Run-Time Management for Multicore Embedded Systems With Energy Harvesting

Yi Xiang, *Student Member, IEEE*, and Sudeep Pasricha, *Senior Member, IEEE*

***Abstract*— In this paper, we propose a novel framework for runtime energy and workload management in multicore embedded systems with solar energy harvesting and a periodic hard real-time task set as the workload. Compared with prior work, our framework makes several novel contributions and possesses several advantages, including the following: 1) a semidynamic scheduling heuristic that dynamically adapts to runtime harvested power variations without losing the consistency of periodic tasks; 2) a battery–supercapacitor hybrid energy storage module for more efficient system energy management; 3) a coarse-grained core shutdown heuristic for additional energy saving; 4) energy budget planning and task allocation heuristics with process variation tolerance; 5) a novel dual-speed method specifically designed for periodic tasks to address discrete frequency levels and dynamic voltage/frequency scaling switching overhead at the core level; and 6) an extension to prepare the system for thermal issues arising at runtime during extreme environmental conditions. The experimental studies show that our framework results in a reduction in task miss rate by up to 70% and task miss penalty by up to 65% compared with the best known prior work.**

***Index Terms*— Dynamic voltage and frequency scaling, energy harvesting, multicore processing, scheduling algorithm.**

I. Introduction

POWER and energy constraints have led to significant changes in the design of contemporary computing systems. With advances in parallel programming and power management techniques, embedded devices with multicore processors are outperforming single-core platforms in performance and energy efficiency [1]. As core counts continue to increase to keep up with rising application complexity, techniques for runtime workload distribution and energy management are the keys to achieve energy savings in emerging multicore embedded systems. Moreover, as CMOS technology scales down to integrate more cores on the same die area, process variations have become prominent, significantly impacting the system-level design and management of multicore chips [2].

For some embedded applications, we may require energy autonomous devices that utilize ambient energy to perform computations without relying entirely on an external power supply or frequent battery charges. Because it is the most widely available energy source, solar energy and its harvesting for embedded systems has attracted a lot of attention in recent years [3]–[5]. Due to the variable nature of solar energy harvesting, deployment of an intelligent runtime energy management scheme is not only beneficial but also essential for meeting system performance, robustness, and energy goals. To exploit the capabilities of energy harvesting systems, several prior efforts have explored workload scheduling for embedded systems with real-time tasks [6]–[9].

In this paper, we propose a novel semidynamic algorithm (SDA)-based framework with energy budgeting that manages energy and workload allocation at runtime for multicore embedded systems with solar energy harvesting capability. The novelty and main contributions of this paper are summarized as follows.

1) Unlike prior work, SDA reacts to runtime energy shortages and fluctuations proactively to find greater scope for energy savings, especially in multicore platforms.
2) A hybrid energy storage system is designed to decouple the runtime management scheme from variations in energy harvesting, as well as to enhance charging/discharging efficiency.
3) The energy and task distribution heuristics in SDA take system heterogeneity into consideration by assigning workloads with an awareness of variations due to within-die process variations.
4) At the core level, a novel dual-speed frequency selection method is deployed to combine two neighboring discrete frequency levels for superior energy efficiency with an awareness of voltage/frequency switching overhead.
5) Our framework cooperates with basic throttling mechanisms to tackle processor overheating. In addition, it dynamically reallocates workload or shuts down cores for more proactive multilevel throttling to reduce the occurrences and overhead of system overheating.

Our experimental studies show that our framework is able to outperform the best known prior work (utilization-based technique (UTB) [9]) on runtime management for real-time systems with energy harvesting, achieving superior task drop penalty/rate reduction and energy efficiency. In addition, our framework also provides the flexibility to adapt to runtime thermal variations and supports core heterogeneity-aware workload distribution.

II. Related Work

Many prior works have focused on the problem of task scheduling for real-time embedded systems. Here, we only discuss the few works that are relevant to our goals.

Manuscript received March 8, 2014; revised September 12, 2014; accepted December 1, 2014. Date of publication March 18, 2015; date of current version November 20, 2015. This work was supported in part by the National Science Foundation under Grants CCF-1252500 and CCF-1302693 and in part by the Semiconductor Research Corporation and the China Scholarship Council.

The authors are with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523 USA (e-mail: yix@colostate.edu; sudeep@colostate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2381658

An early work [12] addressed the problem of power-aware scheduling of periodic hard real-time tasks using dynamic voltage/frequency scaling (DVFS). This work proved that an optimal execution frequency to minimize energy consumption while meeting all task deadlines can be deduced for any periodic hard real-time policy that can fully utilize the processor [e.g., earliest deadline first (EDF), least laxity first]. In [13], algorithms were proposed that minimized energy consumption and task rejection penalty for systems with homogeneous multiprocessor and DVFS support. A Li-ion battery–supercapacitor hybrid storage system that supports a long lifetime and a photovoltaic (PV)-based wireless sensor network was described in [20], presenting a good example of hybrid energy system design, from which we derived a customized hybrid storage system in this paper. In [14], the hypoenergy framework was proposed to extend power supply life time of hybrid battery–supercapacitor systems. An algorithm for application scheduling and power management of chip multiprocessors with an awareness of within-die processor variations was proposed in [15]. In [16], a thermal-aware task allocation and scheduling algorithm was proposed, which was used as a subroutine for hardware/software cosynthesis. None of these works consider the challenges arising from energy harvesting in real-time embedded systems.

Solar energy harvesting is becoming an increasingly attractive paradigm to obtain clean sustainable energy for emerging embedded systems. Recently, a few works have explored improvements in the efficiency and reliability of such systems [3], [4], [17]. Some of these works focus on the implementation of energy harvesting systems and their energy conversion circuits [17]. Another focus in this area is on runtime management and scheduling techniques for real-time embedded system with energy harvesting. An early work [6] proposed the lazy scheduling algorithm (LSA) that executed tasks as late as possible, reducing deadline miss rates when compared with the classical EDF algorithm. However, LSA does not consider DVFS and always executes tasks at full speed. Because a processor's dynamic power is generally a convex function of frequency, operating the processor at a frequency lower than the maximum frequency often results in higher energy efficiency. In [7], the energy-aware DVFS (EA-DVFS) algorithm technique was proposed that takes processor DVFS into consideration. EA-DVFS utilized task slack to slow down execution speed, thereby achieving more energy savings than LSA, especially when total task utilization is low. Later, the same authors proposed a more intelligent technique called harvesting-aware DVFS (HA-DVFS) algorithm [8], which improved energy efficiency by distributing multiple arriving tasks as evenly as possible over time and executing them with more uniform frequency. However, these works focus on uniprocessor systems and have not considered execution on multicore platforms.

In [3], a runtime framework is proposed for intelligently adjusting runtime system workload on multicore platforms that use PV array for energy harvesting, so that the array works at its maximum operation points, producing more power for the computation system. However, the proposed work assumes grid utility as a backup energy source which may not be viable for many types of embedded systems, and also their approach is not applicable to real-time embedded systems with deadlines and operating constraints, which is the focus of our work. Recently, a UTB was proposed in [9] to better address periodic task scheduling in energy-harvesting embedded systems. UTB takes advantage of the predictability provided by the periodic task information for more efficient task allocation than in prior work. Moreover, UTB was extended to support multicore platforms by allocating a subset of tasks to each core and executing the single-core UTB algorithm separately on each core.

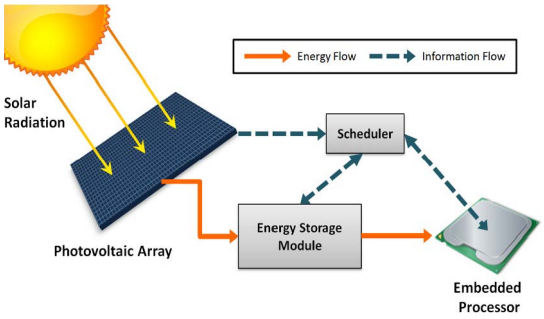


Fig. 1. Real-time embedded processing with solar energy harvesting.

In this paper, we propose an SDA-based framework for runtime management of energy-harvesting-based embedded systems that notably improves upon prior work. Our framework aims to minimize overall task miss rate and penalty for real-time workloads running on multicore systems, with a support for coping with thermal variations and heterogeneity arising due to process variations.

III. Problem Formulation

A. System Model

Our focus in this paper is on the problem of effective workload and energy management for real-time multicore embedded systems running periodic tasks, which are powered by solar energy, as shown in Fig. 1. The following sections describe the key components of our system model.

1) Energy Harvesting and Energy Storage Module: A PV array is used as a power source for our embedded system, converting ambient solar energy into electric power. Naturally, the amount of harvested power varies over time due to changing environmental conditions, like angle of sunlight incidence, cloud density, temperature, and humidity. To cope with the unstable nature of the solar energy source, rechargeable batteries and supercapacitors can be used to buffer solar energy collected by PV cells. The capacity of the energy storage device is limited and harvested energy will be wasted if the energy storage device is already fully charged.

2) Periodic Real-Time Task Workload: In many real-world applications, an energy autonomous embedded system powered by solar energy harvesting is deployed to execute certain types of repetitive lightweight real-time tasks, such as sensing, controlling, and data preprocessing. We assume a task set of N independent periodic real-time tasks $\psi\{\tau_1, \ldots, \tau_N\}$ for such use cases, in which each periodic task τ_i has a characteristic

TABLE I
XSCALE PROCESSOR POWER AND FREQUENCY LEVELS [9]

Level	0	1	2	3	4	5
Voltage(V)	-	0.75	1.0	1.3	1.6	1.8
Power(mW)	40	80	170	400	900	1600
Frequency(MHz)	idle	150	400	600	800	1000
Energy Efficiency	0	1.875	2.353	1.5	0.889	0.625

triplet (C_i, D_i, T_i), $i \in \{1, \ldots, N\}$. C_i is the maximum number of CPU clock cycles needed to finish a job instance of task τ_i, referred to as the worst case execution cycles (WCECs). The relative deadline of the task D_i is the time interval between a job's arrival time and its deadline. A job instance is missed if it is not finished before its deadline. T_i is the period of the task. At the beginning of each period, a new job instance of that task will be dispatched to the system. Like most recent works on periodic task scheduling [9], we assume that D_i equals T_i, with all jobs expected to finish before the arrival of the next job instance of the same task. We also define an attribute X_i, which is the miss penalty associated with each task. Each time that a task's job misses its deadline, the job will be aborted and the penalty applied to the system. Thus, we can refine the triplet for task τ_i as (C_i, T_i, X_i). The relative importance of a task can be characterized by a *penalty density* parameter, defined as the ratio of the task miss penalty and WCEC (X_i/C_i) [13]. In this paper, we assume the system is designed to execute one set of periodic real-time tasks consistently, and information of tasks such as execution time and miss penalty is profiled at design time and, thus, is available to the runtime scheduler.

3) DPM and DVFS-Enabled Multicore Processor: We consider an embedded system with a low power multicore processor that has a support for task preemption. We assume that the frequency of each core can be adjusted individually (i.e., the processor possesses per-core DVFS capability), as observed in recent implementations with this capability enabled in industry and academia [29], [30]. Each core has M discrete voltage and frequency levels: $\varphi\{L_0, \ldots, L_M\}$. Each level is characterized by L_j: (v_j, p_j, f_j), $j \in \{1, \ldots, M\}$, which represents voltage, average power, and frequency, respectively. We consider power-frequency levels of the Xscale processor, as shown in Table I. Here, level 0 represents the idle power of the processor when no task is executed while the system stays in active state. Typically, the dynamic power-frequency function is convex. Thus, a processor running at lower frequency can be expected to execute the same number of cycles with lower energy consumption. However, this is not always the case due to the increasing prominence of leakage power in recent CMOS technologies. To find an energy optimal frequency, we represent energy efficiency of a $v - f$ level L_i by $\delta_i = cycles\ executed/energy\ consumed = f_i/p_i$. From Table I, we can conclude that level 2 is the most energy efficient because executing at this level consumes the least energy for a given number of cycles. The most energy efficient level is often called *critical level* in the literature and thus $f_{\text{crt}} = f_2$ [28]. Although it is desirable to execute tasks at this critical frequency level for energy efficiency, executing tasks at f_{crt} may end up being insufficient to finish all task instances by their deadlines, due to the unique timing constraints of each task. As we also consider intercore heterogeneity caused by within-die process variations, some cores have lower maximum frequency and higher static power values than for the ideal case. For each core, unsupported $v - f$ levels are blocked to ensure system stability.

The utilization of a periodic task (U) is defined with respect to the full speed (maximum frequency) provided by the processor. A task's utilization is its execution time under the maximum frequency divided by its period

$$U_i = \frac{C_i/f_{\max}}{T_i}. \tag{1}$$

The utilization for an entire task set is simply the accumulation of the utilization for all the tasks in the set. In preemptive real-time systems, a task set is schedulable by the EDF algorithm for a frequency j if it meets the following condition:

$$U_{\text{total}} \leq \frac{f_j}{f_{\max}}. \tag{2}$$

When total task set utilization is known, the most energy efficient frequency can be deduced from this equation, assuming $f_j \geq f_{\text{crt}}$ [12].

In addition, unlike any prior work, we consider thermal management in an energy harvesting multicore processing environment. We assume that each core in the multicore processor has a digital thermal sensor (DTS) implemented to monitor runtime temperature independently [21]. We set 85 °C as the thermal *setpoint* at which throttling is initiated to halt all processor executions (i.e., throttling threshold = 85 °C) [18]. When throttling is triggered, a core must halt execution and shift to idle state until its temperature drops to 80 °C.

4) Run-Time Scheduler: This software module is an important component of the system for information gathering and execution control. The scheduler dynamically gathers information by monitoring the energy storage medium and multicore processor state (Fig. 1). The gathered data, together with offline-profiled information about task execution times and energy consumption on cores informs a management algorithm in our scheduler that coordinates operation of the multicore platform at runtime. Each core is eventually assigned a strategy by the scheduler to guide intracore task execution.

B. Scheduling Problem Objective

Our primary optimization objective is to perform task allocation and scheduling at runtime such that total task miss rate (or penalty) is minimized. Our technique must react to changing harvested energy dynamics to complete as much (critical) work as possible, thus maximizing the overall system utility and cost effectiveness. Further, our task allocation should be cognizant of processor thermal behavior and frequency limits of each core (due to process variations) to ensure system stability.

IV. MOTIVATION

A. Motivation for Semidynamic Algorithm

In this section, we provide a motivational example to illustrate the benefits of our SDA framework that integrates energy budgeting to achieve a better workload distribution at

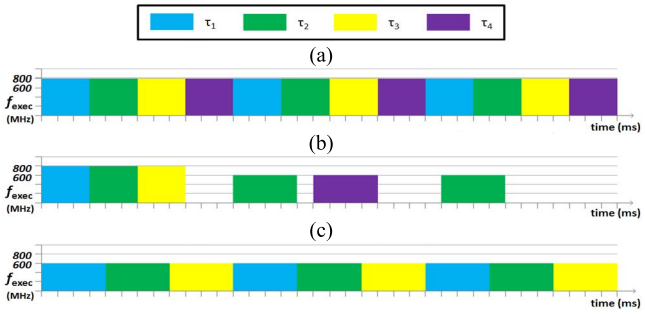


Fig. 2. Motivation for the proposed semidynamic approach. (a) UTB schedule with sufficient energy. (b) UTB schedule with insufficient energy. (c) SDA schedule with insufficient energy.

runtime than in the existing approaches, under varying solar energy harvesting scenarios.

Most prior work deals with dynamic solar energy variations by halting, dropping, or speeding up the execution of a current task, changing instantly from an initial schedule deduced offline. For energy-harvesting-aware periodic task set scheduling, the best known prior work, UTB [9], also follows this strategy. Although UTB deduces an optimal initial schedule offline assuming sufficient energy, it does not cope well with runtime energy variations, and there is a scope for notable improvements, which is discussed as follows.

1) The task dropping mechanism in UTB reacts to runtime energy shortages *passively* only when the current task lacks sufficient energy to finish in time. In the motivational example shown in Fig. 2, we assume a task set with four periodic tasks ($\tau_1 \sim \tau_4$), where each task has WCEC of 2.4 million CPU cycles and a task period of 12 ms. According to Table I and in (1) and (2), UTB initially sets execution frequency to 800 MHz so that all tasks can finish with the best efficiency if energy is sufficient, as shown in Fig. 2(a). However, the real challenge arises when the runtime energy budget is insufficient. Let us assume that the remaining energy in the energy storage is 7200 μJ and harvested power in the next 36 ms (3 periods) is 200 mW, i.e., 200 μJ of incoming energy per microsecond. After finishing three jobs, the energy storage is depleted and UTB has to drop jobs due to insufficient energy, as shown in Fig. 2(b). Only 6 out of 12 job instances are finished with UTB, resulting in a high 50% miss rate. With the same energy budget, our proposed SDA copes with energy shortage by *proactively* dropping tasks. It drops one task, τ_4, based on the energy budget that helps to execute the remaining tasks steadily at a lower frequency of 600 MHz. According to Table I, executing at 600 MHz corresponds to a power consumption of 400 mW, which is dramatically lower than 900 mW at 800 MHz due to the nonlinear relation between frequency and power consumption. As can be observed in Fig. 2(c), all accepted job instances for $\tau_1 \sim \tau_3$ are finished and the overall miss rate is 25%, which is lower than the 50% miss rate achieved by UTB.
2) UTB encourages dropping tasks with longer execution time, because finishing them requires more energy than other tasks. This biased dropping may be undesirable for real-time applications, as tasks with longer execution time may represent complex applications of high priority. Moreover, it is nontrivial to add priority awareness into UTB due to its passive task dropping scheme mentioned above. Our SDA framework allocates tasks and performs task dropping with the awareness of the miss penalty corresponding to each task.
3) On multicore platforms, UTB partitions tasks into separate sets and then executes each set on a core using a single-core scheduling algorithm. However, as all cores are dependent on the same energy source, such isolated runtime adjustment is not amenable to learn upcoming energy requirements of other cores, leading to suboptimal schedules. SDA avoids intercore energy resource contention by allocating tasks based on energy budgets assigned to each core. In addition, static task partitioning in UTB wastes the flexibility provided by a multicore platform. In contrast, SDA triggers task reallocation dynamically for improved results.

In summary, we found several limitations with the best known prior work on energy harvesting-aware energy and workload management. Our SDA scheme is designed to address these limitations and improve upon prior work. In the following sections, we discuss other issues related to multicore embedded systems powered by solar energy harvesting. To cope with these issues, we exploit the flexibility of SDA to integrate hybrid energy storage, heterogeneity-aware task allocation, and runtime thermal management, forming a cross-layer design that improves performance, stability, and adaptivity of target systems.

B. Motivation for Hybrid Energy Storage

Most prior efforts on harvesting-aware task scheduling assume a near-ideal battery as the energy storage medium that is limited merely by its capacity, ignoring other factors such as nonlinear efficiency, slow charge rate, and limited lifetime in terms of recharge cycles [19]. When applied to real-world platforms, overlooking these factors can result in suboptimal or even unrealistic design and scheduling techniques that diminish system efficiency, stability, and lifespan. For example, the rate capacity effect leads to decreasing battery capacity when discharging current increases [14]. Supercapacitors present an interesting alternative to batteries for energy storage with benefits over electrochemical batteries, such as orders of magnitude higher recharge cycles, ease of charging, and significantly higher energy efficiency. However, high-capacity supercapacitors are not practical for small-package low-power embedded systems due to their significantly lower energy density and higher leakage overhead than an electrochemical battery, even with the state-of-art supercapacitor technology [23]. Recent work has shown that a battery–supercapacitor hybrid system can overcome the limitations of both types of energy storage mediums [14], [20]. Therefore, we employ a hybrid energy storage system for this paper.

C. Motivation for Heterogeneity-Aware Workload Allocation

For multicore processors, within-die process variations differentiate critical path delays among cores such that the

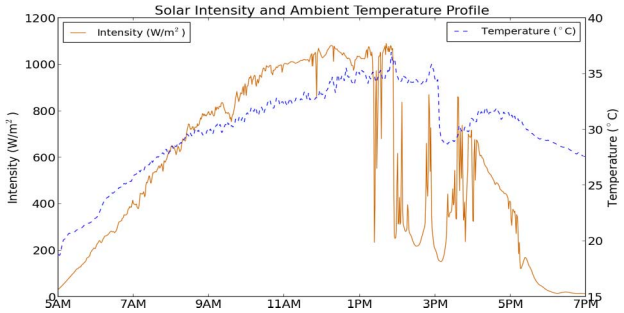


Fig. 3. Example of solar intensity versus ambient temperature.

maximum frequencies supported by cores may diverge from their nominal specification [2]. Without awareness of this undesirable intercore heterogeneity, a runtime management scheme may distribute excessive workload to slower cores. Even worse, faulty schedules that try to finish these excessive workloads will be deployed, ending up with a high miss rate due to energy and CPU time being wasted on tasks that cannot be finished in time. Overclocking slower cores is a possibility, but is often not a viable option due its high likelihood of causing timing violations on the critical path. Thus, an appropriate runtime energy management framework must consider intercore frequency variations; otherwise, it may lower system performance by causing task overloading on certain cores, which can create workload imbalances that also additionally reduce the energy efficiency of the entire system.

D. Motivation for Run-Time Thermal Management

The motivations for considering thermal management for energy-harvesting-based multicore embedded systems are as follows.

1) Limited power budgets and form factors of embedded systems make it uneconomical, if not inapplicable, to apply aggressive cooling techniques used on desktop and server systems, such as cooling fans and large heat sinks. With increasing power density and the absence of active cooling, high-performance multicore embedded processors can easily end up causing thermal emergencies during their long operation periods. Such overheating of processors is known to harm system reliability and stability. A throughput-focused runtime management scheme that ignores this risk may fail to maintain system stability and end up with thermal runaway. Perhaps most importantly, frequent thermal throttling that is initiated in processors to cope with thermal emergencies may end up disrupting balanced scheduling strategies, reducing system performance, and overall energy efficiency.
2) Due to the inherent nature of solar energy, solar energy harvesting systems tend to receive abundant energy to run at full speed around the middle of the day. However, continuously executing at full speed creates excessive heat in the processor package and can lead to overheating issues. Around the same time, the ambient temperature is also usually the highest in the day (Fig. 3), making it even more difficult for the processor to cool down around those hours without intervention.

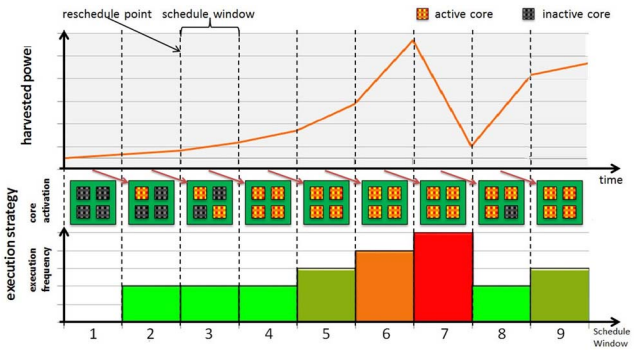


Fig. 4. Illustration of SDA.

Thus, there is a critical need to consider runtime thermal management strategies for energy harvesting based embedded systems as thermal issues can have a notable impact on the performance, energy efficiency, and reliability of such systems.

V. Proposed Run-Time Energy and Workload Management Framework

A. Semidynamic Algorithm Overview

In this section, we present a holistic overview of our novel energy and workload management framework based on an SDA. Subsequent sections present more details of each major component in our SDA-based framework.

One of the underlying ideas behind SDA is to exploit time-segmentation during energy management, as shown in Fig. 4. At each specified time interval (epoch), there is a reschedule point, where the execution strategy can be adjusted based on the energy budget provided by the energy storage system. A time frame between two reschedule points is called a schedule window, within which the strategy specified at the prior reschedule point is in effect until the next reschedule point. Thus, reschedule points provide dynamic adaptivity needed by the energy harvesting aware system to adjust the task execution strategy, while the schedule window enables stable execution that utilizes periodic task information for better energy efficiency, as shown in Fig. 2(c). For example, from schedule window 1 to 4 in Fig. 4, it can be observed that under low energy conditions, SDA maintains execution at an optimal low (critical) frequency with different number of cores activated. Cores only execute at higher frequency when the energy harvested is abundant, as in schedule windows 6 and 7. In this manner, SDA can provide a better execution efficiency to improve performance under variable solar radiance conditions.

At each reschedule point, we update the execution strategy for the upcoming schedule window with a rescheduling scheme composed of three stages.

1) Energy Budgeting: This stage estimates the energy budget available for the upcoming schedule window based on the status of the hybrid energy storage system. Estimating the energy budget decouples runtime system management from energy variations in the environment, making it possible to deduce a stable balanced execution strategy that maximizes energy efficiency.

2) Workload Estimation: This second stage evaluates the amount of workload that can be supported by the energy

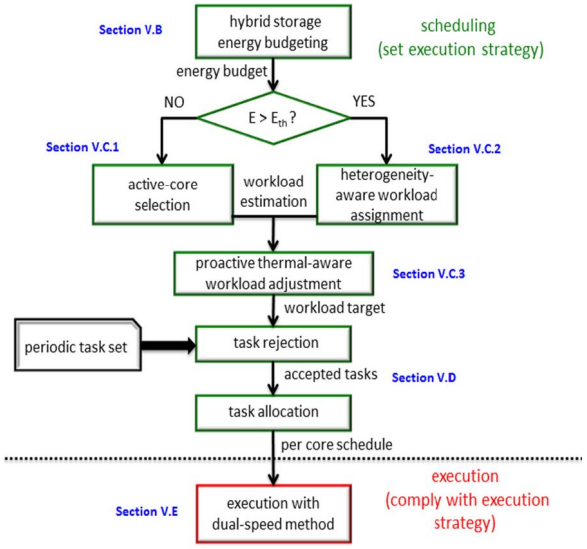


Fig. 5. Design flow of our proposed SDA-based framework.

budget and forks into two separate paths. When energy budget is below a threshold, E_{th}, the first path is chosen with a focus on active-core selection to improve energy efficiency under a low energy budget. When energy budget is above E_{th}, the second path is chosen with a focus on variation-aware workload assignment to ensure that no core is required to run at a frequency higher than its maximum limit. Note that there is no need to consider active core selection and variation-aware assignment at the same time, as maximum frequency variation only matters when the energy budget is high and as active core selection only helps when energy budget is very low (Sections V-C1 and V-C2). In addition, this stage can proactively reduce workload when thermal issues arise at runtime.

3) Task Rejection and Allocation: Based on the amount of workload estimated by the previous stage, this stage takes the periodic task set and filters out the subset of tasks that are less important. The remaining tasks are accepted for execution and are allocated to cores with an awareness of core heterogeneity.

These three stages are organized in an order such that successor stages make use of efforts made by previous stages and are described in Sections V-B–V-D. After the execution strategy is fixed for a schedule window, cores apply a dual-speed switching method to improve energy efficiency in the presence of discrete frequency levels, which is discussed in Section V-E. The complete design flow of our proposed SDA framework is shown in Fig. 5.

B. Hybrid Energy Storage System and Energy Budgeting

In this section, we describe our hybrid energy storage system and its management policy that determines the energy budget for the upcoming schedule window, thereby isolating runtime task scheduling from fluctuations in solar energy harvesting.

1) Battery-Supercapacitor Hybrid Energy Storage: Inspired by [20], we propose a hybrid energy storage system with one Li-ion battery and two separate supercapacitors connected by

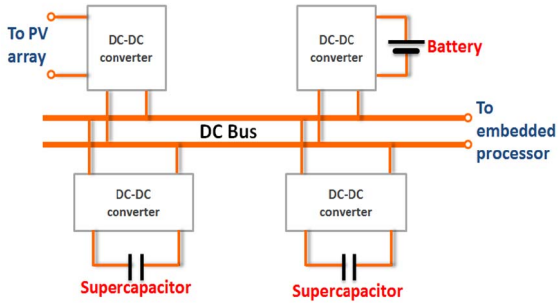


Fig. 6. Proposed hybrid energy storage system.

a dc bus, as shown in Fig. 6. During each schedule window, one capacitor is used to collect energy extracted from the PV array, while the other one is used as a power source for system operation or battery charging. At each reschedule point, the two supercapacitors switch their roles. Supercapacitors charge the battery only when their saved energy exceeds the peak requirements of processors running at full speed. The PV array, battery, and supercapacitors are coupled with bidirectional dc–dc converters to serve the purpose of voltage conversions between components with maximum power point tracking (MPPT) [17] and voltage level compatibility. This hybrid battery and dual-supercapacitor design has several advantages over a nonhybrid system.

1) The supercapacitors can support embedded processors directly, taking advantage of a much lower charging/discharging overhead compared with a battery.
2) The electrochemical battery offers high capacity to preserve energy especially in scenarios with excessive harvested energy. On the other hand, the capacity requirement of supercapacitors is much smaller.
3) The supercapacitor with energy buffered during the last schedule window acts as a known stable energy source for the system in the upcoming schedule window. Thus, our energy budgeting does not require energy harvesting power predication.

2) Hybrid Energy Storage Based Energy Budgeting: We propose a heuristic that selects among energy sources (supercapacitors and battery), sets the amount of energy to charge the battery for (E_{chrg}), and assigns the energy budget for system execution in the upcoming schedule window (E_{budget}), as shown in Algorithm 1. The heuristic is based on storage levels of the battery (LV_B) and supercapacitor (LV_C) with ranges 1–3, representing, respectively, the charge levels of low, medium, and high. LV_C is classified into three levels (lines 1–3) based on two thresholds: 1) energy budget to execute a single core at critical frequency (E_{crt}) and 2) energy budget to execute all cores at maximum frequency ($E_{\text{max}} \times NUM\_CORE$). As we want to avoid battery charging/ discharging overhead, there are only two scenarios where the battery is selected as a power source: 1) when energy harvested in the supercapacitor is below a critical level ($\text{LV}_C = 1$) and 2) when battery storage level is high ($\text{LV}_B = 3$) such that battery overflow becomes a possibility (line 4). The battery is charged only when energy in the supercapacitor exceeds the peak requirements of the

Algorithm 1 Energy Budgeting

Input: (i) harvested energy in charged capacitor, E_{cap}; (ii) battery energy storage level, LV_B; (iii) energy budget to execute one core at critical frequency, E_{crt}; (iv) energy budget to execute one core at maximum frequency, E_{max}; (v) number of cores in embedded processor, NUM_CORE
Output: assigned energy budget for next schedule window, E_{budget}

1. **if** $E_{cap} < E_{crt}$ **then** $LV_C \leftarrow 1$
2. **else if** $E_{cap} > E_{max} \times NUM\_CORE$ **then** $LV_C \leftarrow 3$
3. **else** $LV_C \leftarrow 2$
4. **if** $LV_B > LV_C$ **then**
5. set to discharge battery
6. **if** $LV_B = 2$ **then** $E_{budget} \leftarrow E_{crt} \times NUM\_CORE$
7. **if** $LV_B = 3$ **then** $E_{budget} \leftarrow E_{max} \times NUM\_CORE$
8. **else**
9. set to discharge supercapacitor
10. **if** $LV_C = 1$ **then** $E_{budget} \leftarrow 0$
11. **if** $LV_C = 2$ **then** $E_{budget} \leftarrow E_{cap}$
12. **if** $LV_C = 3$ **then**
13. $E_{budget} \leftarrow E_{max} \times NUM\_CORE$
14. $E_{chrg} \leftarrow E_{cap} - E_{budget}$

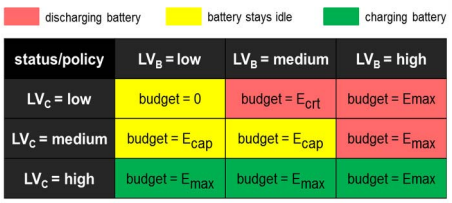


Fig. 7. Hybrid storage management and energy budgeting policy.

processor (lines 12–14). This hybrid storage management and energy budgeting policy is shown in Fig. 7.

The resulting energy budget E_{budget} reflects the amount of energy dynamically collected from the energy harvesting system at runtime and can be considered as a stable energy supply for the next schedule window so that a uniform execution strategy can be enabled for energy efficiency.

C. Critical Frequency, Core Heterogeneity and Thermal Aware Workload Estimation

This section describes our approach for energy budget-based workload estimation at the beginning of each schedule window, which intelligently estimates the optimal workload to be allocated for each core while considering energy efficiency, core heterogeneity, and temperature distributions.

At each reschedule point, our scheme first estimates the amount of workload that can be supported in the upcoming schedule window using the energy budget provided by the hybrid energy storage system. As shown in Fig. 5 earlier, this stage forks into two paths based on the energy budget threshold E_{th}. As discussed in Section III-A3, multicore processors may have cores that have a lower maximum frequency due to within-die process variations. We assume that within-die variations are measured after manufacturing by variation acquisition methods, such as vMeter, proposed in [24], and maximum frequency of each core is considered as known to the runtime manager. The energy budget threshold E_{th} is defined as the energy budget required for the slowest core

Algorithm 2 Active Core Selection and Workload Estimation

Input: (i) energy budget for coming schedule window, E_{budget} (ii) dual-speed method energy efficiency profile (see Section V.E) for task utilizations from 0 to 1, $\delta(U)$
Output: objective utilization for next schedule window, U_{obj}

1. num_active $\leftarrow$ NUM_CORE, $E_{per\_core} = E_{budget}$/num_active
2. **while** $E_{per\_core} < E_{crt}$ **and** num_active > 0 **do**
3. $E_{num\_core} \leftarrow E_{budget}$ / num_active
4. $E_{num\_core-1} \leftarrow E_{budget}$ / (num_active-1)
5. calculate f_{num_core-1} and f_{num_core}, maximum frequencies supported by E_{num_core-1} and E_{num_core}
6. based on Inequality (3), calculate U_{num_core-1} and U_{num_core}, maximum utilization supported by f_{num_core-1} and f_{num_core}
7. look up profile for δ (U_{num_core}) and δ (U_{num_core-1})
8. **if** $\delta(U_{num\_core-1}) > \delta(U_{num\_core})$ **then**
9. num_active $\leftarrow$ num_active – 1
10. update E_{per_core} , U_{per_core}
11. $U_{obj} \leftarrow U_{per\_core} \times$ num_active

to run at its maximum frequency. As we assume even the slowest core is able to run above critical level, it is always true that $E_{\text{th}} > E_{\text{crt}}$. When the average budget per core is below E_{th}, uniform workload distribution is sufficient to ensure that every core runs below its maximum frequency and the runtime manager focuses on active core count selection for energy savings. On the other hand, when the average energy budget for each core is higher than E_{th}, the core heterogeneity cannot be ignored and the runtime manager switches to a heuristic that activates all cores and estimates workload based on each core's achievable frequency. Apart from workload estimation, this stage also takes core temperatures into consideration for proactive runtime thermal management. The final outputs of this stage are the cores to activate and the workload to support in the upcoming schedule window. The following sections describe the three main components of this stage.

1) Critical Frequency-Aware Active Core Selection: We propose a heuristic that selects the number of cores to activate and workload to allocate on each core, assuming uniform workload distribution among activated cores. The motivation for this active core selection heuristic (that is executed only for low energy budget scenarios) is that running a processor below its critical frequency decreases energy efficiency, as can be observed from Table I. With our active core selection heuristic, we can shut down some cores at each reschedule point based on the estimated energy budget. The power dissipated by inactive cores is negligible and the remaining cores can then receive enough workload to run at critical frequency. In addition, the associated power state switching overhead is minimal as we only trigger core shutdown at reschedule points.

The pseudocode of the active core selection heuristic is given in Algorithm 2. The core shutdown procedure is triggered when the energy budget is unable to support all active cores to execute at their critical frequency (line 2). Subsequently (lines 3–10), if one less active core results in a better efficiency, $\delta(U_{\text{num\_core}-1}) > \delta(U_{\text{num\_core}})$ then the scheduler shuts down one core. If the energy budget for the current schedule window is extremely low, eventually all cores in the system will be shut down to save harvested energy for future

Algorithm 3 Heterogeneity-Aware Workload Estimation

Input: peak frequency supported by each cores, f_{peak}(core_id)
Output: objective workload utilization of system for next window, U_{obj}

1. num_active ← NUM_CORE
2. num_unassigned ← NUM_CORE
3. **while** $E_{per\_core} > E_{th}$ **and** num_unassigned > 0 **do**
4. low_id ← core_id of unassigned core with lowest peak frequency
5. $U_{low} \leftarrow f_{peak}(cur\_id)/f_{max}$
6. $U_{obj} \leftarrow U_{obj} + U_{low}$, $E_{budget} \leftarrow E_{budget} - E_{low}$
7. num_unassigned ← num_unassigned - 1
8. update E_{per_core}, E_{th} for unassigned cores
9. calculate f_{per_core}, maximum frequencies supported by E_{per_core}
10. based on Inequality (3), calculate U_{per_core}, maximum utilization supported by f_{per_core}
11. $U_{obj} \leftarrow U_{obj} + U_{per\_core} \times$ num_unassigned

execution. Recursively, these steps set the number of cores to keep active. Finally, the objective task-set utilization (i.e., the amount of workload that the system can support) is obtained by aggregating the supported utilization of each core (line 11). As a result of this selection heuristic, the number of cores activated is tightly related to the energy budget available.

2) Core Heterogeneity-Aware Workload Estimation: When per-core average energy budget for the next schedule window E_{budget}/NUM_CORE is above the energy threshold E_{th}, we have sufficient energy budget to activate all cores and the main concern shifts to assigning workload in a heterogeneity-aware manner (Algorithm 3). The key idea is to recursively assign workload and energy budget to the slowest unassigned core based on its frequency limit until energy budget per core is below a threshold for the remaining unassigned cores. The inputs of this heuristic are the energy budget for the upcoming schedule window E_{budget}, number of cores on the chip *NUM_CORE*, and peak frequency supported by each cores f_{peak} (*core_id*). Initially, all cores will be activated for the next schedule window (line 1) as the energy budget is capable of executing all cores above critical level, i.e., $E_{budget}/NUM\_CORE > E_{th} > E_{crt}$. In the main loop, we first find the slowest core and calculate U_{low}, which is the maximum workload utilization that the core can support (lines 4 and 5). This utilization is accumulated into the objective workload utilization of the system U_{obj}, and the corresponding energy consumption E_{low} is deduced from the energy budget (line 6). Then the heuristic updates (lines 7 and 8) and compares (line 3) per-core average budget and threshold energy again for the rest of cores. After the main loop, the remaining energy budget will be evenly distributed to the unassigned cores and the final utilization is calculated (lines 9–11).

3) Proactive Run-Time Thermal Management: As discussed in Section III-A3, processors typically enforce throttling mechanisms to avoid thermal run away. However, when a throttling decision is enforced, a processor has to drop all executing tasks and halt the system until temperature drops below a certain value. A system that encounters throttling often has frequent and dramatic changes in execution speed, which will hamper system energy efficiency. For this reason, in addition to the baseline enforced throttling mechanisms in processors, we propose to integrate a proactive reaction threshold T_{pro} at a slightly lower temperature than the baseline throttling threshold T_{th} to trigger measures that reduce system workload proactively with the goal of minimizing overheating and balancing workload over time. The details of our proposed scheme are summarized as follows.

1) Cores with higher temperature than others are always given priority when there is a chance of core shutdown in Algorithm 2.
2) Cores with temperature above a proactive reaction threshold T_{pro} only run at critical frequency so as to finish their limited workload with the highest energy efficiency and low power dissipation.
3) When system peak temperature exceeds T_{pro}, the core which is operating at the peak temperature will be shut down to address the thermal hotspot in the system.

Thus, our runtime thermal management approach proactively manages workload to limit processor overheating so that occurrences of enforced throttling can be reduced for more stable execution, compared with traditionally used reactive throttling solutions.

Algorithm 4 Heterogeneity Aware Task Rejection and Assignment

Input: (i) objective utilization from algorithm 2, U_{obj}, (ii) full task set assigned to system for scheduling, ψ; (iii) total utilization of task set ψ, U_{ψ}
Output: optimal execution frequency of each core for upcoming schedule window, f_{opt}*(core_id)*

1. sort task set ψ in non-decreasing order of tasks' penalty densities
2. $\psi_{accepted} \leftarrow \psi$, Uaccepted ← U_{ψ}
3. **for** n = 1:N **do**
4. **if** $U_{accepted} > U_{obj}$ **then**
5. reject n^{th} task
6. $U_{accepted} \leftarrow U_{accepted} - U(n^{th}$ task)
7. **else**
8. done with task rejection, **break**
9. sort accepted task set $\psi_{accepted}$ in non-increasing order of task utilization
10. **for** n = 1:$N_{accepted}$ **do**
11. filter out cores that has $U_n + U_{core} > U_{core\_max}$
12. assign n^{th} task to active core with the lowest utilization
13. get assigned task utilization for each active core, $U_{(core\_id)}$
14. based on Inequality (3), calculate f_{opt}(core_id)
15. execute assigned tasks on each core with dual-speed heuristic

D. Task Penalty and Core Heterogeneity Aware Task Rejection and Allocation

This section describes how periodic tasks are allocated to cores or dropped, based on the awareness of individual task penalties and available core heterogeneity.

To add task priority control in SDA, we distinguish a task's importance by assigning a miss penalty to each task [13]. In this stage, our framework rejects tasks with lower penalty density (Section III-A2) first, rather than simply drop tasks with longer execution time, to allocate the limited energy budget to more important tasks for miss penalty reduction. In particular, for the case when all tasks are assigned an identical miss penalty, this scheme reduces miss penalty equivalent to miss rate. We describe our task rejection heuristic in Algorithm 4.

In lines 1–8, we sort all tasks in nondecreasing order of the tasks' penalty densities so that we can then reject tasks iteratively until the remaining tasks' total utilization is lower than the objective utilization given by Algorithm 2 (described earlier). The remaining tasks form the accepted task set and are assigned to all active cores using a simple but effective approach in lines 9–12. This approach not only enables priority control among tasks but also distributes workload to each core as evenly as possible for balanced execution under a stable frequency. It also ensures that the assigned workload will not exceed a core's maximum capability. After all accepted tasks are assigned, we obtain the actual utilization and optimal frequency of each core for the next schedule window.

E. DVFS Switching-Aware Dual-Speed Method

The previous section showed how we distribute accepted tasks among cores and deduce the theoretical optimum execution frequency for each core, which, however, is unlikely to be supported directly by processors with discrete frequency levels. To address this problem this section introduce a dual-speed method, which approximates the objective optimal frequency by switching between its two adjacent discrete frequencies [25]. For convenience, we denote the adjacent higher frequency as f_{high}, the lower one as f_{low}, and the objective optimal frequency as f_{obj}.

First, to guide the switching between two adjacent discrete frequencies, we need to calculate the proportion of cycles to execute with f_{high}, denoted as α_{high}. Assume that the total number of cycles to be executed is C. Emulating f_{obj} with a combination of f_{low} and f_{high} implies finishing C within the same amount of time, which is shown as

$$\frac{C}{f_{\text{obj}}} = \frac{\alpha_{\text{high}} C}{f_{\text{high}}} + \frac{(1-\alpha_{\text{high}})C}{f_{\text{low}}}. \tag{3}$$

From this equation, we can deduce the proportion α_{high} for each objective frequency f_{obj} as

$$\alpha_{\text{high}}(f_{\text{obj}}) = \frac{1/f_{\text{obj}} - 1/f_{\text{low}}}{1/f_{\text{high}} - 1/f_{\text{low}}}. \tag{4}$$

As f_{low} and f_{high} are determined by f_{obj}, there is a one-to-one correspondence between α_{high} and f_{obj}, and the values of $\alpha_{\text{high}}(f_{\text{obj}})$ can be calculated offline for a given task set. However, it is nontrivial to get close to theoretical efficiency in a dual-speed method implementation due to the following difficulties.

1) Excessive DVFS switching results in massive switching overhead that considerably reduces energy efficiency [22].
2) Executing at f_{low} for too long can cause task timing violations.
3) Executing at f_{high} for too long results in timing slack before the arrival of new job instances of periodic tasks, which is wasted as idle cycles, thus reducing energy efficiency.

To address these issues, we implement a simple and intuitive dual-speed mechanism with intertask switching, which aims to execute as many tasks as possible before switching to another frequency, as summarized in Algorithm 5. Note that in line 3, frequency switching will not be triggered if $f_{\text{obj}} < f_{\text{crt}}$, as executing below critical frequency must be avoided. A more detailed explanation of this algorithm can be found in [11].

Algorithm 5 Dual-Speed Method With Intertask Switching

Input: (i) objective optimal frequency, f_{obj}; switching proportion and threshold profile for f_{obj} from 400 to 1000 MHz, $\alpha_{high}(f_{obj})$ and $C_{thresh}(f_{obj})$

```
1. f_cur ← f_high
2. while true do
3.     if f_obj > f_crt then
4.         if f_cur = f_high then
5.             C_high ← C_high + 1
6.             if jobpool.size ≤ 1 and C_high > C_thresh then
7.                 f_cur ← f_low
8.         if f_cur = f_low then
9.             C_low ← C_low + 1
10.            if C_low > C_high × (1- α )/ α then
11.                f_cur ← f_high
12.                C_low←0, C_high←0
13.    if at reschedule point then
14.        update f_obj based on Inequality (3)
15.        find adjacent frequencies such that f_low < f_opt < f_high
16.        fetch α_high(f_obj) and C_thresh(f_obj) from profile
```

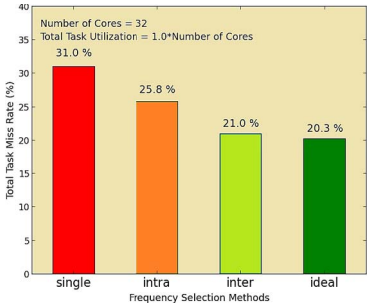


Fig. 8. Comparison of frequency selection methods.

To illustrate the advantage of our dual-speed method with intertask switching, we compare it to three alternatives: 1) *single-speed method* that finds a higher than optimal frequency directly supported by the processor; 2) *intratask method* that aggressively toggles speed during executions for every task while considering switching overhead; and 3) *ideal case* where intratask method is applied, with switching overhead set to 0 to achieve theoretical best case efficiency. As can be observed in Fig. 8, the *single-speed* scheme shows the worst result as it always executes at f_{high} when energy is available. The *intratask* method works better by switching between two DVFS levels. However, its miss rate is still significantly higher than the ideal case due to excessive DVFS switching overhead. By presetting switching threshold and monitoring available workloads in the job pool, our *intertask* switching scheme finds appropriate switching points and results in a miss rate that is close to the ideal case.

VI. Experimental Studies

A. Experiment Setup

We developed a simulator in C++ to implement and evaluate the effectiveness of our proposed SDA framework

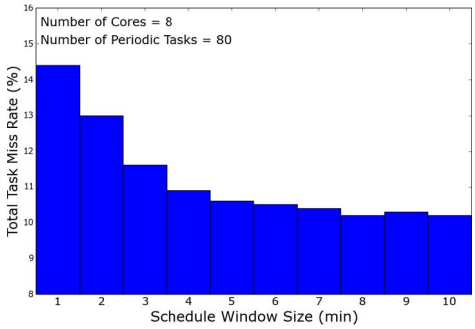


Fig. 9. Miss rates with different schedule window sizes.

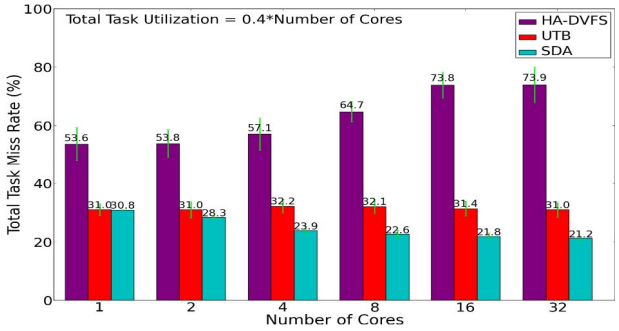


Fig. 10. Miss rate comparison with light workload.

for runtime energy and workload management. The processor's power model was described in Table I. The energy harvesting profile is obtained from historical weather data from Golden, Colorado, USA, provided by the Measurement and Instrumentation Data Center at the National Renewable Energy Laboratory [26]. Our harvesting-based embedded system only executes during daytime over a span of 750 min, from 6:00 A.M. to 6:30 P.M. and shuts down when solar radiation is unavailable.

In most experiments, we generated 50 random tasks for each test set configuration. Each task set has an average task execution time randomly selected from 5–10 s. We vary the periods of all tasks in a task set based on the desired level of utilization required from the entire task set. We also ran experiments with the MiBench benchmark suite of embedded applications [32].

To determine the appropriate schedule window size for the SDA algorithm, we ran several experiments with different window sizes. Fig. 9 shows a set of results (miss rates) for our random task with 100% utilization on a core. We found that when window size increases from 1–5 min, there is a notable decrease in task miss rate. The reason behind this trend is that smaller schedule window sizes cause more task instances to span across the boundary of two different schedule windows, disrupting the newly assigned execution schedules of the next window. When we continue increasing window size beyond 5 min, the performance benefits become negligible while the demand on supercapacitor capacity to buffer energy harvested during a schedule window increases linearly. We found this trend to be consistent for simulations with multiple cores as well. Thus, we set 5 min as the size of schedule window in SDA for our experiments, to balance system performance and supercapacitor capacity requirements.

B. Comparison Between SDA and Prior Work

In this first set of experiments, we compare overall miss rates between *HA-DVFS* [8], *UTB* [9], and our proposed *SDA* framework for different number of homogeneous cores ranging from 1 to 32 with insufficient energy harvesting, for which the energy storage system is not stressed with surplus energy, so that the comparison in this section is focused on scheduling performance of SDA compared to prior works, without considering the advantages from our improved energy storage system design. We modeled the state-of-the-art utilization-based algorithm (UTB) in our environment. We also extended the energy harvesting-aware technique (HA-DVFS) for multicore systems with balanced task partitioning across multiple cores, to enable another comparison point. With increasing number of cores, we scale harvesting power, number of tasks, and total task utilization linearly to keep a consistent and reasonable per core workload and energy budget.

First, we experiment on a workload with per core utilization set to 40%, which has moderate energy requirements such that the system can execute at critical frequency for the highest efficiency when energy is sufficient. The results are shown in Fig. 10. HA-DVFS can be seen to have a much higher miss rate as it does not make use of periodic task information and, thus, underestimates future workload. For the other two techniques, the advantage of SDA over UTB is small for the single-core setup because task utilization is not very high. However, with increasing number of cores, SDAs advantage expands considerably even though per-core workload and energy budget stays the same. One reason for this trend is that UTB uses an isolated task dropping scheme on each core, which is based on energy availability prediction for one upcoming task, ignoring workload on other cores that compete for the same energy source. In contrast, SDA performs task rejection before assigning accepted tasks to different cores; thus, the workload is adapted to a system-wide energy budget that has been predicted. Furthermore, SDA actually benefits from increasing number of cores as it exploits the flexibility to shut down some cores for higher efficiency.

We also compare UTB and SDA under a much heavier workload, with per core utilization set to 100%, the results for which are shown in Fig. 11. Not surprisingly, the heavier workload expands the performance gap between UTB and SDA. The reason for the increasing performance gap is that the higher workload implies more stringent timing and energy constraints, under which SDAs balanced runtime adjustment becomes more effective, as discussed in Section IV-A2. As a result, the most significant difference between these two techniques can be seen for the 32-core platform scenario, where SDA achieves approximately 70% miss rate reduction compared with UTB. In addition, the results of SDA have less variation on multiple task sets compared with UTB, which indicates that task set randomness has less impact on SDA as its dynamic adjustment is based on the scope of the entire task

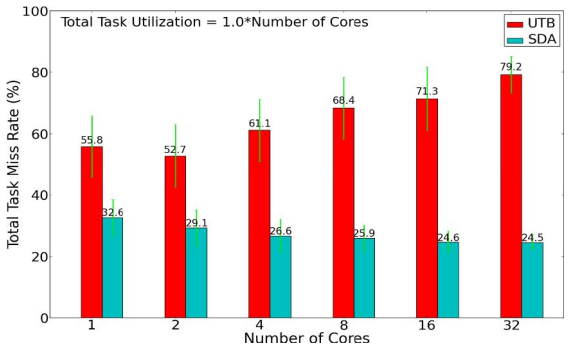


Fig. 11. Miss rate comparison with heavy workload.

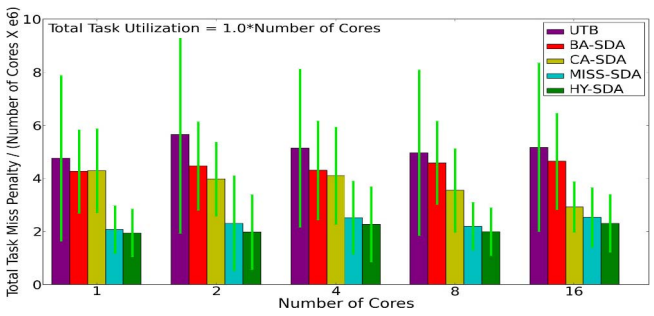


Fig. 12. Overall miss penalty comparison.

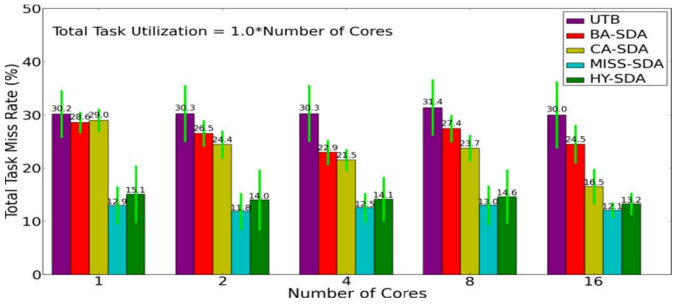


Fig. 13. Overall miss rate comparison.

set, and not just individual tasks.

In addition, we compare performance of HA-DVFS, UTB, and SDA by scheduling a four-core system running a set of applications (*jpeg, qsort, dijkstra, patricia, blowfish, susan, tiff*) extracted from MiBench, a benchmark suite of embedded applications [32], with a total utilization of 160%, where every application executes recursively based on its assigned period with each application execution request considered as an independent task instance. In this experiment, HA-DVFS, UTB, and SDA all show higher miss rates compared with average values of a similar four-core configuration in Fig. 10, with miss rates of 58.5%, 33.8%, and 25.8%, respectively. The reason lies in the application set's higher average length of task instances compared with most of the randomly generated tasks, making it harder to balance workload among cores and leading to a higher overhead when a task instance's life cycle spans across two schedule windows, as discussed earlier in Section VI-A.

C. Analysis of SDA With Hybrid Energy Storage

This set of experiments explores the performance benefits of our proposed SDA algorithm together with the proposed hybrid energy storage system. Compared with the previous section that focuses on scenarios with insufficient energy harvesting, experiments in this section assume per-core nominal harvested energy scaled up by a factor of two, so that the system receives more than sufficient energy occasionally and surplus energy needs to be stored to support execution when harvesting power drops. We use the approach from [14] to model rate capacity effect of batteries by scaling efficiency based on discharge current. In addition, we implemented four variants of SDA: 1) *BA-SDA*: SDA for battery-only system with doubled battery capacity; 2) *CA-SDA*: SDA for supercapacitor-only system with doubled supercapacitor capacity; 3) *MISS-SDA*: SDA with hybrid storage and focus on miss rate reduction; and 4) *HY-SDA*: SDA with hybrid storage and focus on miss penalty reduction. These variants of our approach were compared against UTB. In addition, UTB, BA-SDA, and CA-SDA rely on a *moving average* algorithm for energy-harvesting prediction [8] as they do not have dual-supercapacitor design to buffer harvested energy for upcoming schedule windows. All task sets have per-core utilization of 100% for this set of experiments. In addition, tasks are assigned a miss penalty ranging from 1 to 100 with a uniform distribution. We compared average overall miss penalty and miss rate for these various techniques, with increasing multicore platform complexity (from 1 to 16 cores). Capacities of batteries and supercapacitors, and nominal harvested energy for the entire system scale linearly with number of cores in the processors.

The results for this experiment are shown in Figs. 12 and 13. Similar to the conclusion in the previous section, both BA-SDA and CA-SDA have lower miss penalty (Fig. 12) and miss rate (Fig. 13) than UTB, and their advantage expands with increasing number of cores. However, their advantage over UTB is less significant compared with what we observe in the previous section (Fig. 11). The reason is twofold.

1) With doubling of per-core nominal harvested energy in this set of experiments, the stringent energy constraint that highlights the difference between UTB and SDA is relaxed significantly.
2) With more than sufficient energy harvesting, management of surplus energy becomes the new bottleneck that partially diminishes the advantage of SDA, namely, the performances of BA-SDA and CA-SDA mainly suffer from lower charging/discharging efficiency of the battery and limited capacity of the supercapacitor. In addition, CA-SDA has an advantage over BA-SDA with increasing number of cores in the system as systems with more cores have higher demands on discharging current, and supercapacitors, with their high power density, serve high-current load more efficiently than batteries [14].

For MISS-SDA and HY-SDA, integration with our hybrid storage system managed by the SDA-based policy results in a much lower miss penalty and miss rate compared with

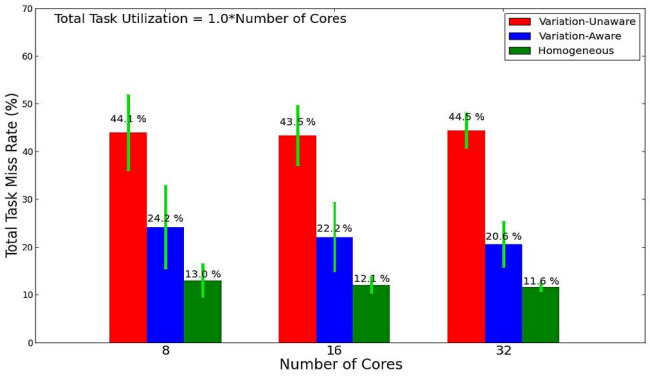


Fig. 14. Overall miss rate comparison with core heterogeneity.

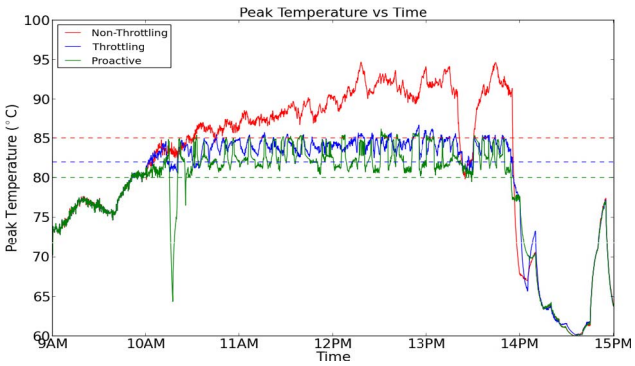


Fig. 15. Peak temperature of various thermal management techniques.

UTB, BA-SDA, and CA-SDA. Even though this significant performance improvement is due to the introduction of a hybrid storage system in MISS-SDA and HY-SDA, the efficient management of such a hybrid storage system is made possible by the semidynamic scheme of SDA, which offers flexibility at reschedule points to select the appropriate energy source and to deduce optimal energy budgets at the start of each schedule window. In addition, the difference between MISS-SDA and HY-SDA is in how they prioritize minimization of miss rate and miss penalty. The HY-SDA scheme leads to the lowest task miss penalty, with up to 65% reduction compared with UTB, while MISS-SDA results in a slightly higher miss penalty than HY-SDA as it focuses on miss rate reduction. As expected, the miss rate for MISS-SDA is the lowest and has less variation compared with HY-SDA (Fig. 13).

D. Analysis of Core Heterogeneity-Aware Management

Next, we study the performance impact of core heterogeneity caused by within-die process variations. Based on the results from [15], we set core frequency variation within a die as 33% and static power variation as 50% with normal distribution. When a frequency level cannot be reached by a core, the system always conservatively sets frequency to the next lower discrete frequency level. We tested three different setups as follows.

1) *Variation-Unaware:* SDA with core heterogeneity-aware techniques disabled. In addition, we assume that the system will force cores to execute at frequencies no higher than their maximum capability to ensure stability.
2) *Variation-Aware:* SDA with our core heterogeneity-aware techniques.
3) *Homogeneous:* An ideal case assuming no heterogeneity.

The results for this paper are shown in Fig. 14. It can be seen that without the awareness of within-die process variation, the system suffers from a very high miss rate, as the assigned workload exceeds the actual execution capabilities of slower cores on the die, resulting in a faulty schedule that wastes energy and CPU time on tasks that cannot be finished on time. In comparison with core heterogeneity-aware workload distribution, the system avoids faulty scheduling and alleviates the impact of process variation. However, as expected, the results are inferior to that obtained for the ideal case that has homogeneous cores unaffected by process variation because of the degradation in maximum throughput supported and nonuniform workload distribution forced by intercore heterogeneity.

E. Analysis of Run-Time Thermal Management

In this section, we explore the impact of runtime thermal management in an energy harvesting environment. While prior work [3] has considered the effect of temperature on MPPT in energy harvesting systems, it has not considered the impact of thermal-induced overheating on task execution throttling and slowdown in energy-harvesting embedded systems. To simulate a scenario with high overheating risk (as discussed in Section IV-D), we evaluate our approach for a very heavy workload with per core utilization set to 100%. Our environmental profile considers high solar intensity and ambient temperatures from 9 A.M. to 3 P.M. For thermal analysis, we integrated our simulator with Hotspot, a thermal modeling and analysis tool [27]. We set package parameters of the Hotspot tool to model a 16-core processor with no power-hungry cooling system (only a heat spreader and heat sink is assumed). We assume die area of our chip to be a 16 mm $\times$ 16 mm, with cores placed in a mesh topology. Then, we set processor package size as 60 mm $\times$ 60 mm, which is also the size of heat spreader and heat sink. In our tests, we compare the performance of three schemes as follows.

1) *Nonthrottling:* A basic SDA scheme with no runtime thermal management scheme. This is representative of current state of the art scheduling techniques for energy harvesting systems that ignore thermal issues.
2) *Throttling:* We again consider our SDA scheme without thermal-awareness, but here system hardware can measure temperature and reactively enforce throttling when temperature exceeds the throttling threshold.
3) *Proactive:* This is our SDA approach that integrates proactive core slowdown and task redistribution from Section V-C3 to proactively address hotspots.

The results for the three schemes are shown in Fig. 15. It can be seen that the *Nonthrottling* scheme suffers from high peak temperatures for extended periods of time. Such high temperatures will significantly impact the system stability and

TABLE II
COMPARISON BETWEEN THROTTLING AND PROACTIVE SCHEMES

Thermal management scheme	average peak temperature	number of throttlings	overall task miss rate
Throttling	79.60°C	94	35.92%
Proactive	78.53°C	74	35.33%

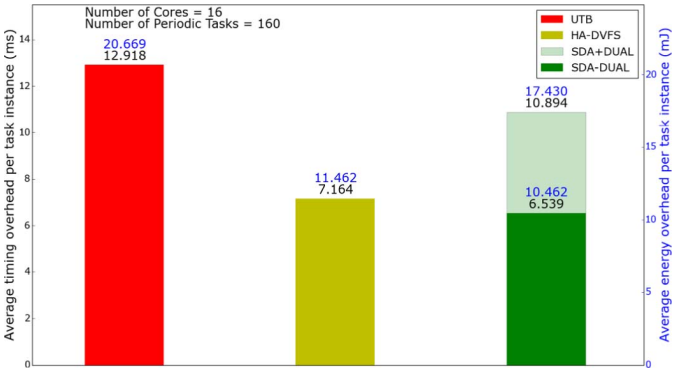


Fig. 16. Comparison of scheduling overhead.

reliability. In contrast, the reactive *Throttling* scheme is able to control temperature to stay below the throttling threshold for a majority of the time. In Fig. 15, the red dashed line indicates the throttling threshold at 85 °C and the green dashed line shows the threshold at 80 °C at which throttling terminates. Note that peak temperature seldom drops to 80 °C in simulation. This is due to the fact that other unthrottled cores take over the role of thermal hotspots in the system from the throttled cores. Our TA-SDA *Proactive* scheme proactively performs core slowdown when temperature exceeds a proactive reaction threshold (set to 82 °C, and shown with the blue dashed line in Fig. 15). This scheme helps to increase energy efficiency by avoiding unbalanced frequencies created by thermal throttling. Table II shows how our proactive approach not only reduces peak temperature but also reduces the number of throttling instances, which allows more efficient scheduling management, culminating in an overall improved task miss rate. The results highlight the benefits of proactive runtime thermal management.

F. Analysis of Scheduling Overhead

To compare scheduling overhead between UTB, HA-DVFS, and our proposed SDA framework, we executed the scheduling procedures of these schemes on the gem5 simulator [33] with a single thread at 1 GHz to observe average execution time overhead averaged over all task instances when managing a 16-core system running 160 periodic tasks with a scheduling granularity of 1 ms. The results of this paper are shown in Fig. 16, in which we can observe that SDA+DUAL has less scheduling overhead (with respect to performance and energy) compared with UTB, while providing more features such as hybrid storage-based energy budgeting, thermal management, and dual-speed switching. The main reason for the lower overhead with SDA+DUAL is that it is designed to reuse intermediate information computed at the beginning of each schedule window, avoiding frequent on-the-fly scheduling procedure invocations during task execution, with dual-speed method as the exception. The HA-DVFS also has much lower overhead than UTBs, as well as SDA+DUAL, as most of its features are triggered only when a new task instance is available. We were also interested in quantifying the overhead of our dual-speed method, which is perhaps the most complex runtime component in our scheduling framework. We therefore also present the scheduling overhead for SDA-DUAL, which disabled the dual-speed feature. It can be seen that without the dual-speed method, our scheduler execution time and energy overheads become lower than overheads for UTB and HA-DVFS.

VII. CONCLUSION

In this paper, we proposed a novel framework for runtime energy and workload management based on an SDA, for real-time multicore embedded systems with solar energy harvesting. Compared with the best known previous work, our approach is promising for energy-harvesting-based multicore embedded systems.

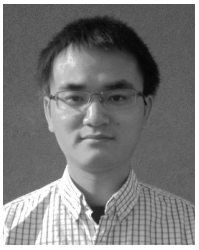
1) Up to 70% miss rate reduction and 65% miss penalty reduction for SDA compared with the best known prior work, UTB.
2) An analysis with system overheating considerations establishes the need for combining proactive thermal management during scheduling, as done in our SDA approach, to reduce both miss rate and average peak temperature among cores.
3) The SDA with core-heterogeneity awareness presents miss rate reduction of 49% compared with SDA without such awareness when process variation effects on maximum frequency and power are considered.

Overall, SDA provides a holistic solution with many novel components, integrating a new hybrid energy-storage system, task drop penalty awareness, runtime thermal management, and core heterogeneity awareness. Our future work will focus on scheduling of applications on embedded systems powered by energy harvesting, with heterogeneous multiprocessors composed of complex, high-performance, and simple, high-efficiency cores (e.g., ARM big.LITTLE [31]).

REFERENCES

[1] Nvidia Corp. (2010). *The Benefits of Multiple CPU Cores in Mobile Devices*. [Online]. Available: http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices.pdf
[2] E. Humenay, D. Tarjan, and K. Skadron, "Impact of process variations on multicore performance symmetry," presented at the Conf. Design, Autom. Test Eur., San Jose, CA, USA, Apr. 2007, pp. 1–6.
[3] C. Li, W. Zhang, C.-B. Cho, and T. Li, "SolarCore: Solar energy driven multi-core architecture power management," presented at the Int. Symp. High Perform. Comput. Archit., San Antonio, TX, USA, 2011, pp. 205–216.
[4] X. Lin, Y. Wang, D. Zhu, N. Chang, and M. Pedram, "Online fault detection and tolerance for photovoltaic energy harvesting systems," presented at the IEEE/ACM Int. Conf. Comput.-Aided Design, San Jose, CA, USA, Nov. 2012, pp. 1–6.
[5] Y. Zhang, Y. Ge, and Q. Qiu, "Improving charging efficiency with workload scheduling in energy harvesting embedded systems," presented at the 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC), Austin, TX, USA, May/Jun. 2013, pp. 1–8.

[6] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Lazy scheduling for energy harvesting sensor nodes," presented at the Conf. Distrib. Parallel Embedded Syst., Braga, Portugal, 2006, pp. 125–134.
[7] S. Liu, Q. Qiu, and Q. Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting," presented at the Conf. Design, Autom. Test Eur., Munich, Germany, Mar. 2008, pp. 236–241.
[8] S. Liu, J. Lu, Q. Wu, and Q. Qiu, "Harvesting-aware power management for real-time systems with renewable energy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1473–1486, Aug. 2012.
[9] J. Lu and Q. Qiu, "Scheduling and mapping of periodic tasks on multicore embedded systems with energy harvesting," presented at the Int. Green Comput. Conf., Los Alamitos, CA, USA, Jul. 2011, pp. 1–6.
[10] Y. Xiang and S. Pasricha, "Harvesting-aware energy management for multicore platforms with hybrid energy storage," presented at the Int. Conf. Great Lakes Symp. VLSI, Paris, France, 2013, pp. 25–30.
[11] Y. Xiang and S. Pasricha, "Thermal-aware semi-dynamic power management for multicore systems with energy harvesting," presented at 14th Int. Symp. Quality Electron. Design, Santa Clara, CA, USA, Mar. 2013, pp. 619–626.
[12] H. Aydin, P. Mejia-Alvarez, D. Mossé, and R. Melhem, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," presented at the 22nd IEEE Real-Time Syst. Symp., Washington, DC, USA, Dec. 2001, pp. 95–105.
[13] J.-J. Chen, T.-W. Kuo, C.-L. Yang, and K.-J. King, "Energy-efficient real-time task scheduling with task rejection," presented at the Conf. Design, Autom. Test Eur., San Jose, CA, USA, Apr. 2007, pp. 1–6.
[14] A. Mirhoseini and F. Koushanfar, "HypoEnergy. Hybrid supercapacitor-battery power-supply optimization for energy efficiency," presented at the Conf. Design, Autom. Test Eur., Los Alamitos, CA, USA, Mar. 2011, pp. 1–4.
[15] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," presented at the 35th Annu. Int. Symp. Comput. Archit., New York, NY, USA, 2008, pp. 363–374.
[16] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-aware task allocation and scheduling for embedded systems," presented at the Conf. Design, Autom. Test Eur., Washington, DC, USA, Mar. 2005, pp. 898–899.
[17] M. Veerachary, T. Senjyu, and K. Uezato, "Voltage-based maximum power point tracking control of PV system," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 1, pp. 262–270, Jan. 2002.
[18] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 9, pp. 1127–1140, Sep. 2008.
[19] B. Carter, J. Matsumoto, A. Prater, and D. Smith, "Lithium ion battery performance and charge control," presented at the Int. Energy Convers. Eng. Conf., Piscataway, NJ, USA, Aug. 1996, pp. 363–368.
[20] F. Ongaro, S. Saggini, and P. Mattavelli, "Li-ion battery-supercapacitor hybrid storage system for a long lifetime, photovoltaic-based wireless sensor network," *IEEE Trans. Power Electron.*, vol. 27, no. 9, pp. 3944–3952, Sep. 2012.
[21] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," presented at the Design Autom. Conf., Anaheim, CA, USA, Jun. 2008, pp. 734–739.
[22] J. Park, D. Shin, N. Chang, and M. Pedram, "Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors," presented at the Int. Symp. Low-Power Electron. Design, Austin, TX, USA, Aug. 2010, pp. 419–424.
[23] Z. Xu *et al.*, "Electrochemical supercapacitor electrodes from sponge-like graphene nanoarchitectures with ultrahigh power density," *J. Phys. Chem. Lett.*, vol. 3, no. 20, pp. 2928–2933, Sep. 2012.
[24] B. Hargreaves, H. Hult, and S. Reda, "Within-die process variations: How accurately can they be statistically modeled?" presented at the Asia South Pacific Design Autom. Conf., Seoul, Korea, Mar. 2008, pp. 524–530.
[25] D. Rajan, R. Zuck, and C. Poellabauer, "Workload-aware dual-speed dynamic voltage scaling," presented at the Int. Conf. Embedded Real-Time Comput. Syst. Appl., Sydney, Qld., Australia, 2006, pp. 251–256.
[26] National Renewable Energy Lab. (2002). *Measurement and Instrumentation Data Center*. [Online]. Available: http://www.nrel.gov/midc
[27] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.
[28] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," presented at the Design Autom. Conf., San Diego, CA, USA, 2004, pp. 275–280.
[29] E. L. Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," presented at the Int. Conf. Power Aware Comput. Syst., Berkeley, CA, USA, 2010, pp. 1–8.
[30] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," presented at the Int. Symp. High Perform. Comput. Archit., Salt Lake City, UT, USA, Feb. 2008, pp. 123–134.
[31] ARM Holdings. (2011). *big.LITTLE Processing With ARM Cortex-A15 & Cortex-A7*. [Online]. Available: http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf
[32] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," presented at the Int. Workshop Workload Characterization, Washington, DC, USA, 2001, pp. 3–14.
[33] (2014). *The GEM5 Simulator System*. [Online]. Available: http://www.m5sim.org



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. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA.

His current research interests include low-power computing, parallel embedded systems, heterogeneous computing, and CAD algorithms.

Mr. Xiang was a recipient of the A. Richard Newton Young Student Fellow Award in 2013.

Sudeep Pasricha (M'02–SM'13) received the B.E. degree in electronics and communication engineering from the Delhi Institute of Technology, New Delhi, India, in 2000, and the M.S. and Ph.D. degrees in computer science from the University of California at Irvine, Irvine, CA, USA, in 2005 and 2008, respectively.

He is currently an Associate Professor of Electrical and Computer Engineering with Colorado State University, Fort Collins, CO, USA. His current research interests include energy-efficient and fault-tolerant design for high-performance computing and embedded systems.

Dr. Pasricha was a recipient of the George T. Abell Outstanding Mid-Career Faculty Award in 2014, the AFOSR Young Investigator Award in 2013, the ACM SIGDA Technical Leadership Award in 2012, and Best Paper Awards at the IEEE International Conference on Computer Systems and Applications in 2011, the IEEE International Symposium on Quality Electronic Design in 2010, and the ACM/IEEE Asia and South Pacific Design Automation Conference in 2006. He has been an Organizing Committee Member and a Technical Program Committee Member of various IEEE/ACM conferences, such as the Design Automation Conference (DAC), the Design Automation and Test in Europe Conference (DATE), the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), the International Symposium on Networks on Chip (NOCS), the ISQED, and the Great Lakes Symposium on VLSI (GLSVLSI).