Computer Society, Washington, DC, USA, 531-537.
[48] Vince Major Bhadauria, Weaver, SallyA. McKee, A characterization of the PARSEC benchmark suite for CMP design, Technical Report, Cornell University, Sep 2008, 2008.
[49] Changjiu Xian, Yung-Hsiang Lu, Zhiyuan Li, Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time, in: Proceedings of the 44th Annual Design Automation Conference (DAC '07). ACM, New York, NY, USA, 664-669, 2007.
[50] Binzhou Xia, Zhiyi Tan, Tighter bounds of the first fit algorithm for the bin-packing problem, Discret. Appl. Math. 158 (15) (2010) 1668–1675 (August 2010).
[51] NREL. NREL measurement and instrumentation data center (MIDC). Retrieved from ⟨http://www.nrel.gov/midc/⟩.
[52] ARM. Cortex-R52 processor overview. Retrieved from ⟨http://developer.arm.com/products/processors/cortex-r/cortex-r52⟩.
[53] Elif Mugan, Chao Lu, Vijay Raghnathan, Kaushik Roy, Modeling, design and cross-layer optimization of polysilicon solar cell based micro-scale energy harvesting systems, in: proceedings of the 2012 ACM/IEEE international Symposium on Low Power Electronics and Design (ISLPED '12). Redondo Beach, CA, USA, 123-128, 2012.
[54] Chao Lu, Vijay Raghunathan, Kaushik Roy, Efficient design of micro-scale energy harvesting systems, IEEE J. Emerg. Sel. Top. Circuits Syst. 1 (3) (2011) 254–266.
[55] Hui Shao, Chi-Ying Tsui, Wing-Hung Ki, The design of a micro power management system for applications using photovoltaic cells with the maximum output power control. IEEE Transactions on very large scale, Integr. (VLSI) Syst. 17 (8) (2009) 1138–1142.



Yi Xiang (S'11) was born in Suining, China, in 1987. He received the B.S. degree in microelectronics from the University of Electronic Science and Technology of China, Chengdu, China, in 2010 and his Ph.D. degree with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA in 2016. He is currently a senior R & D engineer at Synopsys. His research interests include low-power computing, parallel embedded systems, heterogeneous computing, and CAD algorithms.

Sudeep Pasricha (M'02, SM'13) received his Ph.D. degree in computer science from the University of California, Irvine in 2008. He is currently a Monfort and Rockwell-Anderson Associate Professor of Electrical and Computer Engineering at Colorado State University, Fort Collins. His research interests include energy efficiency and fault tolerance for multicore computing. Dr. Pasricha is in the editorial board of various journals such as TCAD, TMSCS, TECS, etc. He currently is or has been an Organizing Committee Member and/or TPC member of various conferences such as DAC, DATE, ESWEEK, NOCS, VLSID, GLSVLSI, etc. He has received Best Paper Awards at SLIP'16, GLSVLSI'15, AICCSA'11, ISQED'10 and ASPDAC'06.