Leakage-Aware Energy Minimization using Dynamic Voltage Scaling and Cache Reconfiguration in Real-Time Systems

Weixun Wang and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL
{wewang,prabhat}@cise.ufl.edu

***Abstract*—System optimization techniques are widely used to improve energy efficiency as well as overall performance. Dynamic voltage scaling (DVS) is acknowledged 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 saving cache subsystem's energy consumption. As the manufacturing technology scales into the order of nanometers, leakage current, both in the processor and cache subsystem, becomes a significant contributor in the overall power dissipation. In this paper, we efficiently integrate processor voltage scaling and cache reconfiguration together that is aware of leakage power to minimize overall system energy consumption. Experimental results demonstrate that our approach outperforms existing techniques by on average 12 - 23%.**

I. Introduction

Energy conservation is a primary optimization objective in embedded systems design since these systems are generally limited by battery lifetime. Various low-power techniques focus on different components in the system. Dynamic voltage scaling (DVS) [1] 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 frequency. At the same time, memory hierarchy, especially cache subsystem, has become comparable to the processor with respect to the contribution in overall energy consumption [2]. 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 [3]. It will be promising to use both DVS and DCR to conserve both processor and memory energy dissipations.

Real-time embedded systems bring particular design and optimization considerations in order to satisfy the deadline constraints imposed on each task. In real-time systems, all tasks have to finish execution before their deadlines to ensure correct system behavior. Earliest Deadline First (EDF) [4], which employs a dynamic priority scheme, is the most commonly used scheduling algorithm in the real-time research community. Under EDF, a preemptive task set is said to be *schedulable* as long as the system utilization rate is not more than 1 [5]. Essentially, DVS in real-time systems slows down the processor, thus saves power/energy, at the cost of stretched task execution time. On the other hand, applying DCR in such systems may lead to similar effects so that it needs to be done in a controlled way. 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 with preemptive periodic task sets.

In the last decade, we have observed a continuous CMOS device scaling process in which higher transistor density and smaller device dimension lead to increasing leakage (static) power consumption. This is mainly due to the proportionally reduced threshold voltage level with the supply voltage. Lower threshold voltage results in larger leakage current which mainly consists of subthreshold current [6] and reverse bias junction current [7]. Study has shown that leakage power is increased by five times in each technology generation [8] and can exceed above half of the total power dissipation [9]. On-chip caches nowadays contribute a significant share of the system leakage power. Static energy is projected to account for near 70% of the cache subsystem's budget in 70nm technology [9]. Furthermore, higher temperature have adverse impact on leakage power in both processor [10] and cache [11]. Therefore, decisions should be made judiciously on whether slowing down the system to save dynamic power or switching the system to sleep mode to reduce static power. While existing techniques try to control the leakage power along with DVS [12], extra consideration needs to be taken when DCR is also employed as described in this paper.

The rest of the paper is organized as follows. Related works are discussed in Section II. Section III presents the system model 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 dynamic voltage scaling 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 long as possible through various directions including task scheduling,

This work was partially supported by NSF grant CCF-0903430.

voltage selection and worst-case execution time estimation [13][14][15]. Meanwhile, a number of existing works pay attention to control processor leakage power in real-time systems [16][12][17][18]. Lee et al. [16] propose a scheduling algorithm to minimize leakage energy consumption by procrastinating currently ready tasks to enlarge system idle periods based on a non-DVS platform. Jejurikar et al. [12] 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 consumption to compensate the reduction in dynamic energy. They also propose a procrastination scheduling technique to maximize processor idle intervals. Chen et al. [17] address the same problem in a rate-monotone scheduling system. They also propose a procrastination scheme based on energy consumption evaluation [18]. However, none of the above techniques considers dynamic cache reconfiguration.

Cache reconfiguration has drawn considerable amount of research efforts in both general-purpose [19] and real-time [20][21] systems. Reconfigurable cache architectures are proposed in [2][22]. Wang et al. [20] employed DCR in real-time systems by dynamically utilizing static profiling information to tune the cache configuration in order to achieve significant energy savings. 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 [23][24]. Chi et al. [25] applied these techniques in hard real-time systems. However, none of these approaches takes processor voltage into consideration. Nacul et al. [26] 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 when leakage energy dominates the total consumption.

Our proposed research in this paper integrates DVS and DCR in hard real-time systems to reduce both dynamic and static energy consumption. We take a step forward by examining the correlation between the energy models of processor and cache subsystem. We statically assign voltage level and cache configuration to each task based on slack allocation method and procrastinate task execution at runtime to shutdown the processor/cache when beneficial to achieve more energy savings. Extensive experiments show that our approach can result on average 46% energy savings compared to DVS-only systems and up to 12 - 23% extra savings compared to leakage-oblivious DVS + DCR technique [26].

III. System Model

In this section, we describe our system model including the task model and the energy model. We assume that DVS and DCR are available in the target system. 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 architecture, with reconfigurable parameters including cache size, line size and associativity, which can be tuned to l different configurations C$\{c_1,c_2, \ldots ,c_l\}$.

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). 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:

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

subject to,

$$\sum_{i=1}^{m} \frac{T_i(v_{j_i}, c_{k_i})}{p_i} \leqslant U_{EDF} \tag{2}$$

B. Energy Model

1) Processor Energy Model: Since short circuit power is negligible [27], the energy consumed in a processor mainly comes from dynamic and static power. The dynamic power can be presented as:

$$P_{proc}^{dyn} = C_{eff} \cdot V_{dd}^2 \cdot f \tag{3}$$

where C_{eff} is the total effective switching capacitance of the processor, V_{dd} is the supply voltage level and f is the operating frequency. We adapt the analytical processor energy model from [7], whose accuracy has been verified with SPICE simulation. The threshold voltage V_{th} can be presented as:

$$V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs} \tag{4}$$

where V_{th1}, K_1, K_2 are all constants and V_{bs} represents the body bias voltage. As mentioned in Section I, static current mainly consists of the subthreshold current I_{subth} and the reverse bias junction current I_j. Hence, the static power is given by:

$$P_{proc}^{sta} = V_{dd} \cdot I_{subth} + |V_{bs}| \cdot I_j \tag{5}$$

where I_j is approximated as a constant and I_{subth} can be calculated by:

$$I_{subth} = K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}} \tag{6}$$

where K_3, $K4$ and K_5 are constant parameters. Obviously, to avoid junction leakage power overriding the gain in lowering I_{subth}, V_{bs} has to be constrained (between 0 and -1V). Let P_{proc}^{on} be the intrinsic energy needed for keeping the processor on (idle energy). The processor power consumption can be computed as:

$$P_{proc} = P_{proc}^{dyn} + P_{proc}^{sta} + P_{proc}^{on} \tag{7}$$

The cycle length, t_{cycle}, is given by a modified alpha power model which is verified by SPICE simulation:

$$t_{cycle} = \frac{L_d \cdot K_6}{(V_{dd} - Vth)^{\alpha}} \tag{8}$$

where K_6 is a constant. In this paper, we estimate L_d to be the average logic depth of all instructions' critical path in the processor. Table I lists the constants for 70nm technology.

Let CC denote the number of clock cycles executed, the processor energy consumption becomes:

$$E_{proc} = P_{proc} \cdot \underline{CC} \cdot t_{cycle} \tag{9}$$

TABLE I
CONSTANTS FOR 70NM TECHNOLOGY

Const	Value	Const	Value	Const	Value
K_1	0.063	K_6	5.26×10^{-12}	V_{th1}	0.244
K_2	0.153	K_7	-0.144	I_j	4.80×10^{-10}
K_3	5.38×10^{-7}	V_{dd}	$[0.5, 1.0]$	C_{eff}	0.43×10^{-9}
K_4	1.83	V_{bs}	$[-1.0, 0.0]$	L_d	37
K_5	4.19	α	1.5	L_g	4×10^6

2) Cache Energy Model: Cache energy consumption also consists of dynamic energy E^{dyn}_{cache} and static energy E^{sta}_{cache}:

$$E_{cache} = E^{dyn}_{cache} + E^{sta}_{cache} \tag{10}$$

The number of cache accesses *num_accesses*, cache misses *num_misses* and clock cycles *CC* are obtained from simulation using SimpleScalar [28] for any given task and cache configuration. Let E_{access} and E_{miss} denote the energy consumed per cache access and miss, respectively. Therefore, we have:

$$E^{dyn}_{cache} = num\_accesses \cdot E_{access} + num\_misses \cdot E_{miss} \tag{11}$$

$$E_{miss} = E_{offchip\_access} + \underline{E_{\mu P\_stall}} + E_{block\_fill} \tag{12}$$

$$E^{sta}_{cache} = P^{sta}_{cache} \cdot CC \cdot \underline{t_{cycle}} \tag{13}$$

where $E_{offchip\_access}$ is the energy required for fetching data from off-chip memory, $E_{\mu P\_stall}$ is the energy consumed when the processor is stalled due to cache miss, E_{block_fill} is for cache block refilling after a miss and P^{sta}_{cache} is the static power consumption of cache. We collect E_{access}, P^{sta}_{cache} and E_{block_fill} from CACTI [29] for all cache configurations and adopt numbers for others from [22].

IV. LEAKAGE-AWARE DVS AND DCR

Our approach addresses three major challenges including profiling information analysis to determine the critical speed, configuration selection and task procrastination to significantly reduce overall energy consumption while meeting task deadlines. This section describes each of these steps in detail.

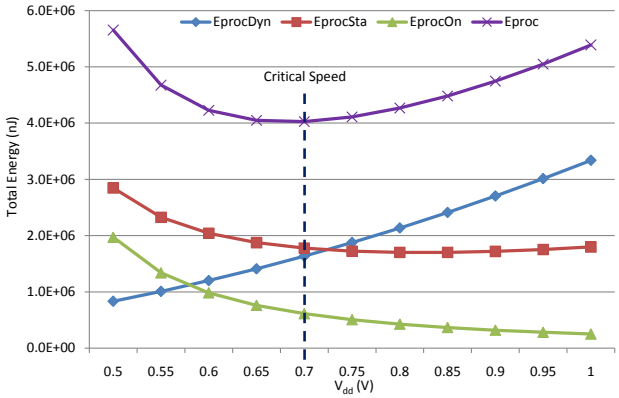


Fig. 1. Processor energy consumption E_{proc} for executing *cjpeg*: $E_{procDyn}$ is the dynamic energy, $E_{procSta}$ is the static energy and E_{procOn} is the intrinsic energy needed to keep processor on.

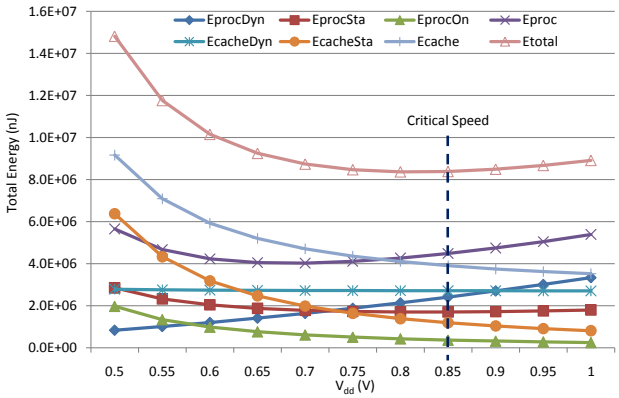


Fig. 2. Overall system energy consumption E_{total} of the processor and cache subsystem (configured to 16KB,32B,2-way) for executing *cjpeg*: $E_{cacheDyn}$ and $E_{cacheSta}$ are the dynamic and static cache consumption, respectively.

A. Critical Speed

The critical speed for processor voltage scaling defines a point which the processor speed cannot be slowed down below otherwise DVS will no longer be beneficial [12]. The dynamic power consumption of processors, which is exclusively considered in traditional DVS, is usually a convex and increasing function of the operating frequency. However, since lower processor speed stretches the task execution time which leads to higher static energy consumption, the 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. Note that as described in Section III-B, there exists strong correlation between the energy models of the processor and cache subsystem. Since different cache configuration leads to different miss ratio and miss penalty cycles, the number of clock cycles (CC) required to execute an application is decided by the cache configuration, which directly affects processor's energy consumption as shown in Equation (9). On other hand, the length of each clock cycle (t_{cycle}) and the energy consumed at the stalled processor during an off-chip access ($E_{\mu P\_stall}$) are determined by the processor frequency, which directly affects

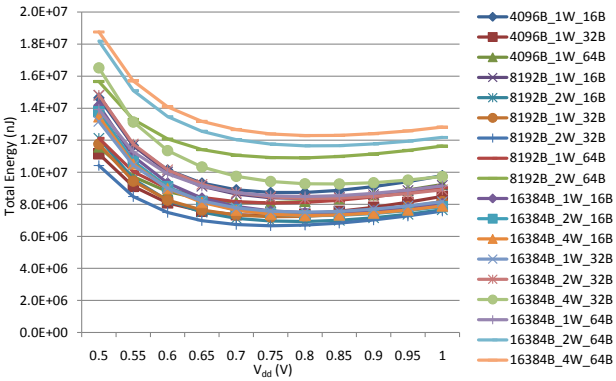


Fig. 3. Total energy consumption across all cache configurations for executing *cjpeg*.

cache energy consumption as shown in Equation (12) and (13). The former fact results in drastically varying processor energy consumption across different cache configurations. The later factor leads to faster critical speed compared to the DVS-only scenario. It is caused by the impact of increasing cache static power on the total static energy consumption when the processor is slowed down.

Now we show a simple motivating example in which a single benchmark (*cjpeg*) is executed under all processor voltage levels. It can be observed that in Figure 1, when only processor energy is considered, the critical speed is achieved at V_{dd} = 0.7V. However, as shown in Figure 2, with respect to the total amount of energy consumption, combining DVS and DCR increases the critical speed to around V_{dd} = 0.85V. In fact, when DCR is applied, different cache configuration will lead to different critical speeds.

B. Real-Time Scaling and Reconfiguration

We define a *configuration point* as a pair of processor voltage level and cache 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. In fact, we observe that cache configurations behave quite consistently across different processor voltage levels as shown in Figure 3. The cache configuration favored by *cjpeg*, 8KB cache size with 32B line size and 2-way associativity, outperforms all the other configurations with respect to energy consumption. Although there are some exceptions that large cache configurations with highest voltage level could have slightly fast performance, they normally consume much more energy. Hence, we setup the profile table of each task consisting of its favored cache configurations with all the voltage levels higher than the critical speed.

Any existing static slack allocation scheme can be employed in our approach. For each task, we assign the configuration point which is most energy efficient while does not stretch the task execution beyond the deadline decided by the allocated slack. As long as the slack allocation is safe, we can always ensure that the schedulability condition (Equation (2)) is satisfied. Any DVS algorithm that can generate optimal or approximated-optimal voltage assignment is also applicable though most of them are more computationally complex. In this paper, we use a heuristic algorithm motivated by the uniform constant slowdown scheme which is proved to be optimal in continuous voltage scaling [30]. The optimal common slowdown factor η is given by the system utilization rate. In our approach, we only consider a finite number of discrete configuration points as defined above. Therefore, for each task, we select the configuration point with minimum energy consumption but equal or shorter execution time compared to the one decided by the optimal slowdown factor. 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}), which is used in the optimal slowdown factor calculation. Algorithm 1 illustrates our method.

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}), $\forall i \in [1,m]$

C. 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-A, taking cache into consideration leads to even faster critical speed compared to leakage-aware DVS-only scenario. In other words, for a same task set, the idle periods during which there is no active task are getting longer. However, bringing the system into sleep mode and vice versa requires certain amount of overhead in terms of energy and timing. In order to reduce the number of processor 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 no time constraint will be violated. We adapt the task procrastination algorithm from [16] into our EDF scheduler. We ensure that when the system gets shut down, there is no unfinished job in the system. This avoids cold start penalty being introduced since otherwise resumed task after wakeup has to refetch its data from memory. Hence, the shutdown overhead consists of the energy consumed for circuit logic recharging and dirty data flushing-back in the cache subsystem provided write-back policy is used.

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 absolute deadline compared to the current earliest deadline. Upon timeout, all delayed ready tasks are executed in EDF order. Arriving tasks during busy period are allowed to preempt as normal EDF scheduler does. 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 at the time when it arrives. Also, $p_r \cdot \lceil \frac{time}{p_r} \rceil$ and $p_r \cdot \lfloor \frac{time}{p_i} \rfloor$ are essentially the absolute deadline and the arrive time of τ_i's current job.

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[31], MiBench[32] and EEMBC[33] 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 cache behaviors of real applications can be revealed. Table II 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 [12]: $V_{bs} = -0.7V$, $L_d = 37$, $\alpha = 1.5$. We assume dirty data write back and circuit logic recharging penalty for shutdown to be $85\mu J$ and $300\mu J$. Idle power for processor and cache subsystem are assumed to be $240mW$ and $36mW$. System in sleep mode is assumed to consume $80\mu W$ of power. Hence, the shutdown threshold interval is 1.14ms and any interval whose length is shorter than the threshold will not lead to a shutdown. Energy model as well as the scheduling simulator are implemented in C++.

TABLE II
Task sets consisting of real benchmarks.

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

B. Results

We compare the following techniques across various system utilizations (from 0.1 to 0.9):

- **DVS**: Traditional DVS without DCR which assigns lowest voltage level to each task whenever possible.
- **CS-DVS**: Leakage-aware DVS without DCR which never assigns a processor speed level below the critical speed.
- **CS-DVS-P**: Leakage-aware DVS without DCR which employs task procrastination.
- **DVS-DCR**: DVS + DCR without leakage awareness which assigns the configuration point that best fits the slack.
- **CS-DVS-DCR**: Leakage-aware DVS + DCR which generates profile table in aware of leakage power and never assigns a configuration point below the critical speed.
- **CS-DVS-DCR-P**: Leakage-aware DVS + DCR which also employs task procrastination.

Note that all the results are the average of all task sets and are normalized to **DVS** scenario.

Figure 4 shows total energy consumption (processor + cache) of different approaches. The first observation is that generally, applying DVS + DCR outperforms DVS-only across all utilization rates by 43% on average. Our approach (cs-DVS-DCR) outperforms leakage-aware DVS (cs-DVS) by 46% on average. Above the critical speed point, as expected, leakage-aware and leakage-oblivious approaches behave almost the same way since they are inclined to make similar decisions. But when the utilization ratio is low, cs-DVS-DCR achieves around 12 - 23% (up to 55%) energy savings compared to DVS-DCR. Figure 5 (a) shows the reduction in static energy

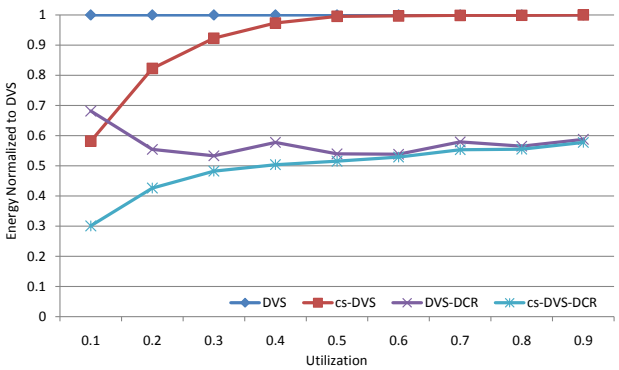
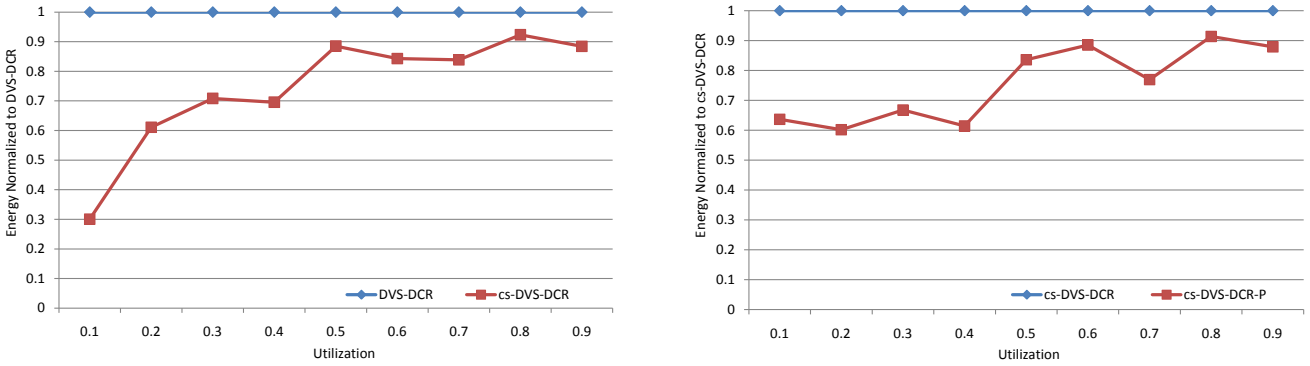


Fig. 4. Total energy consumption of different approaches.



(a) (b)

Fig. 5. Results: (a) Static energy consumption of DVS-DCR and cs-DVS-DCR; (b) Idle energy consumption of cs-DVS-DCR and cs-DVS-DCR-P.

consumption by using cs-DVS-DCR compared to DVS-DCR. Our approach gains averagely about 25% static energy savings across all utilizations and around 42% in low utilization cases. Note that with respect to total energy consumption, techniques with and without procrastination show similar energy savings. It is mainly due to the fact that the dynamic and static energy of real benchmarks we used dominates the idle energy consumption. To illustrate the effectiveness of procrastination, Figure 5 (b) shows the result in idle energy savings. It can be observed that around 10 - 40% savings can be achieved across all utilization rates by using cs-DVS-DCR-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. We focus on reducing both dynamic and static overall 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 46% and outperform leakage-oblivious DVS + DCR techniques by up to 42%.

References

[1] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems" *IEEE TCAD*, vol. 18, pp. 1702–1714, 1999.

[2] A. Malik et al., "A low power unified cache architecture providing power and performance flexibility" *ISLPED*, 2000.

[3] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation" *Micro*, 1999.

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

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

[6] J. A. Butts et al., "A static power model for architects" *Micro*, 2000.

[7] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads" *ICCAD*, 2002.

[8] S. Borkar, "Design challenges of technology scaling" *Micro*, 1999.

[9] 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.

[10] L. Yuan, S. Leventhal, and G. Qu, "Temperature-aware leakage minimization technique for real-time systems" *ICCAD*, 2006.

[11] H. Noori et al., "The effect of temperature on cache size tuning for low energy embedded systems" *GLSVLSI*, 2007.

[12] R. Jejurikar, C. Pereira, and R. K. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems" *DAC*, 2004.

[13] R. Jejurikar and R. Gupta, "Energy-aware task scheduling with task synchronization for embedded real-time systems" *IEEE TCAD*, vol. 25, pp. 1024–1037, 2006.

[14] S. Zhang, K. Chatha, and G. Konjevod, "Approximation algorithms for power minimization of earliest deadline first and rate monotonic schedules" *ISLPED*, 2007.

[15] S. Oh, J. Kim, S. Kim, and C. Kyung, "Task partitioning algorithm for intra-task dynamic voltage scaling" *ISCAS*, 2008.

[16] Y.-H. Lee, K. Reddy, and C. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems" *ECRTS*, 2003.

[17] J.-J. Chen and T.-W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor" *LCTES*, 2006.

[18] J.-J. Chen and T.-W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems" *ICCAD*, 2007.

[19] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache" *DAC*, 2007.

[20] W. Wang, P. Mishra, and A. Gordon-Ross, "Sacr: Scheduling-aware cache reconfiguration for real-time embedded systems" in *VLSI Design*, 2009.

[21] W. Wang and P. Mishra, "Dynamic reconfiguration of two-level caches in soft real-time embedded systems" *ISVLSI*, 2009.

[22] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems" *ACM TECS*, vol. 6, pp. 362–387, 2005.

[23] 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" *ISLPED*, 2000.

[24] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction" i*Micro*, 2002.

[25] J.-W. Chi, C.-L. Yang, Y.-J. Chen, and J.-J. Chen, "Cache leakage control mechanism for hard real-time systems" *CASES*, 2007.

[26] A. C. Nacul and T. Givargis, "Dynamic voltage and cache reconfiguration for low power" *DATE*, 2004.

[27] H. J. M. Veendrick, "Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits" *IEEE Journal of Solid-State Circuits*, vol. 19, no. 4, pp. 468–473, Aug 1984.

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

[29] CACTI, *CACTI, HP Labs, CACTI 4.2*, http://www.hpl.hp.com/.

[30] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems" in *RTSS*, 2001.

[31] C. Lee, M. Potkonjak, and W. H. Mangione-smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems" *Micro*, 1997.

[32] M. Guthaus, J. Ringenberg, D.Ernest, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite" *WWC*, 2001.

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