Runtime Architecture Adaptation for Energy Management in Embedded Real-Time Systems

H. Wang*, I. Koren†, and C.M. Krishna†
* Marvell Semiconductor, Inc. Marlborough, MA 01752
†University of Massachusetts, Amherst, MA 01003

Abstract—**Energy consumption has long been recognized as an important performance metric for many real-time and embedded systems. The traditional approach to energy-aware computing is to dynamically scale the voltage: this results in a significant drop in the energy consumed at the cost of a slowdown of the computation.**

In this paper, we explore a complementary approach to energy-aware real-time computing: that of runtime architecture configuration. As embedded real-time systems become ever more complex, the processors used will no longer be the bare-bones pipelines traditionally used, but rather high-end processors capable of meeting the timing needs of increasingly demanding applications. Also, the price of superscalar processors continues to fall, which allows them to be considered even for relatively cost-sensitive applications. Such high-end processors lend themselves to dynamic architecture adaptation. We describe how to exploit such adaptation and show that architecture adaptation when combined with dynamic voltage scaling, provides significant advantages over dynamic voltage scaling alone.

I. Introduction

Energy management is especially important in embedded real-time systems. Conventionally, this has been achieved by dynamic voltage scaling [24]. We present here a complementary approach: architecture adaptation.

Modern processors have many features whose goal is to improve execution rate. They use large load/store queues, multiple functional units, reorder buffers, branch predictors, banks of registers, and other mechanisms by which to enhance performance through speculative execution. Each of these hardware components consumes energy and the extent to which each of these improves performance varies from one application to the next. This offers up an opportunity to select individual hardware configurations for each task with the best execution time/energy profile[1]. This is the architecture adaptation approach studied in this paper.

While there has been some prior work reported on reconfiguration for real-time applications, such work is largely confined to a study of FPGAs [3], [7], [8], [19]. The reconfiguration of conventional pipelined architectures as an adaptation step to conserve energy in real-time applications is a largely unexplored topic (however, some recent work has considered procedures for selecting appropriate configurations during processor design [2]). This is in sharp contrast to dynamic voltage scaling, which has been the focus of intense activity in the real-time community since the 1990s [24].

[1]As pointed out in [1], the baseline overheads of adaptive processors are typically very low.

In what follows we describe an approach to select the appropriate architecture configuration from an available repertoire of such configurations. We do not focus on techniques by which the architecture can be reconfigured: these have been described elsewhere in the computer architecture literature (see, for example, [4], [22]).

The novel contribution of this paper consists of an algorithm that permits architecture adaptation on top of a conventional, dynamic voltage scaling algorithm. We are not aware of any prior, systematic, approach to integrate the architecture adaptation of general-purpose pipelines into a framework of dynamic voltage scaling for real-time, cyber-physical, applications.

The rest of this paper is organized as follows. We start by providing several experimental results which indicate that there is indeed some advantage to architecture adaptation. Then, we present a greedy offline algorithm to select the appropriate architecture configuration for each task based on its Worst Case Execution Time (WCET). This is followed by a description of how to carry out online resource reclamation if, as is usually the case, tasks do not consume their WCET.

II. Some Motivational Examples

In order to obtain some concrete figures for the impact of architecture configuration on the energy consumption, we carried out several studies using the Simplescalar [5] and Wattch [6] simulation framework. The following well-known benchmarks from the Mibench suite [9] were used: `basicmath`, `cjpeg`, `crc`, `dijkstra`, `djpeg`, `sha`, `fft`, `tiffmedian`, `patricia`, `mpeg2dec`, `mpeg2enc`, `rawcaudio`, and `rawdaudio`. The baseline architecture configuration (against which other configurations are assessed) and eight other configurations are shown in Tables I and II, respectively. We vary the load-store queue (LSQ) and the reorder buffer (ROB). These two are commonly implemented as banked buffers requiring only simple circuitry to power down some banks. In addition, there are three fetch policies: one involving no fetch throttling and the other two alternating one fetch cycle with one or two idle cycles. Fetch throttling is also simple to implement, and imposes a negligible hardware overhead. Table III shows the execution time and energy consumed by each configuration at high voltage ($v = 1.2$ V). It can be seen that there is considerable scope for energy savings by selecting a suitable architecture configuration, as long as sufficient time is available in the schedule. On average, for the 13 benchmarks considered, the

Config	basicmath		crc		djpeg		sha		cjpeg		dijkstra	
	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy
Baseline	84.080	0.366	15.261	0.106	6.221	0.048	3.506	0.022	19.340	0.120	18.344	0.116
1	85.890	0.335	15.338	0.093	6.222	0.043	3.511	0.018	19.528	0.104	18.473	0.100
2	94.346	0.341	16.206	0.089	6.228	0.041	3.538	0.018	20.863	0.101	19.046	0.096
3	85.076	0.342	15.632	0.099	6.266	0.044	3.570	0.020	19.633	0.111	18.564	0.107
4	87.088	0.312	15.705	0.087	6.266	0.040	3.575	0.017	19.804	0.096	18.696	0.093
5	94.639	0.316	16.547	0.083	6.229	0.038	3.540	0.016	20.987	0.093	19.174	0.088
6	86.593	0.334	16.411	0.098	6.730	0.044	3.585	0.020	20.232	0.109	19.035	0.105
7	88.485	0.303	16.484	0.085	6.730	0.039	3.590	0.016	20.412	0.094	19.151	0.091
8	96.125	0.307	17.326	0.082	6.731	0.037	3.616	0.015	21.531	0.090	19.631	0.086

Config	fft		tiffmedian		patricia		mpeg2dec		mpeg2enc		rawcaudio		rawdaudio	
	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy
0	50.873	0.279	47.425	0.334	99.693	0.357	41.800	0.263	38.031	0.262	17.687	0.075	13.332	0.059
1	51.678	0.248	47.632	0.297	101.119	0.331	42.748	0.230	38.345	0.235	19.295	0.066	14.825	0.053
2	56.085	0.244	57.298	0.293	110.283	0.341	47.307	0.228	42.339	0.230	22.781	0.069	18.204	0.056
3	52.402	0.261	49.109	0.318	101.160	0.338	42.221	0.247	40.655	0.242	17.681	0.069	13.333	0.055
4	53.162	0.231	49.304	0.280	102.681	0.314	43.207	0.215	41.070	0.216	19.295	0.062	14.825	0.050
5	56.958	0.227	58.385	0.275	111.568	0.323	47.673	0.212	43.014	0.213	22.781	0.065	18.204	0.053
6	54.031	0.254	51.073	0.317	102.732	0.332	42.864	0.240	46.316	0.254	17.706	0.067	13.458	0.054
7	54.782	0.224	51.292	0.275	104.059	0.306	43.876	0.208	46.645	0.223	19.294	0.060	14.816	0.048
8	58.454	0.221	60.227	0.269	112.700	0.314	48.154	0.205	47.500	0.214	22.781	0.063	18.205	0.051

TABLE III: Energy and Time at 1.2 Volts for the eight configurations

Clock rate	2 GHz
Process parameters	45 nm
Threshold Voltage	0.2398 V
Supply Voltage Range	0.75 to 1.2 V
Fetch, Issue, Decode Commit widths	4-way
Fetch queue size	16
Instruction queue size	64
Load-store queue (LSQ) size	32
Branch prediction	2K entry, bimodal
Int. functional units	4 ALU, 1 mult/div
FP functional units	4 ALUs, 1 mult/div
L1 D-cache	32 KB, 2-way, writeback
L1 I-cache	32 KB, 2-way, writeback
Combined L2 cache	256 KB, 4-way
L2 hit time	20 cycles
Main memory hit time	100 cycles

TABLE I: Baseline configuration

Config	LSQ	ROB	Fetch
Baseline	32	64	no fetch throttling
1	32	64	fetch 1/stop 1
2	32	64	fetch 1/stop 2
3	16	32	no fetch throttling
4	16	32	fetch 1/stop 1
5	16	32	fetch 1/stop 2
6	8	16	no fetch throttling
7	8	16	fetch 1/stop 1
8	8	16	fetch 1/stop 2

TABLE II: Architecture configurations

lowest energy configuration requires 27% less energy than the baseline configuration. Also, note that the minimum-energy configuration varies with the application. For example, if there is no constraint on execution time, configuration 7 is minimum-energy for `basicmath`, while configuration 8 is preferred for `crc`.

III. Review of Related Work

Over the last decade, a few authors have considered architecture adaptation [1], [18], [21], [29]. These works largely focus on general-purpose applications. In addition, there has been the work on reconfigurable FPGA-based hardware, as mentioned earlier.

By contrast, there have been only a few studies focusing on the architecture adaptation of pipelined processors for real-time systems. Hughes, *et al.*, studied its use in multimedia applications [10]. The architecture was varied by changing the issue width, the instruction window size and the number of functional units.

Recently, a more generally applicable, but static, approach has been proposed by Zeng, *et al.* [26]. In this approach, the choice of the configuration and voltage level is made by solving an integer programming problem and using information gained by prior (to system operation) profiling of the workload under consideration. This is a static approach: the choice is made by running the optimization algorithm over a hyper-period equal to the least common multiple (LCM) of the periods of the individual tasks comprising the workload. The architecture was varied by changing the branch prediction algorithm and, the cache size and organization.

The work presented in this paper assumes that task deadlines are hard, i.e., that missing a deadline is not acceptable. Further, it is a dynamic approach, which is able to reclaim time released from tasks which finish earlier than their worst-case execution time, in order to improve the efficiency with which other tasks are run.

IV. Fine-grained Voltage Scaling

In this section, we assume that any desired voltage level, in the range $(v_T, v_H]$, can be applied to the processor, where v_H and v_T are the high (baseline) voltage and threshold voltage, respectively. From basic voltage scaling theory, the voltage at which a given slowdown, σ, can be attained is given by:

$$v = \frac{(2v_T\sigma + \lambda) + \sqrt{(2v_T\sigma + \lambda)^2 - 4v_T^2\sigma^2}}{2\sigma} \tag{1}$$

where $\lambda = (v_H - v_T)^2/v_H$. The energy consumed in running the *entire* task at this voltage is the fraction $(v/v_H)^2$ of the energy consumed in running it at voltage v_H.

For each configuration c ($c = 0, 1, \cdots 8$), we can obtain by profiling experiments the associated execution time and energy consumption of each task, T, at voltage v_H, as was done in Table III. Let $E_{i,c}(\tau)$ denote the energy consumption of task T_i when using configuration c and at a voltage $v_{i,c}(\tau)$ which sets its execution time to τ (the energy function is only defined for those values of τ for which there exists some voltage in $(v_T, v_H]$ under which c can execute task T_i in time τ). Then, we have the following result, the (easy) proof of which is omitted due to space restrictions.

Observation 1: Consider any two configurations, c and d, and some task, T_i. If there exists time τ such that $v_{i,c}(\tau) < v_{i,d}(\tau)$ and $E_{i,c}(\tau) > E_{i,d}(\tau)$, then for all $t > \tau$, we will have $E_{i,c}(t) > E_{i,d}(t)$.

Observation 1 allows us to prune out configurations from consideration. If, for any execution time τ, we have $v_{i,c}(\tau) < v_{i,d}(\tau)$ and $E_{i,c}(\tau) > E_{i,d}(\tau)$, it follows that configuration c for task T_i is dominated by d for all times $t > \tau$. In other words, if configuration d takes less energy to execute a particular workload in τ seconds than configuration c does, *and* if configuration c requires a higher supply voltage than d in order to execute in the same time, then d is to be preferred for all times beyond τ as well.

To illustrate this, consider the `rawdaudio` benchmark. In Table IV, we present the energy consumption of each configuration for selected execution times. Row j of the table consists of the energy consumption for the various configurations at the high-voltage execution time of configuration j. The rows are sorted in order of increasing execution times. Consider the row corresponding to configuration 3. Its execution time at high voltage is 13.3326 and the corresponding energy is 0.0553. The energy consumed at the same execution time by configuration 0 is 0.0593. Moreover, configuration 0 consumed this energy while running at a lower voltage than v_H (since its high-voltage execution time is less than this time). Hence, beyond 13.3326, configuration 0 is always dominated by configuration 3 and can be pruned out from consideration over this interval. Consider the row corresponding to the high-voltage execution time of configuration 2. Here, we have configuration 2 consuming 0.0562 which is greater than the minimum of items to the left of it on that row. Hence, we cannot use Observation 1 to prune out configuration 2 at this time; we have to wait until the subsequent row (corresponding to the high-voltage execution of configuration 5) before pruning it out.

When a certain amount of time to run a task has been assigned, we select a configuration by comparing the energy cost of all available configurations (that have not been pruned out) at the voltage associated with the configuration and assigned execution time, and selecting the least expensive such configuration.

The key performance measure is the ratio of the energy consumed by the combination of dynamic voltage scaling and architecture adaptation (DVSAA) to that consumed by dynamic voltage scaling (DVS) alone. This is plotted as a function of the normalized execution time, which is the execution time divided by the execution time of the baseline configuration at high voltage. Figure 1 contains the ratio for some selected programs. Figure 2 provides this ratio for each of the 13 tasks, over a small range of normalized execution time. Sharp drops in the curves indicate that a new, more energy-efficient, configuration is available for the indicated execution time. At the end, for large normalized execution times, the ratio rolls off again. This reflects the fact that the baseline configuration, for those execution times, is already being executed at its lower voltage limit. As a result, increasing the available execution time beyond that point does not save the baseline configuration any energy. However, since the high-voltage execution times of the other configurations are greater, their low-voltage execution times are also greater than that of the baseline. There is therefore an interval over which making available more execution time does not lower the baseline configuration energy consumption but does reduce it for one or more other configurations. Overall, the reduction in energy due to the use of DVSAA vs. DVS can be as high as 25%.

Note that the energy savings arise from reconfiguring the hardware; using a different technology is not expected to change the energy ratios by much (obviously, the absolute energy levels will be different). For example, consider an old 0.18 micron technology, using a voltage range of 0.9 to 2 Volts and a threshold voltage of 0.5V. Figure 3 shows the results for a selected subset of the workload considered earlier. These results are relevant since older technology is often used in embedded applications for two reasons: cost and robustness. Older technology is significantly cheaper, and that can make the difference in a highly cost-sensitive consumer application. More mature, larger feature-sized, technology also makes for an intrinsically lower susceptibility to soft upsets and tends to exhibit a lower transient failure rate.

Finally, we should point out that the number of reconfigu-

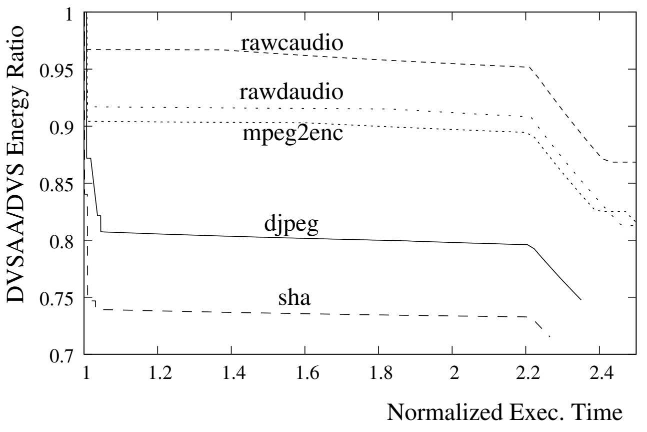
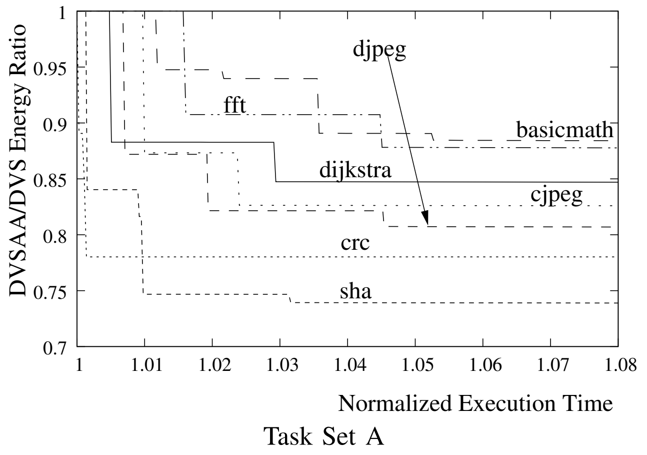


Fig. 1: Energy Ratio of DVSAA to DVS for Selected Tasks (45 nm technology)

Config	time	Energy Consumption								
		Config 0	Config 3	Config 6	Config 7	Config 1	Config 4	Config 2	Config 5	Config 8
0	13.3320	0.0593								
3	13.3326	0.0593	0.0553							
6	13.4582	0.0586	0.0546	0.0537						
7	14.8155	0.0516	0.0481	0.0473	0.0482					
1	14.8248	0.0516	0.0481	0.0473	0.0482	0.0526				
4	14.8254	0.0516	0.0481	0.0473	0.0482	0.0526	0.0497			
2	18.2039	0.0399	0.0372	0.0365	0.0369	0.0403	0.0381	0.0562		
5	18.2041	0.0399	0.0372	0.0365	0.0369	0.0403	0.0381	0.0562	0.0531	
8	18.2046	0.0399	0.0372	0.0365	0.0369	0.0403	0.0381	0.0562	0.0531	0.0514

TABLE IV: `rawdaudio` Energies for Various Configurations



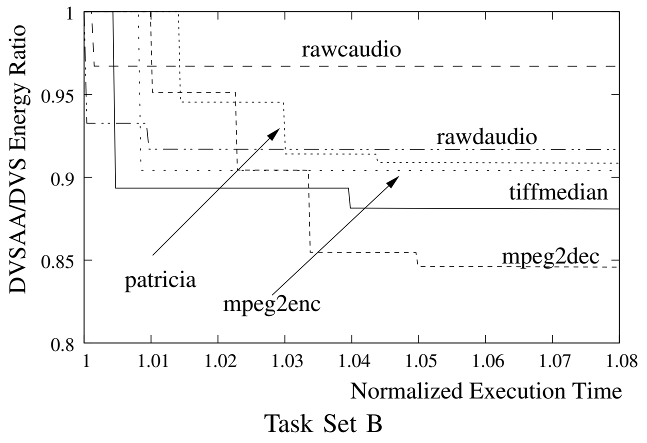
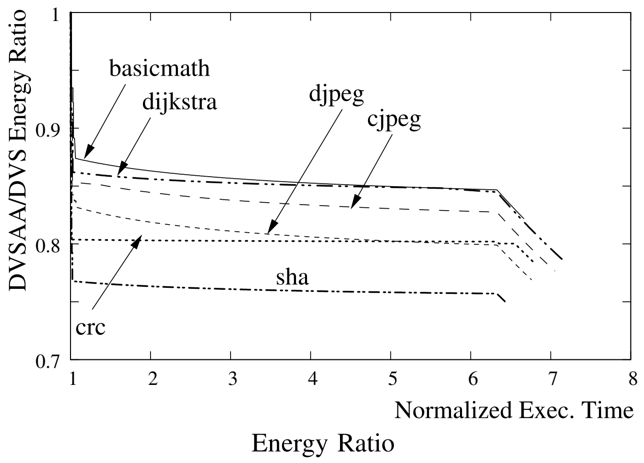


Fig. 2: Energy Ratio of DVSAA to DVS: Detail (45 nm technology)



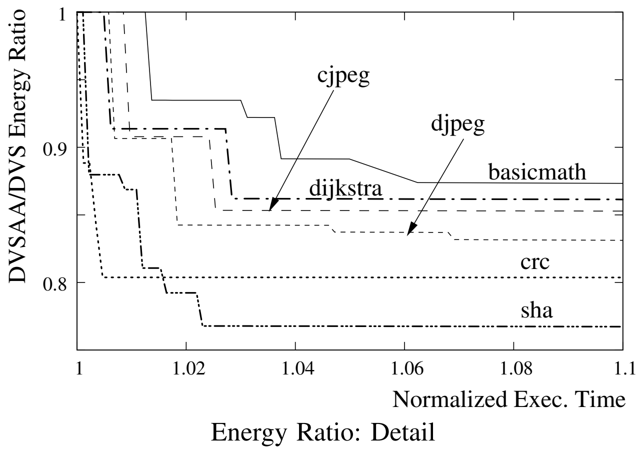


Fig. 3: Energy Ratio of DVSAA to DVS for Selected Tasks: 0.18 Micron Technology

rations is likely to be quite small during the execution of any given task. Reconfiguration happens only when an opportunity to do so occurs due to some task completing ahead of its estimated worst-case execution time. The number of reconfigurations that occur during the execution of a task is therefore related to the number of such early task completions that are nested between the start of that execution and its completion. Further, each reconfiguration action is quite lightweight. For example, changing from the baseline configuration to Configuration 1 only requires that we turn on fetch throttling. For this reason, the overhead of reconfigurations is likely to be negligible, and is not modeled in this paper.

A. Architecture Configuration and Voltage Scaling Scheme

We now turn to the issue of how to incorporate architectural reconfiguration into a voltage-scaling scheme. The overall framework is assumed to be the Earliest Deadline First (EDF) algorithm and the deadlines of the periodic tasks are assumed to equal their periods. It is well-known that the necessary and sufficient condition for EDF-schedulability is that the total utilization be bounded above by 1 [15], [17] where the processor utilization for a given task is defined as the task's WCET divided by its period. EDF is used in this paper as one of the two most widely used real-time scheduling algorithms. The same approach can equally well be used for any other scheduling algorithm, such as the Rate Monotonic algorithm. Similarly, the assumption that the relative deadline of a task equals its period is used for concreteness; the same approach can be used in any setting where it is possible to evaluate the task-set schedulability.

Our approach has offline and online components. The offline component assumes that all tasks run to their worst-case execution times and uses a greedy strategy to select the configurations to be used. The online component allows the reclaiming of time from tasks that do not consume all their assigned time and adjusts the allocation of other tasks appropriately.

1) Offline Algorithm: The assignment of execution times to each task is a nonlinear optimization problem. The greedy offline heuristic that we consider for this purpose is shown in Figure 4. It starts by checking if the task set is schedulable at the fastest configuration for each task and at high voltage. If not, the user is notified that the task set is not schedulable and the algorithm exits. If it is schedulable, then we can proceed to assign the execution times and architecture configurations for each task. The starting point is the execution time at high voltage of the baseline configuration; as additional time is provided to the task, more degraded (and economical) configurations may become feasible.

We then greedily augment utilization to each task until the worst-case task set utilization becomes 1. At this point, we have the final offline time assignment for all tasks. For each task, given its time assignment, we can choose the configuration that minimizes the energy consumed. This time and configuration assignment is the output of the offline algorithm. The stepsize, namely the unit in which additional time increments are assigned, has an impact on the output, as we shall see later.

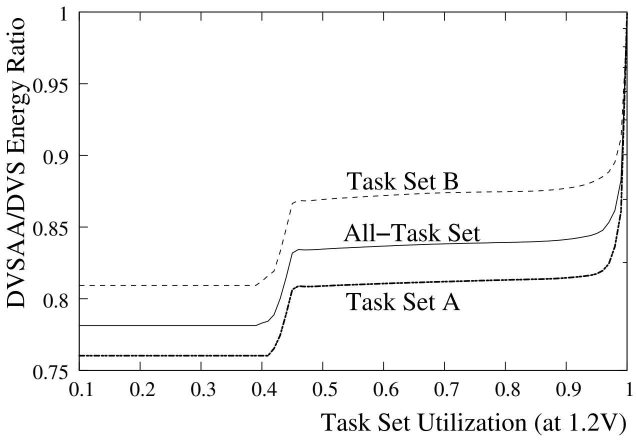
We present numerical examples for the 45 nm technology parameters. The task sets used are Set A comprising `basicmath`, `dijkstra`, `crc`, `sha`, `cjpeg`, `djpeg`, `fft`, and Set B consisting of `tiffmedian`, `patricia`, `mpeg2dec`, `mpeg2enc`, `rawcaudio`, and `rawdaudio`. In addition, we considered a task set generated by the union of Sets A and B. The task periods in this example are selected to be such that the utilizations of all these tasks are equal. Under our offline scheduling algorithm and assuming that each of the tasks runs to its worst-case execution time, the ratio of the DVSAA energy consumption to that of traditional DVS is shown in Figure 5. As utilizations increase, the number of available architecture configurations capable of meeting task deadlines decreases; as a result, when utilization approaches 1, the benefit from architecture adaptation disappears.

The impact of the step size on the performance of this algorithm is shown in Figure 6 for a range of step sizes over which the algorithm performs well; step sizes should therefore be kept small. Small steps allow fine adjustments to be made to the time allocations to the various tasks; bigger steps allow one to leap over local minima. The greedy algorithm is sufficiently fast that one can afford to make a search over a number of step size values to select the best. Beyond this range of step sizes, the performance of the offline algorithm degrades quickly.

2) Online Time Reclamation Methodology: Two possible approaches exist for online time reclamation. The first is to only adjust the voltage online while using the configuration

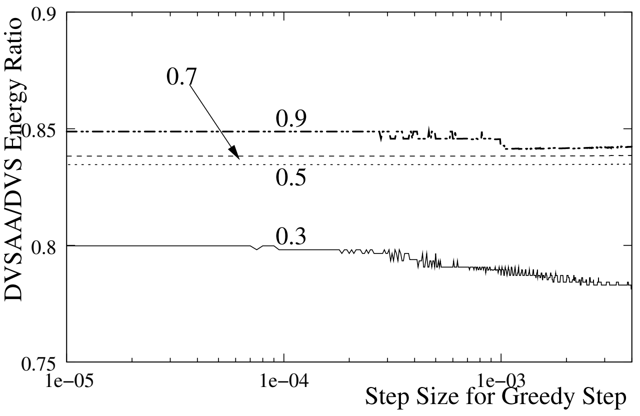
```
// time(task, n, v) is the time taken to run
// the specified task under configuration n
// and voltage v.
// totUtil() is the total utilization of the
// system assuming the prevailing time
// assignment
// delta is a prespecified constant which
// controls the step size of the greedy search
// minTask() returns the task which, under
// some configuration, yields the greatest
// savings in energy if provided with an
// additional utilization of utilStep
// minConfig(task,time) returns the configuration
// under which the given task running for the
// WCET<=time consumes the least energy

for (task=0; task<numberOfTasks; task++)
    timeAssigned[task]=time(task,baseline,vH);
if (totUtil() > 1.0)
    return failure;

// Increment time assignments until the processor
// is fully utilized under worst case execution
// times.
while (totUtil() < 1.0) {
    utilStep = min(stepSize, 1.0-totUtil());
    chosenTask = minTask(utilStep, timeAssigned);
    timeAssigned[chosenTask] +=
             utilStep*period[chosenTask];
    chosenConfig =
      minConfig(chosenTask,timeAssigned);
}
   Return configuration and time assignments
   to each task corresponding to the lowest
   energy consumption found;
```

Fig. 4: Offline Assignment Algorithm



Task Set A {`basicmath`, `dijkstra`, `crc`, `sha`, `cjpeg`, `djpeg`, `fft`} Task Set B: {`tiffmedian`, `patricia`, `mpeg2dec`, `mpeg2enc`, `rawcaudio`, `rawdaudio`}

Fig. 5: Greedy Offline Algorithm on Task Sets



Curve labels show task set utilization at 1.2 Volts and 2 GHz
Set of all 13 Tasks in Table III.

Fig. 6: Impact of Step Size on Offline Algorithm

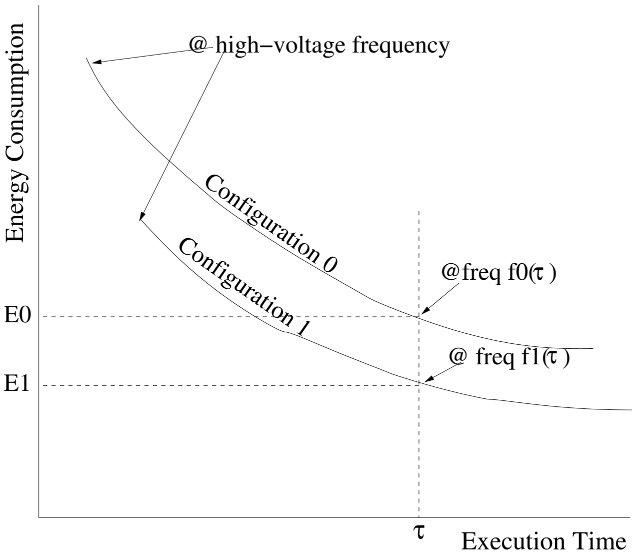


Fig. 7: Equivalent Frequencies for Various Configurations

that has been selected by the offline assignment. In such a case, one of the many published dynamic voltage scaling schemes can be used.

The second approach is to carry out an online voltage scaling *and* an architecture reconfiguration. We present here a methodology by which this can be done within the framework of any conventional dynamic voltage scaling algorithm. This methodology applies to any workload that satisfies the following uniform scaling assumption (we indicate below how to extend the methodology to cases in which this assumption does not hold).

Denote by $t(i, S, v, c)$ the time taken by configuration c running at voltage v to execute any segment S of task T_i. Then, our uniform scaling assumption is that for all $v \in (v_T, v_H]$,

$$\frac{t(i, S_1, v, c_a)}{t(i, S_1, v, c_b)} \approx \frac{t(i, S_2, v, c_a)}{t(i, S_2, v, c_b)} \tag{2}$$

for any configurations c_a, c_b and any segments S_1, S_2 of task T_i. This assumption states that the relative advantage of one configuration over another with respect to any given task is preserved across all segments of that task. That is, we do not have a situation in which the relative advantage of the configurations varies depending on which part of T_i's code we are executing. In other words, no one task consists of multiple phases, with the relative advantage of the various architectural configurations varying from one phase of *that* task to the next. (To avoid potential confusion, we stress that this restriction does not apply across tasks, only to different segments of the *same* task.)

In what follows, assume that we have some online voltage scaling algorithm, A, to handle resource reclamation. Every time we have a task arrival or completion, this algorithm outputs the frequency at which each task must be run. This frequency can be updated at every task arrival or departure point. Note that A is a conventional DVS algorithm that does not take account of architecture reconfiguration; it assumes that the baseline architecture configuration is used all the time.

Fitting architecture reconfiguration into this framework is simple. At any given frequency, different configurations generally consume differing amounts of time to execute any given task; conversely, if they are to execute it in the same time, they must be run at different frequencies. Consider Figure 7, which plots the energy against execution time for some given task under two architectural configurations 0 and 1 that complete the task in exactly the same time τ but at two different clock frequencies f_0 and f_1, respectively, However, configuration 1 consumes less energy than configuration 0 for the same amount of work. From this, we have the observation that for any τ and any nonempty time interval I, configuration 0 does the same amount of computational work over I when running the task at frequency f_0 as does configuration 1 running at frequency f_1.

For convenience, we express this equal computational progress by saying that that (f_0, c_0) and (f_1, c_1) are in the same equivalence class with respect to the given task. Note that equivalence classes are defined with respect to individual tasks, and will vary from one task to another depending on the particular need of each task for the various architectural modules.

This yields the following approach for incorporating architecture adaptation into a DVS-resource reclaiming framework:

> *Suppose the DVS resource reclaiming algorithm assigns some task to execute over a given interval I at frequency f_b under the baseline architecture configuration. Then, if (f_c, c) is in the same equivalence class for this task as $(f_b, baseline)$, assigning frequency f_c to this task in I under configuration c will ensure that all task deadlines are met.*

This leads to our methodology for integrating reconfiguration into any DVS resource reclaiming algorithm: use the DVS algorithm to specify a frequency for the baseline configuration but use (if it exists) an equivalent frequency and configuration under which less energy is consumed.

More formally, denote as before by $E_{i,c}(\tau)$ the energy

```
// e_i: WCET of T_i
// P_i: Period of T_i
select_frequency():
   Use the lowest frequency f such that
   U_1+U_2+...+U_n <= f/f_max;

upon task_release(T_i):
   Set U_i = e_i/P_i;
   select_frequency();

upon task_completion(T_i):
   Set U_i to k/P_i;
   // k is the actual time used
   select_frequency();
```

Fig. 8: Pillai-Shin Algorithm for EDF

Task	Period	Config x		Config y	
		WCET	Energy	WCET	Energy
1	200	90	5	110	3.5
2	150	75	3	100	2

$v_H = 1.2$ V, $v_T = 0.2398$ V, clock= 2 GHz at v_H
WCET = Worst Case Execution Time at high voltage
Relative task deadline equals its period

TABLE V: Example Parameters

consumed while executing task T_i under configuration c for an overall task execution time of τ and let $f_{i,c}(\tau)$ denote the associated clock frequency. Suppose now that the DVS algorithm calls for running task T_i for duration w under frequency $f_i^{(DVS)}$. Denote by θ_i the entire execution time of task T_i when run at frequency $f_i^{(DVS)}$ under the baseline configuration. Clearly, $w \leq \theta_i$. Find the configuration, c_{min}, for which $E_{i,c}(\theta_i)$ would be minimized. Run T_i for w seconds under this configuration and clock frequency $f_{i,c_{min}}(\theta_i)$.

We turn now to relaxing the uniform scaling assumption. If the assumption does not hold across the entire task, then the task must be divided into multiple phases of execution such that the scaling assumption holds (to a reasonable approximation) in each phase. These phases can be found by profiling (by recording the average number of instructions per cycle over segments of the program under each of the configurations). The same approach as above can now be implemented with respect to the individual task phases.

Example: To illustrate our approach, we show how to integrate architecture reconfiguration into the Pillai-Shin DVS resource reclaimer [20]. This algorithm (slightly modified for an infinitely tunable frequency and voltage) is shown in Figure 8. It determines the appropriate frequency every time a task is released or completed. The voltage is assumed to be set at the minimum available level needed to sustain this clock frequency.

Suppose we have a two-task system with the characteristics shown in Table V. We make the usual assumption that the relative task deadlines equal their respective periods. At time 0, we release the first iterations of both T_1 and T_2. From the WCETs, we can determine the equivalent frequencies. Consider configuration x running at clock frequency f_0. Under configuration y, the same rate of progress in executing task 1 will be possible if it is running at frequency $\frac{110}{90}f_0 = 1.22f_0$, and in executing task T_2 at frequency $\frac{100}{75}f_0 = 1.33f_0$.

The baseline utilization is $90/200+75/150 = 0.95$. Clearly, we cannot run configuration y for either task to begin with: there is not enough time, but we can run configuration x under a slowdown of $1/0.95$, i.e., at a clock rate of 1.9 GHz (corresponding to a supply voltage of 1.79 V). If it were to run to its worst-case time, it would take a total of $75/0.95 = 78.94$ units of time.

At time 0, the first iterations of both tasks are released. Task T_2 has priority over task T_1 since its deadline of 150 is earlier. It starts running at a clock rate of 1.9 GHz. Suppose it completes at time 50, i.e., much earlier than its worst-case time. At this point, the DVS algorithm recomputes the frequency in order to reclaim the time released. The utilization calculation is now $50/150 + 90/200 = 0.783$. The baseline frequency is now set at $2\times 0.783 = 1.567$ GHz, corresponding to a supply voltage of 1.025 V. The equivalent frequency for T_1 running under configuration y is $1.22 \times 1.567 = 1.915$ GHz; the supply voltage to attain this frequency is 1.157 V.

At these frequencies, the energy required for the worst-case execution of T_1 under configuration x (the baseline) is $1.314\mu J$ and under configuration y is $1.170\mu J$. Hence, we select configuration y, set the supply voltage to 1.949 V and the clock rate to 1.915 GHz.

The system keeps running at this setting until the next iteration of T_2 is released at time 150. At this time, we recalculate the frequencies: we go back to the baseline configuration running at 1.9 GHz, and so on.

We now calculate the ratio of the energy consumption up to time 150 under DVSAA and DVS. First, we calculate the equivalent relative capacitance of both configurations. Power consumption when switching capacitance C across V volts at frequency f is proportional to CV^2f; hence the relative capacitance of the circuit for each task under each configuration can be written as

$$C_{rel} = \frac{\text{Energy at } v_H}{v_H^2 f \times WCET}. \quad (3)$$

Based on this, the relative capacitances for Task 1 under configurations x and y are $C_{x,1} = 0.01929$ and $C_{y,1} = 0.011048$, respectively; the corresponding numbers for Task 2 are $C_{x,2} = 0.013889$ and $C_{y,2} = 0.006944$.

We can now write the energy consumption ratio of DVSAA to DVS, up to time 150, as

$$\frac{E_{DVSAA}}{E_{DVS}} = \frac{C_{x,2}\cdot 1.9 \cdot 1.2^2 \cdot 50 + C_{y,1}\cdot 1.915 \cdot 1.157^2 \cdot 100}{C_{x,2}\cdot 1.9 \cdot 1.2^2 \cdot 50 + C_{x,1}\cdot 1.567 \cdot 1.025^2 \cdot 100} = 0.93$$

Note that exactly the same amount of computational work has been completed in both cases; we have additional savings of energy up to this point as a result of architecture adaptation.

V. Finite Number of Voltage Levels

The approach we have presented above can also be used when the number of voltage (and therefore frequency) levels is finite. The principal difference will be the shape of the energy-to-execution time curve for each configuration.

As an example, consider a system that is restricted to having two voltage levels, v_H and v_L. For some configuration c and task T_i, let $p_{H,i,c}$ and $p_{L,i,c}$ be the power consumption at v_H and v_L, respectively. Let the slowdown factor in switching to low voltage be $\sigma(v_L)$ and $\tau_{H,i,c}$ the execution time at v_H. Suppose the task is assigned $\tau_{H,i,c} < t < \sigma(v_L)\tau_{H,i,c}$ seconds of execution time. Let α_H be the fraction of time it is executed at v_H. Then, we can write

$$\begin{aligned}\alpha_H t + (1-\alpha_H)t/\sigma(v_L) &= \tau_{H,i,c} \\ \Rightarrow \alpha_H &= \frac{\tau_{H,i,c} - t\sigma^{-1}(v_L)}{t - t\sigma^{-1}(v_L)}\end{aligned}$$

The energy consumed will now be given by

$$\begin{aligned}E_{i,c}(t) &= \alpha_H p_{H,i,c} t + (1-\alpha_H) p_{L,i,c} t \\ &= \frac{\sigma(v_L)(p_{H,i,c} - p_{L,i,c})\tau_{H,i,c} + (\sigma(v_L)p_{L,i,c} - p_{H,i,c})t}{\sigma(v_L) - 1}\end{aligned}$$

The energy-execution time plot is therefore linear in t. It might provide better insight into this relationship by rewriting it in terms of energies. At time $\tau_{H,i,c}$, we consume $E_{i,c}(\tau_{H,i,c})$, which is obtaining by profiling. We also have from the basic voltage scaling equation that $E_{i,c}(\sigma(v_L)\tau_{H,i,c}) = \rho E_{i,c}(\tau_{H,i,c})$, where $\rho = (v_L/v_H)^2$. A straight line between these two points is the energy-execution time relationship:

$$E_{i,c}(t) = E_{i,c}(\tau_{H,i,c}) - \frac{(1-\rho)E_{c,i}(\tau_{H,i,c})}{(\sigma(v_L)-1)\tau_{H,i,c}}(t - \tau_{H,i,c}). \quad (4)$$

Examination of this equation shows that the greater the value of $E_{i,c}(\tau_{H,i,c})$, the more rapidly the energy consumption falls as additional execution time is allowed. Therefore, the gap between the curves corresponding to different configurations tends to narrow with increasing execution time, until the slowdown limit for one of the curves is reached; the gap then expands as the second line continues downwards.

Figure 9 provides an example of the difference in behavior between the fine-grained and two-level scaling approaches for the `basicmath` benchmark. To reduce clutter, results for only three configurations are shown. As is well-known, restricting the number of voltage levels to 2 reduces the energy gain one can attain. On the other hand, this is a much simpler and less expensive scheme to implement. Figures 10 and 11 provide results for the DVSAA/DVS energy consumption ratio.

VI. Conclusions

In this paper, we have outlined a dynamic adaptation approach to energy management in real-time systems. We should emphasize that architecture adaptation is complementary to voltage scaling; adaptation is finer-grained than voltage scaling and allows one to tailor the existing configuration to the available time and the needs of the task at hand. We have shown how to carry out such adaptation within the framework of a resource-reclaiming dynamic voltage scaling algorithm. Architectural adaptation can also be combined with techniques used to reduce static energy consumption [12], [16], [27].

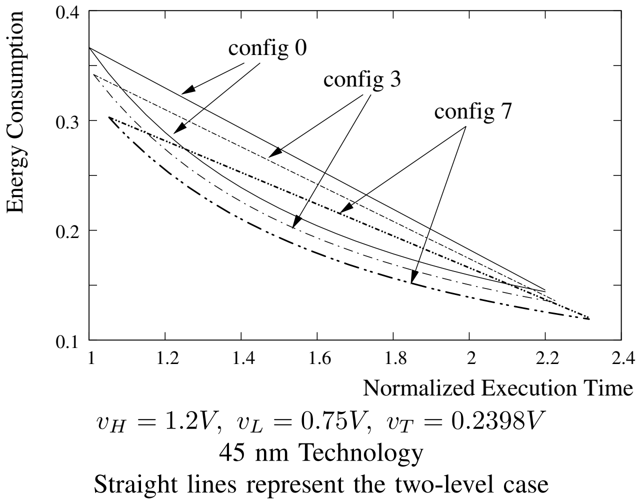


Fig. 9: Fine-Grained vs. Two-Level Voltage Scaling for `basicmath`

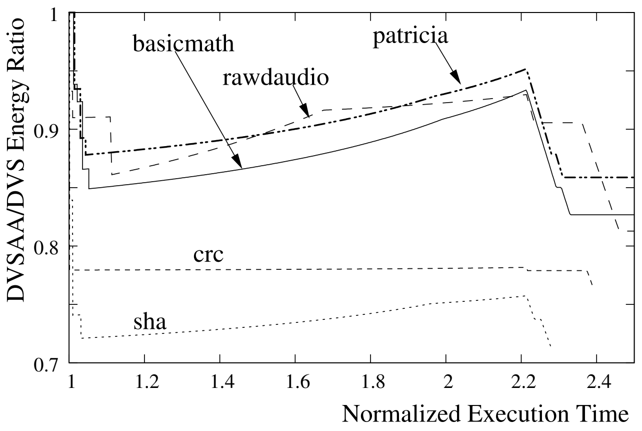


Fig. 10: Energy Ratio of DVSAA to DVS: Two Voltage Levels

Acknowledgements

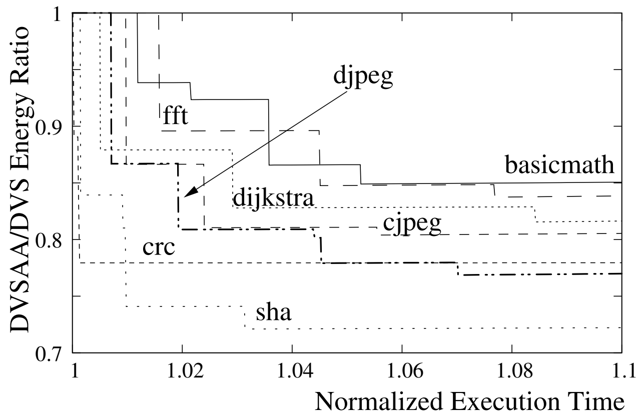
This work was partially supported by the National Science Foundation under grant CNS-0931035.

References

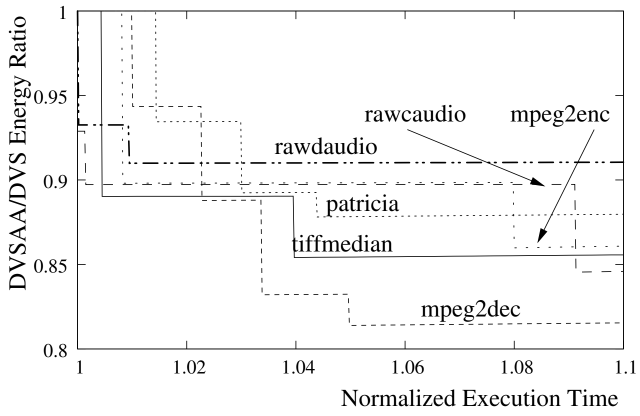
[1] D.H. Albonesi, R. Balasubramonian, S.G. Dropsho, S. Dwarkadas, E.G. Friedman, M.C. Huang, V. Kurson, G. Magklis, M.L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P.W. Cook and S.E. Shuster, "Dynamically Tuning Processor Resources with Adaptive Processing," *IEEE Computer*, Vol. 36, No. 12, December 2003, pp. 49–58.

[2] O. Azizi, A. Mahesri, B.C. Lee, S.J. Patel and M. Horowitz, "Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis," *International Symposium on Computer Architecture* (ISCA), 2010.

[3] N. Bergmann, P. Waldeck, and J. Williams, "A Catalog of Hardware Acceleration Techniques for Real-Time Reconfigurable System on Chip,"*IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2003 pp. 112–115.



Task Set A



Task Set B

Fig. 11: Energy Ratio Detail: Two Voltage Levels

[4] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P.W. Cook, and D.H. Albonesi, "An Adaptive Issue Queue for Reduced Power at High Performance," *International Workshop on Power-Aware Computer Systems*, 2000, pp. 25–39.

[5] D.C. Burger and T.M. Austin, *The Simplescalar Tool Set, Version 2.0*, Technical Report CS-TR-1997-1342, University of Wisconsin, 1997.

[6] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A Framework for Architectural-level Power Analysis and Optimizations," *International Symposium on Computer Architecture (ISCA-27)*, 2003, pp. 741–750.

[7] K. Danne, R. Muhlenbernd, and M. Platzner, Executing Hardware Tasks on Dynamically Reconfigurable Devices Under Real-Time Conditions," *International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–6.

[8] K. Danne and M. Platzner, "Periodic Real-Time Scheduling for FPGA Computers," *Workshop on Intelligent Solutions in Embedded Systems*, 2005, pp. 117–127.

[9] 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," *WWC-4: IEEE International Conference on Workload Characterization*, 2001.

[10] C.J. Hughes, J. Srinivasan and S.V. Adve, "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications," *34th International Symposium on Microarchitecture*, December 2001, pp. 250–261.

[11] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *ISLPED*, 1998, pp. 197-202.

[12] N.S. Kim, K. Flautner, D. Blaauw, and T.N. Mudge, "Drowsy Instruction Caches," *Micro-35*, 2002. pp. 148–157.

[13] C.M. Krishna and Y.-H. Lee, "Voltage-clock-scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems," *IEEE Real-Time and Applications Symposium (RTAS)*, 2000, pp. 156–165.

[14] C.M. Krishna and Y.-H. Lee, "Voltage-clock-scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems," *IEEE Transactions on Computers*, Vol. 52, No. 12, December 2003, pp. 1586–1593.

[15] C.M. Krishna and K.G. Shin, *Real-Time Systems*, McGraw-Hill, 1997.

[16] Y.-H. Lee, K.P. Reddy, and C.M. Krishna, "Scheduling Techniques for Reducing Leakage Power in Hard Real-Time Systems," *Euromicro Conference on Real-Time Systems*, 2003.

[17] J.W.S. Liu, *Real-Time Systems*, Wiley, 2000.

[18] A.C. Nacul and T. Givargis, "Dynamic Voltage and Cache Reconfiguration for Low Power," *Conference on Design, Automation and Test in Europe (DATE)*, 2004, p. 1376.

[19] R. Pellizzoni and M. Caccamo, "Hybrid Hardware-Software Architecture for Reconfigurable Real-Time Systems," *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2008, pp. 273–284.

[20] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *ACM Symposium on Operating System Principles,* 2001.

[21] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources," *International Symposium on Microarchitecture (MICRO-34)*, 2001, pp. 90–101.

[22] R. Sasanka, C.J. Hughes, and S.V. Adve, "Joint Local and Global Hardware Adaptations for Energy," *Intl. Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS), 2002, pp. 144–155.

[23] K.G. Shin and C.M. Krishna, "Performance Measures for Control Computers," *IEEE Transactions on Automatic Control*, Vol. AC-32, 1987, pp. 467–473.

[24] O.S. Unsal and I. Koren, "System-level Power-aware Design Techniques in Real-Time Systems," *Proceedings of the IEEE*, Vol. 91, No. 7, July 2003, pp. 1055–1069.

[25] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *Symposium on Operating Systems Design and Implementation (OSDI)*, 1994, pp. 13–23.

[26] G. Zeng, H. Tomiyama and H. Takada, "A Generalized Framework for Energy Savings in Hard Real-Time Embedded Systems," *IPSJ Transactions on System LSI Design Methodology*, Vol. 2, Aug. 2009.

[27] Z. Zhang, X. Kavousianos, K. Chakrabarty, and Y. Tsiatouhas, "A Robust and Reconfigurable Multi-mode Power Gating Architecture," *International Conference on VLSI Design*, 2011, pp. 280–285.

[28] Y. Zhu and F. Mueller, "Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling," *Real Time and Applications Symposium (RTAS)*, 2004.

[29] Z. Zhu and X. Zhang, "Look-Ahead Architecture Adaptation to Reduce Processor Power Consumption," *IEEE Micro*, Vol. 25, No. 4, 2005, pp. 10–19.