System-Wide Leakage-Aware Energy Minimization using Dynamic Voltage Scaling and Cache Reconfiguration in Multitasking Systems

Weixun Wang, *Student Member, IEEE*, Prabhat Mishra, *Senior Member, IEEE*

***Abstract*—System optimization techniques are widely used to improve energy efficiency as well as overall performance. Dynamic voltage scaling (DVS) is well studied and known to be successful in reducing processor energy consumption. Due to the increasing significance of the memory subsystem's energy consumption, dynamic cache reconfiguration (DCR) techniques are recently proposed at the aim of improving cache subsystem's energy efficiency. As the manufacturing technology scales into the order of nanometers, leakage current, which leads to static power consumption, becomes a significant contributor in the overall power dissipation. In this article, we consider various system components and study their impact on system-wide energy consumption under different processor voltage levels as well as cache configurations. Based on the observation, we efficiently integrate DVS and DCR techniques together to make decisions judiciously so that the total energy consumption is minimized. Our studies show that considering only DVS or DCR and ignoring the impact from other system components may lead to incorrect conclusions in overall energy savings. Experimental results demonstrate that our approach outperforms existing leakage-aware DVS techniques by 47.6% and leakage-oblivious DVS + DCR technique by up to 23.5%.**

***Index Terms*—Embedded System, Power Management, Cache, Energy, Leakage Power.**

I. Introduction

Energy conservation is a primary optimization objective in embedded systems design since these systems are generally limited by battery lifetime. Figure 1 shows a system-wide power composition for a typical System-on-Chip (SoC) [1]. It can be seen that processor, cache subsystem, memory and bus are the four main components which make comparable contributions to overall power consumption. Therefore, system-wide energy optimization techniques should consider all of them in order to reflect practical benefits and achieve overall energy reduction. Various low-power techniques exist which tune different components in the system at runtime. Dynamic voltage scaling (DVS) [2] of the processor takes the advantage of the fact that linear reduction in the supply voltage can quadratically reduce the power consumption while linearly slows down the operating frequency. At the same time, memory hierarchy, especially the cache subsystem, has become comparable to the processor with respect to the contribution in overall energy consumption [3]. Dynamic cache reconfiguration (DCR) offers the ability to tune the cache configuration parameters at runtime to meet application's unique requirement so that significant amount of memory subsystem energy consumption can be saved [4] [5]. The working set of the application decides the favored cache capacity. Its spatial and temporal locality reflect the cache line size and associativity, respectively.

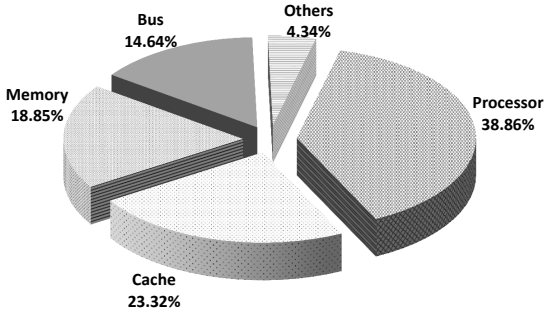


Fig. 1. Power composition of a typical SoC.

Real-time multitasking systems bring particular design and optimization considerations in order to satisfy the imposed timing constraints. In real-time systems, all tasks have to finish execution before their deadlines to ensure correct system behavior. Earliest Deadline First (EDF) [6] is the most commonly used real-time scheduling algorithm. Under EDF, a preemptive task set is said to be *schedulable* as long as the system utilization rate is no more than 1 [7]. Essentially, DVS in real-time systems slows down the processor, thus saves power/energy, at the cost of extending task's execution time. Employing DCR in such systems also needs to be done in a controlled way. This is because different cache configurations will lead to different miss ratios and penalties therefore the total number of clock cycles is affected. While existing research works exploit them separately, our proposed research efficiently employs both DVS and DCR simultaneously to reduce overall energy consumption in hard real-time systems.

In the last decade, we have observed a continuous CMOS device scaling process in which higher transistor density and smaller device dimension have led to increasing leakage (static) power consumption. This is mainly due to the proportionally reduced threshold voltage level with the supply voltage which decreases at a speed of 0.85X per generation [8]. Lower threshold voltage results in larger leakage current which mainly consists of subthreshold current and reverse bias junction current. Study has shown that leakage power is responsible for over 42% of overall power dissipation in

This work was partially supported by NSF grant CCF-0903430 and SRC grant 2009-HJ-1979.

The authors are with the Department of Computer and Information Science and Engineering at the University of Florida, Gainesville FL 32611-6120, USA (email:{wewang, prabhat}@cise.ufl.edu).

the 90nm generation [9] and can exceed above half in recent 65nm technology [10]. Static energy is projected to account for near 70% of the cache subsystem's budget in 70nm technology [10]. Leakage power also constitute a major fraction of the energy consumption of system buses [11]. Memory modules can also consume significant amount of leakage power [12]. Therefore, decisions must be made judiciously on whether to slow down the system to save dynamic power or to finish task execution faster and switch the system to sleep mode to reduce static power. While existing techniques try to control the leakage power along with DVS [13], extra consideration needs to be taken when DCR is also employed and other system components are taken into account.

Our proposed research in this paper integrates DVS and DCR together in hard real-time systems to minimize system-wide energy consumption. Our main contribution is that, unlike existing DVS approaches which either ignore other system components but the processor or assume application-independent constant power values, we systematically incorporate power consumptions from the processor, cache hierarchy, buses and main memory based on the same set of application simulation statistics. The power estimation framework that we propose uses separate power analyzers for different system components and efficiently integrates them together. We take a step forward by examining the correlation among the energy models of all the components and find that they have significant impact on the decision making of both DVS and DCR. Based on the energy estimation, for each task, we can generate the profile table which stores overall beneficial set of voltage levels and cache configurations. While DVS and DCR decisions are made at design time based on static slack allocation, task procrastination is carried out at runtime to achieve more idle energy savings. For real-time embedded systems, off-line analysis is of great importance since they normally have highly deterministic characteristics, e.g., task release time, deadline and execution time, which should be fully utilized for energy optimization. Furthermore, sophisticated analysis such as memory behavior profiling can only be carried out during design time. Extensive experiments show that our approach can result on average 47.6% energy savings compared to DVS-only systems and up to 23.5% extra savings compared to leakage-oblivious DVS + DCR technique [14].

The rest of the article is organized as follows. Related works are discussed in Section II. Section III presents the system models of our work. Our leakage-aware voltage scaling and cache reconfiguration technique is described in Section IV. Section V demonstrates our experimental results. Finally, Section VI concludes the paper.

II. Related Work

A great deal of research work exists on applying DVS in real-time systems. A lot of them focus on minimizing dynamic power consumption and ignoring the static portion by slowing down the processor as much as possible [15] [16]. Meanwhile, a number of existing works pay attention to control processor leakage power in real-time systems [13] [17] [18] [19] [20]. Jejurikar et al. [13] present a leakage-aware DVS scheme which does not allow to slow down the processor speed below a certain level called *critical speed* to avoid growing static energy to compensate the reduction in dynamic energy. They also propose a task procrastination scheduling technique to maximize processor idle intervals. Chen et al. [19] address the same problem in a rate-monotone scheduling system. However, none of the above techniques considered DCR. Furthermore, they did not take other system components into account which potentially limits the benefit of their approaches. Jejurikar et al. [17] and Zhong et al. [18] proposed leakage-aware DVS techniques for system-wide energy minimization. However, the system components considered in their work are only mock units with power consumptions assumed to be application-independent constants. In other words, their approaches use over-simplified models and cannot incorporate other power-hungry components whose power dissipation are determined at runtime. Moreover, cache reconfiguration is not considered by any of them.

Cache reconfiguration has drawn considerable research efforts in both general-purpose [21] and real-time [4] [5] systems. Wang et al. employed DCR in soft real-time systems by dynamically utilizing static profiling information to tune the cache configuration in order to achieve significant energy savings for both single-level cache [4] and multi-level cache hierarchy [5]. Micro-architecture level techniques are proposed at the aim of saving leakage energy in cache subsystem by switching unused cache sub-arrays into low-power mode [22]. Chi et al. [23] applied these techniques in real-time systems. Data compression is also proposed for cache energy reduction in [24]. However, none of these approaches considers processor voltage scaling or other system components. Nacul et al. [14] presented preliminary results to demonstrate the benefit of combining DVS and DCR together in real-time systems but they did not consider leakage power which may make their solution inferior.

III. System Model

In this section, we describe our task model and energy models. We assume that DVS and DCR are available in the target system. We adopt the reconfigurable cache architecture proposed in [25] which is suitable for embedded systems. Specifically, we have:

- A voltage scalable processor which supports h different voltage levels V$\{v_1,v_2, \ldots ,v_h\}$.
- A highly configurable cache hierarchy with reconfigurable parameters including cache size, line size and associativity, in which separate L1 caches and unified L2 cache can be reconfigured individually. Let C$\{c_1,c_2, \ldots ,c_l\}$ denote l configurations of the cache hierarchy.

A. Task Model

If each task is uniformly assigned one voltage level and one cache configuration throughout all its instances, we have:

- A set of m independent periodic tasks T$\{\tau_1,\tau_2,\ldots,\tau_m\}$.
- Each task $\tau_i \in$ T has known period p_i and deadline d_i.
- Task $\tau_i \in$ T has energy consumption and execution time $E_i(v_j,c_k)$ and $T_i(v_j,c_k)$ with voltage level $v_j \in$ V and cache configuration $c_k \in$ C, respectively.

We assume that task deadlines are equal to their periods and the task set is schedulable by EDF scheduler under the highest voltage level and largest cache configuration. Let P denote the task set's hyper-period (equal to the least common multiple of all tasks' periods). Here, j_i and k_i represent the indices of selected voltage level and cache configuration for task τ_i, respectively. Our objective can be stated as:

$$min(E = \sum_{i=1}^{m} \frac{P}{p_i} \cdot E_i(v_{j_i}, c_{k_i})) \quad (1)$$

subject to: $\sum_{i=1}^{m} \frac{T_i(v_{j_i}, c_{k_i})}{p_i} \leqslant 1$.

B. Energy Model

Our energy models for processor and cache are described in [26]. Note that, since lower-level memory and buses are modeled simultaneously in this article, the energy required for off-chip access ($E_{offchip\_access}$) and process stalling ($E_{\mu P\_stall}$) are now counted during the power estimation of the corresponding components (e.g. for a L2 cache miss, $E_{offchip\_access}$ is computed in off-chip buses and DRAM memory).

Bus Energy Model: The average dynamic power consumption of various system buses can be calculated by [27]:

$$P_{bus}^{dyn} = \frac{1}{2} \cdot C_{bus} \cdot V_{dd}^2 \cdot n_{trans} \cdot f \quad (2)$$

where C_{bus} is the load capacitance of the bus, V_{dd} is the supply voltage, f is the bus frequency and n_{trans} denotes the average number of transitions per time unit on the bus. Specifically, we have $n_{trans} = (\sum_{t=0}^{T-1} H(B^{(t)}, B^{(t+1)}))/T$, where T is the total number of discretized time units and $H(B^{(t)}, B^{(t+1)})$ gives the Hamming distance between the binary values on the bus at two neighboring time units in T. Therefore, if CC and t_{cycle} denote the clock cycle number and cycle length respectively, the total energy consumption of a bus is determined by its dynamic power P_{bus}^{dyn} and static power P_{bus}^{sta}:

$$E_{bus} = (P_{bus}^{dyn} + P_{bus}^{sta}) \cdot CC \cdot t_{cycle} \quad (3)$$

Memory Energy Model: Memory consists of DRAM has three sources of power consumption: dynamic energy due to accesses E_{mem}^{dyn}, static power P_{mem}^{sta} and refreshing power P_{mem}^{ref}. Specifically, we have:

$$E_{mem}^{dyn} = num\_accesses \cdot E_{access} \quad (4)$$

where *num_access* is the number of memory accesses and E_{access} denotes the dynamic energy required per access. Therefore, we have: $E_{mem} = E_{mem}^{dyn} + (P_{mem}^{sta} + P_{mem}^{ref}) \cdot CC \cdot t_{cycle}$.

IV. Leakage-aware DVS and DCR

A. Overview

Our approach addresses major challenges including design space exploration, system-wide energy analysis, configuration selection and task procrastination to significantly reduce overall energy consumption while meeting all task deadlines. Figure 2 illustrates the workflow of our approach. Each task is profiled through simulation which is driven by design space exploration heuristics. For each simulation, its total system energy consumption is calculated by our energy estimation framework and put into the task's profile table along with the corresponding execution time. Based on the task set characteristics and the profile tables as well as the scheduling policy, processor voltage level and cache configuration can be selected for each task. Task DVS/DCR assignments and procrastination algorithm are then used in a one-pass task scheduling which produces the total energy consumption of the task set during its hyper-period P. This section describes each of these steps in detail.

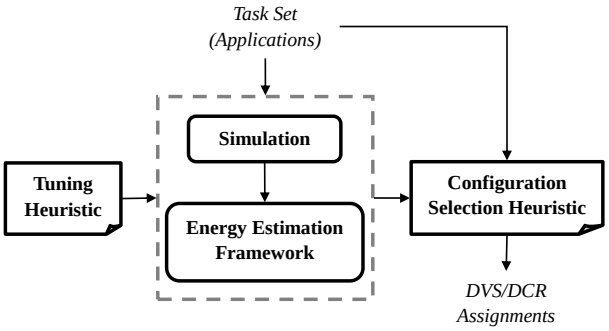


Fig. 2. Workflow of our approach.

B. Two-Level Cache Tuning Heuristic

It is a major challenge to employ multi-level cache reconfiguration since the exploration space is prohibitively large. Our previous work [5] has proposed efficient tuning heuristics for two-level cache hierarchy which can also be applied here. In this article, we use L1 cache sizes of 4KB, 8KB and 16KB, line size of 16, 32 and 64 bytes with set associativity of 1-way, 2-way and 4-way. For L2 cache, the capacity is selected to be 32KB, 64KB and 128KB with line sizes of 64, 128 and 256 bytes and set associativity of 4-way, 8-way and 16-way. Therefore, there are 18 configurations for each individual cache and totally 5832 different configurations for the cache hierarchy [5]. We employ IL1T – Independent Level One Cache Tuning – in this paper to reduce the simulation time while still preserve the most amount of accuracy. Other heuristics described in [5] are also applicable.

C. Power Estimation Framework

Since we do not focus on system design which requires to minimize development time and costs, our energy estimation framework, as shown in Figure 3, targets at a specific SoC micro-architecture and is able to trade more design time for higher accuracy than the one proposed in [28]. We use SimpleScalar [29] as the underlying micro-architectural simulator in our approach. For each application (task) and cache configuration, we run a simulation and collect the execution statistics, memory access statistics and bus activity traces. These information, along with the processor voltage levels, are provided to energy models for each system components, based on which the total system energy can be computed. Note that in our framework, the inputs to each energy model are all from one micro-architectural simulation thus are more comprehensive and systematic, as opposed to [28] in which the inputs are collected separately using instruction-set simulator,

memory trace profiler, cache simulator and bus simulator. Furthermore, by doing this, the impact on DVS/DCR decisions from other system components as well as their correlations, which is not considered in [28], can be reflected in an accurate manner. This framework still provides flexibility to allow different energy models and analyzers to be used.

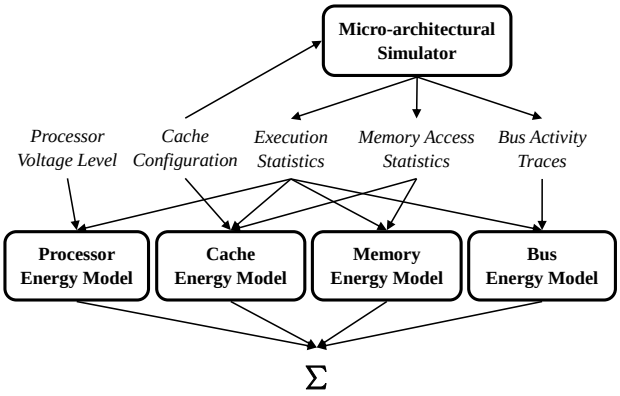


Fig. 3. Overview of our power estimation framework.

D. Critical Speed

The *critical speed* for processor voltage scaling defines a point beyond which the processor speed cannot be slowed down otherwise DVS will no longer be beneficial [13]. The dynamic power consumption of processor, which is exclusively considered in traditional DVS, is usually a convex and increasing function of the operating frequency. However, since lowering processor speed makes the task execution time longer which leads to higher static energy consumption, the total energy consumed per cycle in the processor will start increasing due to further slowdown.

By taking DCR into consideration, we find that cache configuration has significant impact on the critical speed with respect to the overall system energy consumption. As described in Section III-B, there exists strong correlation among the energy models of processor, cache hierarchy and other system components. Since different cache configurations lead to different miss ratios and miss penalty cycles, the number of clock cycles (*CC*) required to execute an application is decided by the cache configuration, which directly affects the energy consumption of other components. On the other hand, the length of each clock cycle (t_{cycle}), which is determined by the processor voltage/frequency level, also directly affects the energy consumption of other components. In other words, DVS and DCR will affect the overall system energy consumption. On the other hand, due to leakage power, all system components will have impact on decision making of DVS/DCR, especially the critical speed. Specifically, when the processor is slowed down by DVS, increasing static energy consumed by cache hierarchy, bus lines and memory will compromise the benefit gained from reduced processor dynamic energy. Therefore, considering DCR and other system components effects, we found that the critical speed is going to increase drastically.

1) Processor + L1 Cache: We use a motivating example in which a single benchmark[1] (*cjpeg* from MediaBench [30]) is executed under all processor voltage levels. It can be observed that in Figure 4, when only processor energy is considered, the critical speed is achieved at $V_{dd} = 0.7V$, which matches the results in [13]. However, as shown in Figure 5, with respect to the total amount of energy consumption, combining processor and L1 caches (both configured to 8KB of capacitance, 32B line size and 2-way associativity) increases the critical speed slightly to around $V_{dd} = 0.75V$, due to the effect from L1 cache's leakage power dissipation. This highlights the importance of considering other system components for accurate analysis when applying DVS. In other words, if L1 caches are incorporated, $V_{dd} = 0.7V$ is no longer a beneficial choice with respect to the overall energy savings. Note that in Figure 5, dynamic energy consumption of L1 caches only includes access energy E_{access} and block refilling energy E_{block_fill}. Energy consumed on buses and lower-level memory hierarchy during L1 cache misses will be incorporated when we add the corresponding components into consideration, as shown in following sections.

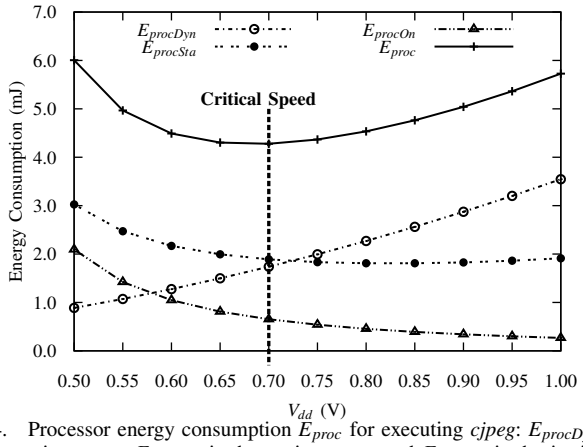


Fig. 4. Processor energy consumption E_{proc} for executing *cjpeg*: $E_{procDyn}$ is the dynamic energy, $E_{procSta}$ is the static energy and E_{procOn} is the intrinsic energy needed to keep processor on.

2) Processor + L1/L2 Cache: Figure 6 shows the impact on the critical speed if L2 cache (with capacity of 64KB, line size 128B and 8-way associativity) is considered in the overall energy consumption. The critical speed increases to the frequency corresponding to $V_{dd} = 0.85V$. For L1 caches, as shown in Figure 5, dynamic energy dominates and leakage energy becomes comparable only when the processor voltage level drops below $0.6V$. However, in L2 cache, for *cjpeg*, leakage energy dissipation dominates while dynamic energy is almost negligible. It is expected since L1 access rate is much higher than L2 while the capacity, thus leakage power, of L2 cache is much larger. Note that, although some other benchmarks (e.g. *qsort* from MiBench [31]) shows non-negligible dynamic energy consumption in L2 cache, the leakage part still dominates when the voltage level goes below a certain point. Therefore, when processor voltage decreases, the total

[1]Although results for *cjpeg* is shown in this section, similar observations have been made for other benchmarks.

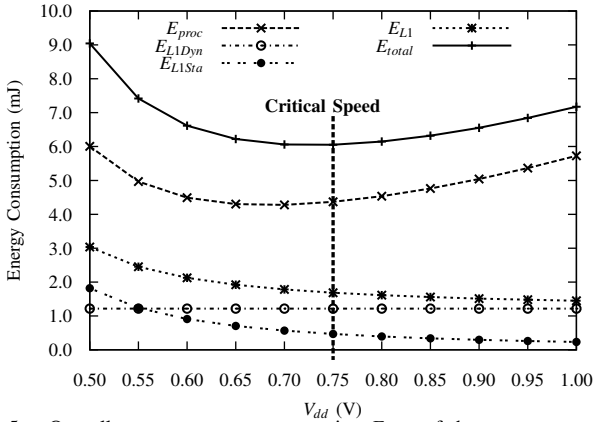


Fig. 5. Overall system energy consumption E_{total} of the processor and L1 caches for executing *cjpeg*: E_{L1Dyn} and E_{L1Sta} are the dynamic and static L1 cache energy consumption, respectively.

leakage energy consumption increases drastically due to the L2 cache. Generally, when DCR is applied, different cache configurations will lead to different critical speed variations.

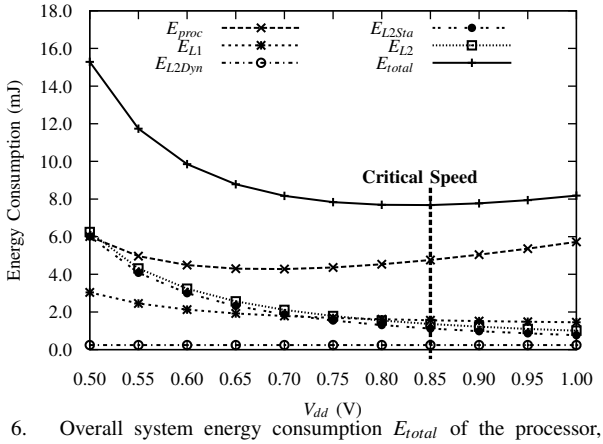


Fig. 6. Overall system energy consumption E_{total} of the processor, L1 caches and L2 cache (configured to 64KB,128B,8-way) for executing *cjpeg*: E_{L2Dyn} and E_{L2Sta} are the dynamic and static L2 cache energy consumption, respectively.

3) Processor + L1/L2 Cache + Memory: Figure 7 illustrates the fact that memory energy consumption also makes the critical speed increase. The memory we considered is modeled as a common DRAM with size of 8MB. It can be observed that memory has a similar effect on the critical speed as L2 cache. In fact, for the configurations we used, the static energy consumptions are comparable for L2 cache and the memory. Although DRAM needs to have its capacitor charge refreshed all the time (which consumes relatively negligible power in 70nm technology [12]), it requires only one transistor to store one bit. Therefore, it consumes much less leakage power per bit compared to cache, which is smaller but made of more power expensive SRAM.

4) Processor + L1/L2 Cache + Memory + Bus: System bus lines have double effect on the critical speed in overall system energy consumption. On one hand, since on-chip buses should have equal frequency as the processor (which makes them dominate in terms of energy among all system buses),

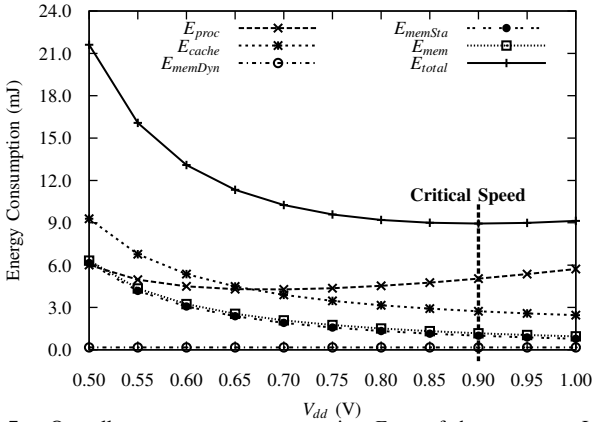


Fig. 7. Overall system energy consumption E_{total} of the processor, L1/L2 caches and memory for executing *cjpeg*: E_{memDyn} and E_{memSta} are the dynamic and static memory energy consumption, respectively; E_{cache} represents the total energy consumption of both L1 and L2 caches.

DVS will lead to dynamic energy reduction in them. On the other hand, like other system components, static power dissipation on system buses is also going to increase along with voltage scaling down, which compensates the dynamic energy reduction. As a result, system buses make very minor impact on critical speed as shown in Figure 8.

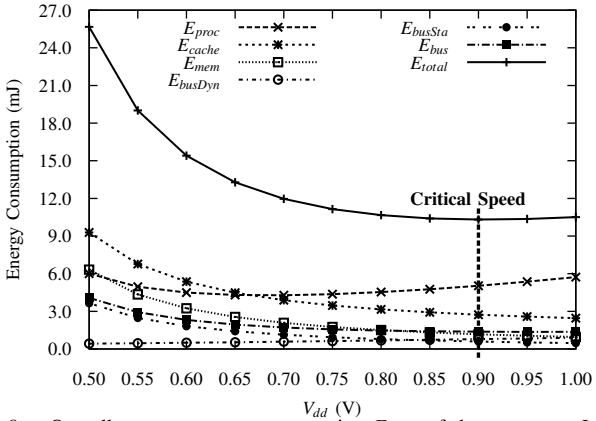


Fig. 8. Overall system energy consumption E_{total} of the processor, L1/L2 caches, memory and system buses for executing *cjpeg*: E_{busDyn} and E_{busSta} are the dynamic and static bus energy consumption, respectively.

For ease of demonstration, we show how energy consumption (both dynamic and static) of each system components vary with voltage scaling in Figure 9. When DVS is not applied ($V_{dd} = 1V$), the processor accounts for over half of overall energy consumption while others also take considerable share. This observation matches what we have shown in Figure 1. When the voltage level decreases, we can see that the energy consumed by the cache hierarchy and memory subsystem increases drastically and, after certain point, it becomes comparable with the processor or even larger. For system buses, due to the reason explained above, this effect is less significant compared to cache and memory.

In conclusion, the discussion above leads to several observations and questions. The critical speed is going to change as different system components are considered – increases

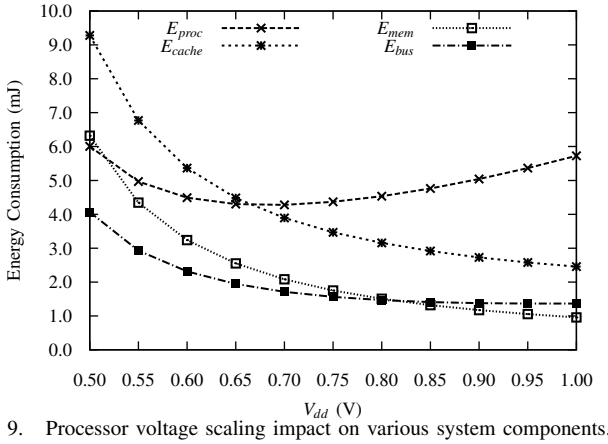


Fig. 9. Processor voltage scaling impact on various system components.

when leakage energy dominant components are added and decreases when dynamic energy dominant components (e.g., DVS-controllable) are added. One would wonder whether DVS is really practically beneficial since our case study shows that the critical speed is at $V_{dd} = 0.9V$ and potentially adding more components may increase it further? A simple answer is yes but it has to be evaluated using leakage-aware DVS and DCR. It is also important to notice that system properties, application characteristics and reconfiguration decisions together will affect the critical speed, which typically varies between $V_{dd} = 0.65V$ and $0.9V$ in our case.

E. Real-Time Voltage Scaling and Cache Reconfiguration

1) Profile Table: We define a *configuration point* as a pair of processor voltage level and cache hierarchy configuration: (v_j,c_k) where $v_j \in$ V and $c_k \in$ C. For each task, we can construct a *profile table* which consists of all possible configuration points as well as the corresponding total energy consumption and execution time. Clearly, all points with the voltage level lower than the critical speed are eliminated. Furthermore, non-beneficial configuration points, which is inferior in both energy and time compared to some other points, are also discarded. In other words, we only consider those Pareto-optimal tradeoff points.

An important observation is that cache configurations behave quite consistently across different processor voltage levels. For example, the L1 cache configuration favored by *cjpeg*, 8KB cache size with 32B line size and 2-way associativity, outperforms all the other configurations in terms of energy. Similar observations can be made when we fix L1 cache configuration while vary L2 cache. Therefore, the profile table for each task actually consists of favored cache hierarchy configurations with voltage levels no higher than the critical speed. In fact, in many cases, we find that only the most energy-efficient cache configuration has beneficial voltage level higher than the critical speed but with much longer execution time than other entries. It can be explained, for example, as shown in Figure 8, that E_{total} only decreased by 1.78% when V_{dd} is lowered down from $1V$ to $0.9V$ but the execution time is increased by 27.45%. Generally speaking, the energy reduction caused by DVS is not worth the loss in performance when the voltage level is close to the critical speed.

2) Reconfiguration Selection Heuristics: Existing DVS algorithms are not applicable when DCR is employed since the energy consumption as well as the impact on task's execution time cannot be calculated from energy models. DCR algorithms for real-time systems proposed in [4] [5] are also not applicable since they only support soft task deadlines. Given a static slack allocation, we can assign the most energy efficient configuration point which does not stretch the execution time over the allocated slack. As long as the slack allocation is safe, we can always ensure that the schedulability is satisfied. Therefore, we adapt a heuristic motivated by the uniform constant slowdown scheme which is proved to be optimal in ideal DVS [32]. The optimal common slowdown factor η is given by the system utilization. In our approach, we only consider a finite number of discrete configuration points. Therefore, as shown in Algorithm 1, for each task, we select the configuration point with minimum energy consumption but equal or shorter execution time compared to the one decided by η. Note that we use each task's execution under the highest voltage in V and largest cache configuration in C as the base case (v_{base}, c_{base}).

Algorithm 1 Configuration selection heuristic.

$\eta = \sum_{i=1}^{m} \frac{T_i(v_{base},c_{base})}{p_i}$
for all task $\tau_i \in$ T **do**
  $T_i^{bound} = T_i(v_{base}, c_{base})/\eta$;
  Assign τ_i with (v_{j_i}, c_{k_i}) which satisfied:
  1) $E_i(v_{j_i}, c_{k_i})$ is the minimum;
  2) $T_i(v_{j_i}, c_{k_i}) \leqslant T_i^{bound}$;
end for
return (v_{j_i}, c_{k_i}), $\forall i \in [1, m]$

F. Procrastination

To further control static energy consumption, it is beneficial to put the system into a sleep mode instead of keep it idle since the static power could be lower by order-of-magnitude. As discussed in Section IV-D, taking various system components into consideration leads to much higher critical speed compared to DVS-only scenario. In other words, the idle periods are getting longer. However, bringing the system into sleep mode and vice versa requires certain amount of overhead in terms of energy and time. In order to reduce the number of mode switches, we need to make the busy/idle periods as long as possible. One way to achieve this is to procrastinate task execution when it is safe. We adapt the task procrastination algorithm from [33] into our scheduler. We ensure that when the system gets shut down, there is no unfinished job in the system to avoid cold start penalty. Hence, the shutdown overhead consists of the energy consumed for circuit logic recharging and dirty data flushing-back in the cache subsystem.

Algorithm 2 outlines our procrastination scheme. A timer is enabled when idle period starts and disabled when busy period starts. A newly arrived task during idle period will update the timer if it has earlier deadline compared to the current earliest one. Upon timeout, all delayed ready tasks are executed in EDF order. Arriving tasks during busy period are allowed to

preempt as usual. Note that *time* represents the current time instant and (v_{j_i}, c_{k_i}) stands for the chosen configuration point for task τ_i. Here, $isEarlier[i]$ records whether the current job of τ_i's deadline is earlier than all the pending tasks in the system when it arrives.

Algorithm 2 Task procrastination algorithm.

$isEarlier[i]$ is initialized to be all false;
Current earliest deadline of delayed jobs $\delta = 0$;
On arrival of a new job of task τ_r:
$d_r = p_r \cdot \lceil \frac{time}{p_r} \rceil$; $actUtil = \sum_{i=1}^{m} \frac{T_i(v_{j_i}, c_{k_i})}{p_i}$;
if System is in sleep mode or is idle **then**
  if *timer* is disabled **then**
    $timer = \lfloor (1 - actUtil) \cdot p_r \rfloor$;
    $\delta = d_r$; $isEarlier[r] = true$;
  else
    if $d_r < \delta$ **then**
      for all τ_i in ready task queue **do**
        if $isEarlier[i]$ is true **then**
          $delayed = delayed + \frac{time - p_i \cdot \lfloor \frac{time}{p_i} \rfloor}{p_i}$;
        end if
        $timer = \lfloor (1 - actUtil - delayed) \cdot p_r \rfloor$;
        $\delta = d_r$; $isEarlier[r] = true$;
      end for
    end if
  end if
end if

V. Experiments

A. Experimental Setup

To evaluate the effectiveness of our approach, we select benchmarks from MediaBench [30], MiBench [31] and EEMBC [34] to from four task sets with each consists of 5 to 8 tasks. While DVS techniques usually use synthetic tasks for evaluation, we choose real benchmarks so that bus and memory hierarchy behaviors of real applications can be revealed. Table I lists our task sets. Task Set 1 consists of tasks from MediaBench, Set 2 from EEMBC, Set 3 from MiBench and Set 4 is a mixture of all three suites. In Set 4, the two benchmarks from EEMBC are set to iterate 100 times in order to make their size comparable with others.

We adapt processor constants described in Section III-B from [13]: $V_{bs} = -0.7V$, $L_d = 37$, $\alpha = 1.5$. Energy and power values used in energy models for the memory hierarchy are collected from CACTI [12]. The on-chip buses and off-chip buses have capacitance of $5pF$ and $60pF$, respectively. The on-chip buses have equal frequency as the processor (decided by the current voltage level) while off-chip buses (from L1 to L2 and from L2 to memory) have a frequency of $400MHz$ and $200MHz$, respectively. The bus static power is assumed to be 50% of the average dynamic power consumption, which is a conservative estimation [11].

We assume cache dirty data write back and circuit logic recharging penalty for shutdown to be $85\mu J$ and $300\mu J$. Therefore, the total shutdown overhead is $385\mu J$ [13]. Based on our energy model, idle power dissipation for the system, which comes from the static energy consumption of processor, cache hierarchy, bus lines and memory, is assumed to be $240 + 200 + 58 + 291 = 789mW$. System in sleep mode is assumed to consume $80\mu W$ of power. Hence, the shutdown threshold interval is $0.49ms$ and any interval whose length is shorter than this threshold will not lead to a shutdown. In this paper, the energy estimation framework (whose input is gathered from SimpleScalar [29]) as well as the scheduling simulator are implemented in C++.

TABLE I
Task sets consisting of real benchmarks.

Set	Tasks
1	cjpeg, djpeg, mpeg2, pegwit, rawcaudio
2	A2TIME01, BaseFP01, BITMNP01, RSPEED01, TBLOOK01
3	CRC32, susan, dijkstra, rijndael, adpcm, qsort, FFT, stringsearch
4	cjpeg, rawdaudio, pegwit, A2TIME01, RSPEED01, pktflow, FFT, dijkstra

B. Results

We consider the following techniques:

- **DVS**: Traditional DVS without DCR which assigns the lowest feasible[2] voltage level.
- **CS-DVS**: Leakage-aware DVS without DCR which assigns lowest feasible voltage level above the critical speed decided by processor energy consumption.
- **CS-DVS-G**: Leakage-aware DVS without DCR which assigns lowest feasible voltage level above the critical speed decided by system-wide energy consumption.
- **DVS-DCR**: Traditional DVS + DCR which assigns the configuration point for minimizing the dynamic energy consumption of processor and cache subsystem.
- **CS-DVS-DCR**: Leakage-aware DVS + DCR which assigns the most energy-efficient while feasible configuration point above the critical speed decided by the energy consumption of processor and cache subsystem.
- **CS-DVS-DCR-G**: Leakage-aware DVS + DCR which assigns the most energy-efficient while feasible configuration point above the critical speed decided by system-wide energy consumption.
- **CS-DVS-DCR-G-P**: Leakage-aware DVS + DCR for system-wide energy minimization which also employs task procrastination.

1) Task Set: For all the task sets described in Table I, we compare all the above listed techniques across various system utilizations (from 0.1 to 0.9 in a step of 0.1). All the results are the average of all task sets and are normalized to **DVS** scenario. Figure 10 shows the normalized system-wide overall energy consumption using different approaches. The first observation is that, for DVS-only approaches, considering other system components (CS-DVS-G) can achieve 12.8% additional energy savings on average (up to 26.6%) compared with traditional leakage-aware DVS (CS-DVS). Generally, applying DVS and DCR together (DVS-DCR) outperforms traditional DVS (DVS) and CS-DVS-G across all utilization rates by 66.3% and 42.1% on average, respectively. Our approach, system-wide leakage-aware DVS + DCR (CS-DVS-DCR-G), outperforms CS-DVS-G by 47.6% on average. It can

[2]"Feasible" means that it satisfies the slack allocation in Section IV-E.

be observed that leakage-aware and leakage-oblivious DVS + DCR approaches behave similarly when the system utilization ratio is beyond 0.5. It is because both of them are inclined to select similar configuration points which have voltage levels above the critical speed (V_{dd} is around 0.8 to 0.9). In other words, in these scenarios, DVS-DCR does not make inferior DVS decisions, which can lead to dominating leakage power, due to limited available slack. However, when the utilization ratio is low, CS-DVS-DCR-G can achieve around 4.6 - 23.5% more energy savings than DVS-DCR since CS-DVS-DCR-G does not lower down the voltage level below the critical speed.

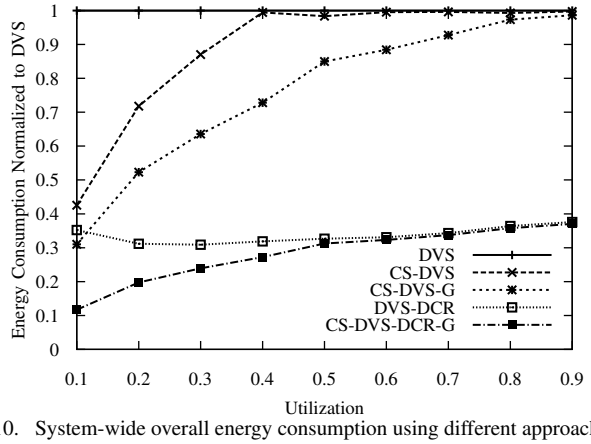


Fig. 10. System-wide overall energy consumption using different approaches.

Figure 11 (a) shows the reduction in static energy consumption by using CS-DVS-DCR-G compared to DVS-DCR as well as CS-DVS-DCR. CS-DVS-DCR-G gains averagely about 26.5% static energy savings over DVS-DCR across all utilizations and around 44.4% in low utilization cases. Compared with CS-DVS-DCR, taking memory and system buses into consideration results in 7.1% static energy savings on average (up to 14.4%). This improvement is not as significant as the difference between CS-DVS and CS-DVS-G since, as shown in Section IV-D, memory and bus lines have relatively less impact on the critical speed compared with cache subsystem. In our study, dynamic procrastination does not bring remarkable savings with respect to overall energy consumption. The reason is that the shutdown threshold is relatively short compared with the execution time of real benchmarks in our approach. Therefore, even without procrastination, the idle periods during system execution normally are longer than the threshold which makes them beneficial to shutdown the system. In other words, the total sleep time for both CS-DVS-DCR-G and CS-DVS-DVS-G-P are close. It is expected, however, if the task sizes are small, reductions of overall energy will be more significant [13]. To illustrate the effectiveness of procrastination, Figure 11 (b) shows the result in idle energy savings. It can be observed that 26.9% savings on average can be achieved across all utilization rates by using CS-DVS-DCR-G-P.

VI. Conclusion

Leakage power can adversely impact any system energy optimization techniques including both dynamic voltage scaling and cache reconfiguration. Employing both DVS and DCR together can lead to greater system energy savings than using them independently. In this paper, we presented an efficient approach to integrate DVS and DCR that is aware of leakage power. Our energy estimation framework comprehensively incorporates various system components which makes the analysis more accurate and effective. Our studies demonstrate that considering only one aspect (e.g., dynamic energy) or one component (e.g., DVS-capable processor) can lead to inaccurate conclusion in terms of overall energy, since critical speed will vary depending on the various components in the system. We focus on reducing system-wide dynamic and static energy consumption. We also integrate task procrastination to further save the energy consumption when the system is idle. Our approach is shown to be superior than both leakage-aware DVS techniques by around 47.6% and outperform leakage-oblivious DVS + DCR techniques by up to 23.5%.

References

[1] K. Lahiri and A. Raghunathan, "Power analysis of system-level on-chip communication architectures," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2004, pp. 236–241.

[2] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1702–1714, 1999.

[3] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2000, pp. 241–243.

[4] W. Wang, P. Mishra, and A. Gordon-Ross, "SACR: Scheduling-aware cache reconfiguration for real-time embedded systems," in *Proceedings of IEEE International Conference on VLSI Design*, 2009, pp. 547–552.

[5] W. Wang and P. Mishra, "Dynamic reconfiguration of two-level caches in soft real-time embedded systems," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, 2009, pp. 145–150.

[6] G. Buttazzo, *Hard Real-Time Computing Systems*. Kluwer, 1995.

[7] J. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[8] B. Doyle, R. Arghavani, D. Barlage, S. Datta, M. Doczy, J. Kavalieros, A. Murthy, and R. Chau, "Transistor elements for 30nm physical gate lengths and beyond," *Intel Technology Journal*, vol. 6, pp. 42–54, 2002.

[9] J. Kao, S. Narendra, and A. Chandrakasan, "Subthreshold leakage modeling and reduction techniques," in *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM, 2002, pp. 141–148.

[10] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[11] R. Rao, H. Deogun, D. Blaauw, and D. Sylvester, "Bus encoding for total power reduction using a leakage-aware buffer configuration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 12, pp. 1376 – 1383, dec. 2005.

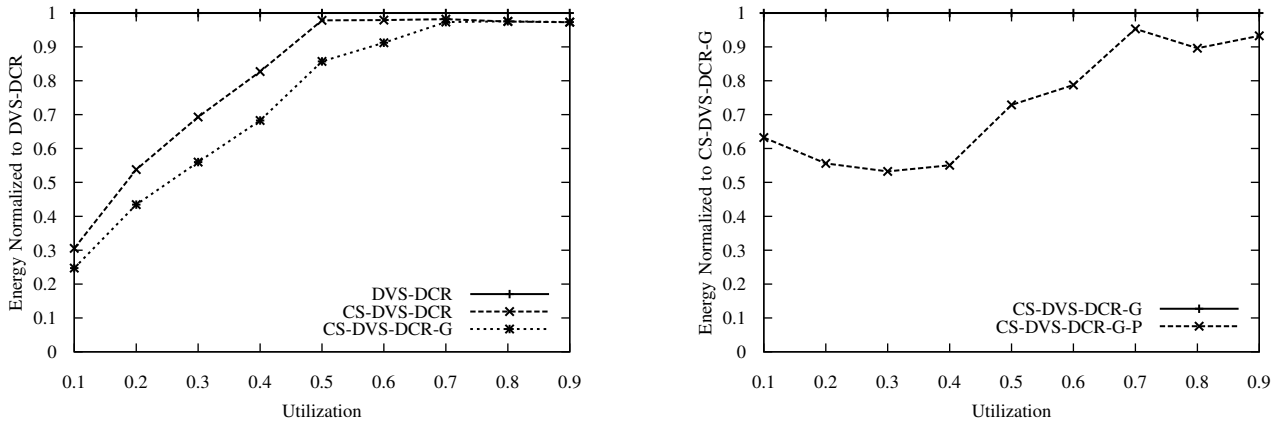
[12] HP, *CACTI, HP Laboratories Palo Alto, CACTI 5.3*, http://www.hpl.hp.com/, 2008.

[13] R. Jejurikar, C. Pereira, and R. K. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of Design Automation Conference*, 2004, pp. 275–280.

[14] A. C. Nacul and T. Givargis, "Dynamic voltage and cache reconfiguration for low power," in *Proceedings of Design, Automation and Test Conference in Europe*, 2004, p. 21376.

[15] R. Jejurikar and R. Gupta, "Energy-aware task scheduling with task synchronization for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 1024–1037, 2006.

[16] S. Zhang, K. Chatha, and G. Konjevod, "Approximation algorithms for power minimization of earliest deadline first and rate monotonic schedules," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2007, pp. 225–230.



(a) (b)

Fig. 11. Results: (a) Static energy consumption using DVS-DCR and cs-DVS-DCR; (b) Idle energy consumption using cs-DVS-DCR and cs-DVS-DCR-P.

[17] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Proc. International Symposium on Low Power Electronics and Design ISLPED '04*, 9–11 Aug. 2004, pp. 78–81.

[18] X. Zhong and C. Xu, "System-wide energy minimization for real-time tasks: Lower bound and approximation," in *Proceedings of International Conference on Computer-Aided Design*, 2006, pp. 516–521.

[19] J.-J. Chen and T.-W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor," in *LCTES '06: Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*. New York, NY, USA: ACM, 2006, pp. 153–162.

[20] W. Wang and P. Mishra, "PreDVS: Preemptive dynamic voltage scaling for real-time systems using approximation scheme," in *Proceedings of Design Automation Conference*, 2010, pp. 705–710.

[21] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," in *Proceedings of Design Automation Conference*, 2007, pp. 234–237.

[22] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 2000, pp. 90–95.

[23] J.-W. Chi, C.-L. Yang, Y.-J. Chen, and J.-J. Chen, "Cache leakage control mechanism for hard real-time systems," in *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*. New York, NY, USA: ACM, 2007, pp. 248–256.

[24] E. Hallnor and S. Reinhardt, "A unified compressed memory hierarchy," 2005, pp. 201 – 212.

[25] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 6, pp. 362–387, 2005.

[26] W. Wang and P. Mishra, "Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems," in *Proceedings of IEEE International Conference on VLSI Design*, 2010, pp. 357–362.

[27] W. Fornaciari, D. Sciuto, and C. Silvano, "Power estimation for architectural exploration of hw/sw communication on system-level buses," in *Proc. Seventh International Workshop on Hardware/Software Codesign (CODES '99)*, May 3–5, 1999, pp. 152–156.

[28] C. Talarico, J. W. Rozenblit, V. Malhotra, and A. Stritter, "A new framework for power estimation of embedded systems," *Computer*, vol. 38, no. 2, pp. 71–78, 2005.

[29] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future microprocessors: The simplescalar tool set," University of Wisconsin-Madison, Tech. Rep., 1996.

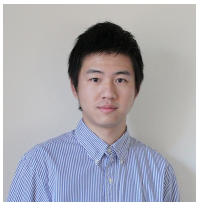
[30] C. Lee, M. Potkonjak, and W. H. Mangione-smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proceedings of International Symposium on Microarchitecture*, 1997, pp. 330–335.

[31] M. Guthaus, J. Ringenberg, D.Ernest, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of IEEE International Workshop on Workload Characterization*, 2001, pp. 3–14.

[32] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of Real-Time Systems Symposium*, 2001, pp. 95–105.

[33] Y.-H. Lee, K. Reddy, and C. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, July 2003, pp. 105–112.

[34] EEMBC. EEMBC, The Embedded Microprocessor Benchmark Consortium. http://www.eembc.org/.



Weixun Wang (S'08) received his B.E. degree in software engineering from the Software Institute, Nanjing University, Nanjing, China, in 2007. He is currently pursuing his Ph.D. degree in the Department of Computer and Information Science and Engineering, University of Florida, USA. His research interests include the area of design automation of embedded systems with focus on dynamic cache reconfiguration, energy optimization, temperature management, design space exploration and lossless data compression.

Prabhat Mishra (S'00-M'04-SM'08) received the B.E. degree from Jadavpur University, India, the M.Tech. degree from the Indian Institute of Technology, Kharagpur, and the Ph.D. degree from the University of California, Irvine – all in computer science. He is currently an Associate Professor with the Department of Computer and Information Science and Engineering, University of Florida. His research interests include design automation of embedded systems, reconfigurable architectures, and functional verification. He has published two books, nine book chapters and more than 70 research articles in premier journals and conferences. His research has been recognized by several awards including an NSF CAREER Award in 2008, two best paper awards (VLSI Design 2011 and CODES+ISSS 2003), several best paper award nominations (including DAC 2009 and VLSI Design 2009), and 2004 EDAA Outstanding Dissertation Award from the European Design Automation Association. Dr. Mishra currently serves as Information Director of ACM Transactions on Design Automation of Electronic Systems, Guest Editor of IEEE Design & Test of Computers, and as a program/organizing committee member of several ACM and IEEE conferences.