Energy-Efficient Embedded Software Implementation on Multiprocessor System-on-Chip with Multiple Voltages

SHAOXIONG HUA
Synopsys
and
GANG QU and SHUVRA S. BHATTACHARYYA
University of Maryland

This paper develops energy-driven completion ratio guaranteed scheduling techniques for the implementation of embedded software on multiprocessor systems with multiple supply voltages. We leverage application's performance requirements, uncertainties in execution time, and tolerance for reasonable execution failures to scale each processor's supply voltage at run-time to reduce the multiprocessor system's total energy consumption. Specifically, we study how to trade the difference between the system's highest achievable completion ratio $\mathcal{Q}^{\max}$ and the required completion ratio $\mathcal{Q}_0$ for energy saving. First, we propose a best-effort energy minimization algorithm (BEEM1) that achieves $\mathcal{Q}^{\max}$ with the provably minimum energy consumption. We then relax its unrealistic assumption on the application's real execution time and develop algorithm BEEM2 that only requires the application's best- and worst-case execution times. Finally, we propose a hybrid offline on-line completion ratio guaranteed energy minimization algorithm (QGEM) that provides the required $\mathcal{Q}_0$ with further energy reduction based on the probabilistic distribution of the application's execution time. We implement the proposed algorithms and verify their energy efficiency on real-life DSP applications and the TGFF random benchmark suite. BEEM1, BEEM2, and QGEM all provide the required completion ratio with average energy reduction of 28.7, 26.4, and 35.8%, respectively.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; D.2.2 [**Software**]: Software Engineering—*Design tools and techniques*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Energy minimization, hardware/software co-design, multiprocessor, multiple voltage, completion ratio

Authors' addresses: Shaoxiong Hua, Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043; email: huas@synopsys.com; This work was performed during his Ph.D. study in the University of Maryland, College Park; Gang Qu (contact author) and Shuvra S. Bhattacharyya, Electrical and Computer Engineering Department and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742; email: {gangqu,ssb}@glue.umd.edu.

1. INTRODUCTION

Performance guarantee and energy efficiency are becoming increasingly important for the implementation of embedded software. Traditionally, the worst-case execution time (WCET) is considered to provide performance guarantee, however, this often leads to overdesigning the system (e.g., more hardware and more energy consumed than necessary). In this paper, we consider the problem of how to implement multiprocessor embedded systems to deliver performance guarantee with reduced energy consumption.

Many applications, such as multimedia and digital signal-processing (DSP) applications, are characterized by repetitive processing on periodically arriving inputs (e.g., voice samples or video frames). Their processing deadlines, which are determined by the throughput of the input data streams, may occasionally be missed without being noticeable or annoying to the end user. For example, in packet audio applications, loss rates between 1–10% can be tolerated [Bolot and Vega-Garcia 1996], while tolerance for losses in low bit-rate voice applications may be significantly lower [Karam and Tobagi 2001]. Such tolerance gives rise to slacks that can be exploited when streamlining the embedded processing associated with such applications. Specifically, when the embedded processing does not interact with a lossy communication channel, or when the channel quality is high compared to the tolerable rate of missed deadlines, we are presented with slacks in the application that can be used to reduce cost or power consumption.

Typically, slacks arise from the run-time task execution time variation and can be exploited to improve real-time application's response time or reduce power. For example, Bambha and Bhattacharyya [2000] examined voltage scaling for multiprocessor with known computation time and hard deadline constraints. Luo and Jha [2000] presented a power-conscious algorithm and static battery-aware scheduling algorithms for distributed real-time battery-powered systems [Luo and Jha 2001]. Zhu et al. [2001] introduced the concept of slack sharing on multiprocessor systems to reduce energy consumption. The essence of these works is to exploit the slacks by using voltage scaling to reduce energy consumption without suffering any performance degradation (execution failures).

The slack we consider in this paper comes from the tolerance of execution failures or deadline missings. In particular, since the end user will not notice a small percentage of execution failure, we can *intentionally* drop some tasks to create slack for voltage scaling, as long as we keep the loss rates to be tolerable. Furthermore, much richer information than task's WCET is available for many DSP applications. Examples include the best-case execution time (BCET), execution time with cache miss, when interrupt occurs, when pipeline stalls, or when a different conditional branch happens. More important, most of these events are predictable and we will be able to obtain the probabilities that they may happen by knowing (e.g., by sampling technique) detailed timing information about the system or by simulation on the target hardware [Tia et al. 1995]. This gives another degree of freedom to explore on-line and offline voltage scaling for energy reduction.

Dynamic voltage scaling (DVS), which can vary the supply voltage and clock frequency according to the workload at run-time, can exploit the slack time generated by the workload variation and achieve the highest possible energy efficiency for time-varying computational loads [Burd et al. 2000; Qu 2001]. It is arguably the most effective technique to reduce the dynamic energy, which is still the dominate part of system's energy dissipation, despite the fast increase of leakage power on modern systems.

The most relevant works on DVS, to this paper, are on the energy minimization of dependent tasks on multiprocessor systems with multiple voltages. Hua and Qu [2005b] studied the voltage setup problem, which determines how many levels and at which values should the voltages be implemented on the multiple voltage system to maximally reduce the energy consumption. Schmitz and Al-Hashimi [2001] investigated DVS processing elements power variations based on the executed tasks, during the synthesis of distributed embedded systems, and its impact on the energy saving. Gruian and Kuchcinski [2001] introduced a new scheduling approach, LEneS, that uses list scheduling and a special priority function to derive static schedules with low energy consumption. The assignment of tasks to multiple processors is assumed to be given. Luo and Jha [2001] presented static scheduling algorithm based on critical path analysis and task execution order refinement. An on-line scheduling algorithm is also developed to reduce the energy consumption for real-time heterogeneous distributed embedded systems while providing the best-effort services to soft aperiodic tasks. The deadlines and precedence relationships of hard real-time periodic tasks are guaranteed [Luo and Jha 2002]. Mishra et al. [2003] proposed static and dynamic power-management schemes for distributed hard real-time systems, where the communication time is significant and tasks may have precedence constraints. Most recently, Hua and Qu [2005a] propose a static power management with proportional slack distribution and parallelism scheme (PDP-SPM). PDP-SPM takes advantage of both local and global static slack in multiprocessor as well as the parallelism to reduce energy consumption. However, these algorithms use the slacks to reduce energy, but they do not drop tasks to create more slacks.

Different from the above, some energy-reduction techniques on single processor have been proposed by Hua et al. [2003] for multimedia applications with tolerance to deadline misses while providing a statistical completion ratio guarantee. Lee et al. [2004] proposed a local voltage-controller scheme that allows each pipeline stages it's own voltage level and a lower cost-dynamic retiming scheme that incorporates per-stage clock delay elements to allow longer-latency pipeline stages to borrow time from shorter-latency stages. Yuan and Nahrstedt [2003] developed an energy-efficient soft real-time CPU scheduler that allocates CPU cycles periodically to tasks based on statistical demand, rather than worst case, to provide statistical performance guarantees.

Finally, we mention that early efforts on multiprocessor design range from the design space exploration algorithm [Karkowski and Corporaal 1998] to the implementation of such systems [Grbic et al. 1998; Sutton et al. 1998]. Scalable architectures and codesign approaches have been developed for the design

Table I. Characteristics of the tasks and the Processor[a]

Task	BCET	WCET
$\mathcal{A}$	(1, 80%)	(6, 20%)
$\mathcal{B}$	(2, 90%)	(7, 10%)
$\mathcal{C}$	(2, 75%)	(5, 25%)

(a) The three tasks.

Voltage (V)	Power	Delay
$v_1 = 3.3$	1	1
$v_2 = 2.4$	0.30	1.8
$v_3 = 1.8$	0.09	3.4

(b) Processor parameters.

[a](a) Execution time distribution of the three tasks. Each row shows a task's best-worst-case execution time at the reference voltage v_1 and the probability this execution time occurs at runtime. (b) Power and delay of the processor at different voltages. *power* is normalized to the power at v_1 and *delay* column gives the normalized processing time to execute the same task at different voltages.

of multiprocessor DSP systems [e.g., see Janka and Wills 2000; Scherrer and Eberle 1998]. These approaches, however, do not provide systematic techniques to handle voltage scaling, nondeterministic computation time, or completion ratio tolerance. Performance-driven static scheduling algorithms that allocate task graphs to multiple processors [Sih and Lee 1993] can be used in conjunction with best- or average-case task computation time to generate an initial schedule for our proposed methods. It can then interleave performance monitoring and voltage adjustment functionality into the schedule to streamline its performance.

2. A MOTIVATIONAL EXAMPLE

We consider a simple case where an application requires the repetitive execution of three tasks $\mathcal{A}, \mathcal{B}, \mathcal{C}$, in that order. Suppose that each iteration of "$\mathcal{A} \rightarrow \mathcal{B} \rightarrow \mathcal{C}$" must be completed in 10 CPU units (deadline) and the application can tolerate 40% of the 10,000 iterations to miss this deadline. We show how to leverage this tolerance to deadline misses for energy reduction. For simplicity, we consider a single processor system that supports multiple voltages.

Table Ia gives each task's only two possible execution time and the probabilities that they occur. For example, task $\mathcal{A}$ has its BCET, 1 CPU unit, for 80% of the time and its WCET, 6 CPU units, happens for the rest 20% of the time. Table Ib shows the normalized power consumption and processing speed of the processor at three different voltages. We now compare the following three different algorithms:

I. For each iteration, run at the highest voltage v_1 to the completion or the deadline whichever happens first.

II. For each iteration, run $\mathcal{A}$ at the highest voltage v_1 to its completion. Then, if $\mathcal{B}$ cannot be completed by **8** at voltage v_1, terminate the current iteration (i.e., not attempt to run $\mathcal{C}$ at all); if $\mathcal{B}$ can be completed before **5** at v_1, use the lowest voltage to complete it by 5; otherwise run at v_1 to its completion. Finally, we stop if $\mathcal{C}$ cannot be completed by the deadline **10** at voltage v_1 or run at the lowest voltage that can finish $\mathcal{C}$ by 10.

III. Assign **1**, **7**, and **2** (a total of 10) CPU units to $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively. During each iteration, each task can only be executed within this assigned

Table II. Completion Ratio and Energy Consumption for the Three Algorithms[a]

	$Q(\%)$	$t@v_1$	$t@v_2$	$t@v_3$	E	E@($Q = 60\%$)
I	91.5	6.94	0	0	6.94	4.55
II	91.5	4.21	4.54	0	5.57	3.65
III	60	2.56	0	4.90	3.00	3.00

[a] $t@v_1$, $t@v_2$, and $t@v_3$ are the average time that the processor operates at three voltages for each iteration; E is the average energy consumption to complete one iteration; and the last column, obtained by $E \cdot 60\%/Q$, gives the minimum average energy consumption per iteration with a 40% deadline misses tolerance.

slot. If it cannot be finished at v_1, terminate the iteration; otherwise, run at the lowest voltage that can complete the task within its assigned slot.

We generate the execution time for each task following the distribution given in Table Ia for 10,000 iterations. We then simulate each of the above three algorithms for 10,000 iterations. Table II reports their completion ratio Q, average processing time (at different voltages) per iteration, and average power consumption per iteration. We observe that (1) algorithm I gives the highest possible completion ratio[1]; (2) algorithm II achieves the same completion ratio as algorithm I, but with less energy consumption; and (3) algorithm III trades unnecessary completions (by completing only the required 60% iterations) for further energy reduction.

2.1 Contributions

Although algorithm I is a straightforward best-effort approach, the settings for algorithms II and III are not trivial. In algorithms II, how do we determine the execution strategies for the three tasks? Is it a coincidence that it achieves the same completion ratio as algorithm I? Does there exist other more energy-efficient algorithm that maintains this completion ratio? For the individual execution slots in algorithm III, how are they determined? Why can they guarantee the 60% completion ratio? Is it always more energy efficient than algorithms I and II? Finally, can we extend them to multiprocessor systems?

We answer these questions by formulating and solving the *energy minimization problem with deadline miss tolerance on multiprocessor systems*. We will show that algorithm II comes from our best-effort energy minimization algorithm (BEEM), which achieves the highest completion ratio with the provably minimum energy consumption. We also show that algorithm III is based on our offline on-line completion ratio (or quality of service) guaranteed energy-minimization algorithm (QGEM). QGEM guarantees the required completion ratio while converting the deadline miss tolerances into energy reduction via DVS. This allows us to depart from the conservative view of overimplementing the embedded software in order to meet deadlines under WCET. Our result is an algorithmic framework that integrates considerations of iterative multiprocessor scheduling, voltage scaling, nondeterministic computation time, and

[1]A 100% is not achievable in this example. For example, if all tasks have them WCETs, we need 18 CPU units and cannot make the deadline 10.

completion ratio requirement, and provides robust, energy-efficient multiprocessor implementation of embedded software for embedded DSP applications.

2.2 Paper Organization

The rest of the paper is organized as follows: in the next section, we formulate the problem of (dynamic) energy minimization with completion ratio guarantee. In section 4, we describe our practical solutions to this problem. The simulation setup and results are reported in section 5. Section 6 concludes the paper with a discussion on the limitation of our approaches and future directions.

3. PROBLEM FORMULATION

3.1 Embedded Software Model

We consider the popular and powerful *task graph*, $G = (V, E)$, model to represent the embedded software to be implemented on multiprocessor systems. Each vertex in the graph represents one computation and directed edges represent the data dependencies between vertices. For each vertex $v_i \in V$, we associate it with a finite set of possible execution time $\{t_{i,1} < t_{i,2} < \cdots < t_{i,k_i}\}$ and the corresponding set of probabilities $\{p_{i,1}, p_{i,2}, \ldots, p_{i,k_i} | \sum_{l=1}^{k_i} p_{i,l} = 1\}$ that such execution time may occur. That is, with probability $p_{i,j}$, vertex v_i requires an execution time in the amount of $t_{i,j}$. Note that t_{i,k_i} is the WCET and $t_{i,1}$ is the BCET for task v_i.

We further define the prefix sum of the occurrence probability

$$P_{i,l} = \sum_{j=1}^{l} p_{i,j} \tag{1}$$

Clearly, $P_{i,l}$ measures the probability that the computation at vertex v_i can be completed within time $t_{i,l}$ and we have $P_{i,k_i} = 1$ which means that a completion is guaranteed if we allocate CPU time to vertex v_i based on its WCET t_{i,k_i}.

A directed edge $(v_i, v_j) \in E$ shows that the data dependency between two vertices v_i and v_j, namely, the computation at vertex v_j cannot start before the completion of vertex v_i. For each edge (v_i, v_j), there is a cost for *interprocessor communication* (IPC) w_{v_i,v_j}, which is the time to transfer data from the processor that executes v_i to a different processor that will execute v_j. There is no IPC cost, i.e., $w_{v_i,v_j} = 0$, if vertices v_i and v_j are mapped to the same processor by the task scheduler. For a given datapath $< v_1 v_2 \cdots v_n >$, its *completion time* is the sum of the execution time at run-time, of each vertex, e_i, and all the IPC costs along the path. That is,

$$C(< v_1 v_2 \cdots v_n >) = e_1 + \sum_{i=2}^{n} (w_{v_{i-1},v_i} + e_i) \tag{2}$$

The *completion time* of the entire task graph G (or equivalently the given application), denoted by $C(G)$, is equal to the completion time of its *critical path*, which has the longest completion time among all its datapaths.

We are also given a *deadline* constraint $\mathcal{M}$, which specifies the maximum time allowed to complete the application. The application (or its task graph) will be executed periodically on a multiprocessor system with its deadline $\mathcal{M}$ as the period. We say that an iteration is *successfully completed* if its completion time, which depends on the run-time behavior, $C(G) \leq \mathcal{M}$. Closely related to $\mathcal{M}$ is a real-valued *completion ratio* constraint(or requirement) $\mathcal{Q}_0 \in [0, 1]$, which gives the minimum acceptable completion ratio over a sufficiently large number of iterations. Alternatively, $\mathcal{Q}_0$ can be interpreted as a guarantee on the probability with which an arbitrary iteration can be successfully completed.

3.2 Multiple Voltage System Model

We assume that there are multiple supply voltage levels available at the same time for each processor in the multiprocessor system. This type of system can be implemented by using a set of voltage regulators each of which regulates a specific voltage for a given clock frequency. In this way, the operating system can control the clock frequency at run-time by writing to a register in the system control state exactly the way as in Burd et al. [2000] except that the system does not need to wait for the voltage converter to generate the desired operating voltage. In sum, we can assume that each processor can switch its operating voltage from one level to another instantaneously and independently with the power dissipation $P \propto CV_{dd}^2 f$ and gate delay

$$d \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}}$$

at supply voltage V_{dd} and threshold voltage V_{th}, where $1 < \alpha \leq 2$ is a constant depends on the technology [Chandrakasan et al. 1992]. Furthermore, on a multiple voltage system, for a task under any time constraint, the voltage scheduling with, at most, two voltages, minimizes the energy consumption and the task is finished just at its deadline [Qu 2001].

The scheduling strategy (or a scheduler) for such multiprocessor multiple voltage system is a means of (1) assigning vertices to processors, (2) determining the execution order of vertices on each processor, and (3) selecting the supply voltage level for each processor.

3.3 Problem Statement

In this paper, we consider the following problem:

For a given task graph G with its deadline $\mathcal{M}$ and completion ratio constraint $\mathcal{Q}_0$, find a scheduling strategy for a multiprocessor multiple voltage system, such that the energy consumption to satisfy the completion ratio constraint $\mathcal{Q}_0$ is minimized.

It is well-known that the variable voltage task scheduling for low power is in general NP-hard [Hong et al. 1998; Qu 2001]. On the other hand, there exist intensive studies on multiprocessor task scheduling problem with other optimization objectives, such as completion time or IPC cost [McCreary et al. 1994; Sih and Lee 1993]. In this paper, we focus on developing on-line algorithms for voltage scaling (and voltage selection in particular) on a scheduled task graph. That is, we assume that tasks have already been assigned to processors and our goal is to determine *when* and *at which voltage* each task should be executed in order to minimize the total energy consumption while meeting the completion ratio constraint $\mathcal{Q}_0$.

4. ENERGY-DRIVEN VOLTAGE SCALING TECHNIQUES WITH COMPLETION RATIO CONSTRAINT

In this section, we first obtain, with a simple algorithm, the best completion ratio on a multiprocessor system for a given task assignment. We then give a lower bound on the energy consumption to achieve the best completion ratio. Our focus will be on the development of on-line energy reduction algorithms that leverage the required completion ratio, which is lower than the best achievable.

4.1 $\mathcal{Q}^{\max}$: the Highest Achievable Completion Ratio

Even when there is only one supply voltage, which results in a fixed processing speed, and each task has its own fixed execution time, the problem of determining whether a set of tasks can be scheduled on a multiprocessor system to be completed by a specific deadline remains NP complete (this is the *multiprocessor scheduling* problem [SS8], which is NP complete for two processors [Garey and Johnson 1979].). However, for a given task assignment, the highest possible completion ratio can be trivially achieved by simply applying the highest supply voltage on all the processors. That is, each processor keeps on executing whenever there exist tasks assigned to this processor ready for execution[2] and stops when it completes all its assigned tasks in the current iteration or when the deadline $\mathcal{M}$ is reached. In the latter, if any processor has not finished its execution, we say the current iteration is *failed*; otherwise, we have a *successful completion* or simply *completion*. Clearly this naïve method is a best-effort approach in that it tries to complete as many iterations as possible. Since it operates all the processors at the highest voltage, the naïve approach will provide the highest possible completion ratio, denoted by $\mathcal{Q}^{\max}$. In another words, if a completion ratio requirement cannot be achieved by this naïve approach within the given deadline $\mathcal{M}$, then no other algorithms can achieve it either.

When the application-specified completion ratio requirement $\mathcal{Q}_0 < \mathcal{Q}^{\max}$, a simple counting mechanism can be used to reduce energy consumption. Specifically, we cut the N iterations into smaller groups and shut the system down once sufficient iterations have been completed in each group. For example, if an MPEG application requires a 90% completion ratio, we can slow down the system (or switch the CPU to other applications) whenever the system has correctly decoded 90 out of 100 consecutive frames. This counting mechanism saves total energy by preventing the system from overserving the application.

For systems with multiple operating voltages, we mention that energy could have been saved over the above naïve approach in the following scenario:

[2]A task is ready for execution if all its predecessors have been completed. The completion means the completion of both processing at the predecessor and the IPC if the current task and its predecessor have been assigned to different processors.

- if we knew that an iteration would be completed earlier than the deadline $\mathcal{M}$, we could have processed with a lower voltage.
- if we knew that an iteration cannot be completed, we could have stopped the execution earlier.

To save the maximal amount of energy, we want to *determine the lowest voltage levels to lead us to completions right at the deadline* $\mathcal{M}$ and *find the earliest time to terminate an incompletable iteration*.

Clearly, additional information about the task's execution time is needed to answer these questions. Although it is not realistic to require the knowledge of the actual execution time before the real execution, it is possible to obtain each task's WCET, BCET, and the probabilistic distribution of its execution time. In the rest of this section, we propose on-line voltage scaling techniques to reduce energy with the help of such information.

4.2 BEEM: Achieving $Q^{\max}$ with the Minimum Energy

The best-effort energy minimization (BEEM) technique proposed by Hua et al. [2003] gives the minimum energy consumption on a single processor system to provide the highest achievable completion ratio. We propose algorithm BEEM1 to extend BEEM to multiprocessor systems. BEEM1 assumes the execution time of each vertex is known before its execution. Note that this is not the same as assuming the execution time of all the vertices are known before the start of an iteration. At run-time with the results of the partial execution, the execution time becomes more predictable for a single vertex. However, we also provide algorithm BEEM2 that only requires the best- and worst-case execution time of each vertex.

We define the *latest completion time* T_l^v and the *earliest completion time* T_e^v for a vertex v using the following recursive formulas:

$$T_e^v = T_l^v = \mathcal{M} \qquad \text{(if } v \text{ is a sink node)} \tag{3}$$
$$T_e^{v_i} = \min\{T_e^{v_j} - t_{j,k_j} - w_{v_i,v_j} | (v_i, v_j) \in E\} \tag{4}$$
$$T_l^{v_i} = \min\{T_l^{v_j} - t_{j,1} - w_{v_i,v_j} | (v_i, v_j) \in E\} \tag{5}$$

where $t_{j,1}$ and t_{j,k_j} are the BCET and WCET of vertex v_j, w_{v_i,v_j} is the cost of IPC from vertices v_i to v_j, which is 0 if the two vertices are assigned to the same processor.

LEMMA 1. *If an algorithm minimizes energy consumption, then vertex v_i's completion time cannot be earlier than $T_e^{v_i}$.*

PROOF. Clearly such algorithm will complete each iteration at deadline $\mathcal{M}$. Otherwise, one can always reduce the operating voltage and processing speed (or adjust the combination of two operating voltages) for the last task to save more energy.

Let t be vertex v_i's completion time at run-time. If $t < T_e^{v_i}$, for any path from v_i to a sink node v, $u_0 = v_i, u_1, \cdots, u_k = v$, let $WCET_{u_j}$ be the worst-case execution time of vertex u_j, then the completion time of this path will be

$$\begin{aligned}
T &\le t + \sum_{j=0}^{k-1} \left(w_{u_j,u_{j+1}} + WCET_{u_{j+1}}\right) < T_e^{v_i} + \sum_{j=0}^{k-1} \left(w_{u_j,u_{j+1}} + WCET_{u_{j+1}}\right) \\
&= T_e^{u_0} + w_{u_0,u_1} + WCET_{u_1} + \sum_{j=1}^{k-1} \left(w_{u_j,u_{j+1}} + WCET_{u_{j+1}}\right) \\
&\le T_e^{u_1} + \sum_{j=1}^{k-1} \left(w_{u_j,u_{j+1}} + WCET_{u_{j+1}}\right) \\
&\le \cdots \\
&\le T_e^{v} \\
&= \mathcal{M}
\end{aligned}$$

This implies that even when the WCET occurs for all the successor vertices of v_i on this path, the completion of this path occurs before the deadline $\mathcal{M}$. Note that this is true for all the path, therefore, the iteration finishes earlier and this cannot be the most energy efficient. Contradiction. □

LEMMA 2. *If vertex v_i's completion time $t > T_l^{v_i}$, then the current iteration is not completable by deadline $\mathcal{M}$.*

PROOF. Assuming that best-case execution time occur for all the rest vertices at time t when v_i is completed, this gives us the earliest time that we can complete the current iteration and there exists at least one path from v_i to one sink node v ($u_0 = v_i, u_1, \ldots, u_k = v$), and for each pair (u_j, u_{j+1}), $T_l^{u_j} = T_l^{u_{j+1}} - BCET_{u_{j+1}} - w_{u_j,u_{j+1}}$. The completion of this path occurs at time

$$\begin{aligned}
T &= t + \sum_{j=0}^{k-1} \left(w_{u_j,u_{j+1}} + BCET_{u_{j+1}}\right) \\
&> T_l^{v_i} + \sum_{j=0}^{k-1} \left(w_{u_j,u_{j+1}} + BCET_{u_{j+1}}\right) \\
&= T_l^{u_0} + w_{u_0,u_1} + BCET_{u_1} + \sum_{j=1}^{k-1} \left(w_{u_j,u_{j+1}} + BCET_{u_{j+1}}\right) \\
&= T_l^{u_1} + \sum_{j=1}^{k-1} \left(w_{u_j,u_{j+1}} + BCET_{u_{j+1}}\right) \\
&= \cdots \\
&= T_l^{v} \\
&= \mathcal{M}
\end{aligned}$$

□

The proposed on-line BEEM1 algorithm leverages these two lemmas to scale the voltage for the execution of each vertex, based on its execution time requirement. As we have stated earlier, we need to know only the execution time for the current vertex. Like the naïve approach, BEEM1 algorithm is capable of providing the highest possible completion ratio $Q^{\max}$ and it does that with the minimum energy consumption.

BEEM1: Let t be the current time that v is going to be processed and t_e^v be v's real execution time,

- if $t + t_e^v > T_l^v$, terminate the current iteration;
- if $t + t_e^v < T_e^v$, scale voltage such that v will be completed at T_e^v;
- otherwise, process at the highest voltage as in the naïve approach;

THEOREM 3. *If the execution time of each vertex becomes available at the start of its execution, then BEEM* 1 *algorithm achieves* $Q^{\max}$ *with the minimum energy consumption.*

PROOF. We first show that BEEM1 achieves $Q^{\max}$ as the naïve approach does. This is clear because (i) we terminate an iteration when $t + t_e^v > T_l^v$; Lemma 2 guarantees that this iteration cannot be completed; (ii) we slow down the execution when $t + t_e^v < T_e^v$, Lemma 1 shows that this delay will not make any completable iteration incompletable.

Now we show that BEEM1 is also the most energy efficient on-line scheduler by contradiction. We consider the completion time of a vertex at $t + t_e^v$, if this finish time is after T_l^v, energy will be wasted on incompletable iteration according to Lemma 2; if this finish time is before T_e^v, Lemma 1 states that it cannot be the most energy efficient; otherwise, if the finish time is in (T_e^v, T_l^v) and the processor does not operate at the highest speed, we can easily construct examples, similar to that in Lemma 1, to show that completable iterations will become incompletable. □

However, when it is unrealistic to know each task's real execution time (for instance, when the cost of execution time prediction is high), we propose the following on-line algorithm:

BEEM2: Let t be the current time that v is going to be processed,

- if $t + BCET_v > T_l^v$, terminate the current iteration;
- if $t + WCET_v < T_e^v$, scale voltage such that $WCET_v$ will be completed at T_e^v;
- otherwise, process at the highest voltage;

Without knowing task's real execution time, BEEM2 conservatively (i) terminates an iteration if it is incompletable even in vertex v's best-case execution time $BCET_v$; and (ii) slows down to save energy while still guaranteeing that vertex v's worst-case execution time $WCET_v$ can be completed at its earliest completion time T_e^v. From the fact that $BCET_v \leq t_e^v \leq WCET_v$, where t_e^v is the actual execution time, we can easily see

THEOREM 4. *BEEM2 algorithm achieves the highest completion ratio* $Q^{\max}$.

We mention that the pair $\{T_e^v, T_l^v\}$ can be computed offline only once and both BEEM1 and BEEM2 algorithms require, at most, two additions and two comparisons. Therefore, the on-line decision making takes constant time and

```
/* Step 1: Minimium effort for completion ratio guarantee. */
 1. find a topological order of the vertices: $v_1, \cdots, v_n$;
 2. $t_s^i = t_{i,k_i}$;                              /* assign WCET to each vertex */
 3. $\mathcal{Q} = 1$;          /* completion ratio must be 1 if each vertex gets its WCET */
 4. determine the completion time $L$;
 5. while ($\mathcal{Q} > \mathcal{Q}_0$)
 6. { for each vertex $v_j$ along critical paths;
 7.     { determine the completion time $L'$ when reduces $t_s^j$ from its current $t_{j,l}$ to $t_{j,l-1}$;
 8.       compute the completion ratio $\mathcal{Q}'_j = \mathcal{Q} \cdot \frac{P_{j,l-1}}{P_{j,l}}$;
 9.     }
10.     pick the vertex $v_j$ that achieves the maximum gain $(L - L') \cdot \frac{P_{j,l-1}}{P_{j,l}}$;
11.     $\mathcal{Q} = \mathcal{Q} \cdot \frac{P_{j,l-1}}{P_{j,l}}$;
12.     if ($\mathcal{Q} > \mathcal{Q}_0$)     $t_s^j = t_{j,l-1}$ ;
13. }
```

Fig. 1. QGEM's offline part to determine the minimum commitment to provide $\mathcal{Q}_0$.

will not increase the run time complexity. Finally, similar to our discussion for the naïve approach, further energy reduction is possible if the required completion ratio $\mathcal{Q}_0 < \mathcal{Q}^{\max}$.

4.3 QGEM: Completion Ratio Guaranteed Energy Minimization

Both the naïve approach and BEEM algorithms achieve the highest completion ratio. Although they can also be adopted to provide exactly the required completion ratio $\mathcal{Q}_0$ for energy reduction, they may not be the most energy efficient way to do so when $\mathcal{Q}_0 < \mathcal{Q}^{\max}$. In this section, we propose a hybrid offline online completion ratio $\mathcal{Q}$ guaranteed energy-minimization (QGEM) algorithm, which consists of three steps: (1) find the minimum effort to provide $\mathcal{Q}_0$; (2) static execution time allocation for each vertex; and (3) on-line voltage scaling.

In Step 1, we seek to find the minimum effort (that is, the least amount of computation t_s^i we have to process on each vertex v_i) to provide the required completion ratio $\mathcal{Q}_0$ (Figure 1). Starting with the full commitment to serve every task's WCET (*line 2*), we use a greedy heuristic to lower our commitment the vertices along critical paths (lines 6–13). Vertex v_j is selected first if the reduction from its WCET t_{j,k_j} to t_{j,k_j-1} (or from the current $t_{j,l}$ to $t_{j,l-1}$) maximally shortens the critical paths and minimally degrades the completion ratio, measuring by their product (line 10).

The goal in Step 2 is to allocate the maximum execution time t_q^i for each task v_i to process the minimum computation t_s^i and to have the completion time L close to deadline $\mathcal{M}$ (Figure 2). Lines 3–9 repetitively scale t_q^i for all the tasks. Because the IPCs are not scaled, maximally extending the allocated execution time to each task by a factor of $\mathcal{M}/L$ (line 6) may not stretch the completion time from L to $\mathcal{M}$. Furthermore, this unevenly extends each path and we reevaluate the completion time (and critical path) at line 7. To prevent an endless repetition, we stop when the scale factor r is less than a small number ϵ (line 5), which is set as 10^{-6} in our simulation. Lines 11–22 continue to scale t_q^i for vertices off critical paths in a similar way.

```
/* Step 2: Maximum execution time allocation with deadline constraint.*/
1. for each vertex $v_i$
2.     $done(v_i) = 0$;   $t_q^i = t_s^i$;            /* allocate time $t_s^i$ to each vertex */
3. determine the completion time $L$;
4. $r = \frac{\mathcal{M}}{L} - 1$;
5. while ( $r \geq \epsilon$ )                      /* to prevent an endless loop */
6. { $t_q^i = t_q^i \cdot (1 + r)$;          /* scale the time allocated to each vertex */
7.     determine the completion time $L$;
8.     $r = \frac{\mathcal{M}}{L} - 1$;
9. }
10. for each vertex $v_i$ on critical paths     $done(v_i) = 1$;
11. while ($done(v_i) = 0$ for some vertex $v_i$)
12. { determine the completion time $L$;
13.   while ($L < \mathcal{M}$)
14.     { for each vertex $v_i$ with $done(v_i) = 0$
15.         $t_q^i = t_q^i \cdot (1 + \delta)$;          /* $\delta$ is a small positive number */
16.       determine the completion time $L$;
17.     }          /* $L$ may exceed deadline $\mathcal{M}$, so we have to scale back $t_q^i$. */
18.   for each vertex $v_i$ with $done(v_i) = 0$
19.     { $t_q^i = t_q^i / (1 + \delta)$;
20.       if $v_i$ is on the critical path     $done(v_i) = 1$;
21.     }          /* it is still possible to scale vertices off critical paths. */
22. }
```

Fig. 2. QGEM's offline part to allocate execution time for each task.

```
/* Step 3: On-line voltage scheduling. */
1. $t$ = current time when vertex $v_i$ is ready for processing;
2. scale voltage such that the fixed workload $t_s^i$ can be completed by time $D_i$;
3. execute task $v_i$ to its completion;
4. if the completion occurs later than $D_i$
5.     report *failure*; break and wait for the next iteration;
```

Fig. 3. On-line scheduling policy for algorithm QGEM.

Now for vertex v_i, we have the pair (t_s^i, t_q^i) that represents the minimum amount of work and maximal execution time we have committed to v_i. We define the expected drop-time for v_i by the following recursive formula:

$$D_i = t_q^i + \max_{(v_k, v_i) \in E} \{D_k + w_{v_k, v_i} | (v_k, v_i) \in E\} \tag{6}$$

Step 3 defines the on-line voltage scheduling policy for the QGEM approach in Figure 3, where we scale voltage to complete a task v_i by its expected drop-time D_i assuming that the real-time execution time requirement equals to the minimum workload t_s^i we have committed to v_i (line 2). If v_i demands more, it will be finished after D_i, and we will drop the current iteration (line 4).

Note that if every task v_i has real execution time less than t_s^i in an iteration, QGEM's on-line scheduler will be able to complete this iteration. On the other hand, if longer execution time occurs at run-time, QGEM will terminate the

Table III. Characteristics of the Benchmark Task Graph[a]

Benchmark	n	m	k	$\mathcal{M}$	E
FFT1	28	2	15	1275	1040.4
FFT2	28	2	16	2445	2122.4
Laplace	16	2	13	2550	1799.7
karp10	21	2	12	993	592.8
TGFF1	39	2	20	4956	4437.7
TGFF2	51	3	36	4449	6102.8
TGFF3	60	3	51	5487	8541.2
TGFF4	74	2	49	9216	8838.9
TGFF5	84	3	74	6990	11137.6
TGFF6	91	2	59	11631	10799.3
TGFF7	107	3	89	9129	13608.3
TGFF8	117	3	111	9705	15674.0
TGFF9	131	2	85	15225	15165.7
TGFF10	147	4	163	10124	21925.8
TGFF11	163	3	159	13068	22984.4
TGFF12	174	4	169	12183	25220.2

[a] n, Number of vertices; m, number of processors; k, number of effective IPCs in the task graphs scheduled by DLS, that is, edges in the task graph whose two vertices are assigned to different processors; $\mathcal{M}$, deadline; E, average energy consumption per iteration, in the unit of the dissipation in one CPU unit at the reference voltage 3.3V, by the naïve scheduler to achieve $\mathcal{Q}_0 = 0.900$ with deadline constraint $\mathcal{M}$.

iteration right after the execution of this task. From the way we determine t_s^i (in Figure 1), we know that the required completion ratio $\mathcal{Q}_0$ will be guaranteed. Energy saving comes from two mechanisms: the early termination of *unnecessary* iterations (line 5 in Figure 3) and the use of low voltage to fully utilize the time from now to a task's expected drop-time (line 2 in Figure 3). We will confirm our claim on QGEM's completion ratio guarantee and demonstrate its energy efficiency by simulation in the next section.

5. SIMULATION RESULTS

In this section, we present the simulation results to verify the efficacy of our proposed approaches. We have implemented the proposed algorithms and simulated them over a variety of real-life and random benchmark graphs. Some task graphs, such as FFT (Fast Fourier Transform), Laplace (Laplace transform), and karp10 (Karplus-Strong music synthesis algorithm with 10 voices), are extracted from popular DSP applications. The others are generated by using *TGFF* [Dick et al. 1998], which is a randomized task graph generator. Table III gives the basic information about these benchmark task graphs. We assume that there are a set of homogeneous processors available. However, our approaches are general enough to be applied to embedded systems with heterogeneous multiprocessors.

Before we apply our approaches to the benchmark graphs, we need to schedule all tasks to available processors based on the performance, such as latency. Here we use the dynamic level scheduling (DLS) [Sih and Lee 1993] method to schedule the tasks; however, our techniques could be used with any alternative static scheduling strategy. The DLS method accounts for interprocessor communication overhead when mapping precedence graphs onto multiple processors in order to achieve the latency from the source to the sink as small as possible. We apply this method to the benchmarks and obtain the scheduling results, which include the task execution order in each processor and interprocessor communication links and costs. Furthermore, we assume that the interprocessor communication is full duplex and the intraprocessor data communication cost can be neglected.

After we obtain the results from DLS, we apply the proposed algorithms to them. There are several objectives for our experiments. First, we want to compare the energy consumption by using different algorithms under same deadline and completion ratio requirements. Second, we want to investigate the impact of completion ratio requirement and deadline requirement to the energy consumption of the proposed approaches. Finally, we want to study the energy efficiency of our algorithms with different numbers of processors.

We set up our experiments in the following way. For each task, there are three possible execution time, $e_0 < e_1 < e_2$, that occur at the following corresponding probabilities $p_0 \gg p_1 > p_2$, respectively. All processors support real-time voltage scheduling and power management (such as shut down) mechanism. Four different voltage levels, 3.3, 2.6, 1.9, and 1.2 V are available with threshold voltage 0.5 V. For each pair of deadline $\mathcal{M}$ and completion ratio $\mathcal{Q}_0$, we simulate 1,000,000 iterations for each benchmark by using each algorithm. Because naïve, BEEM1, and BEEM2 all provide the highest possible completion ratio that is higher than the required $\mathcal{Q}_0$, in order to reduce the energy, we take 100 iterations as a group and stop execution once $100\mathcal{Q}_0$ iterations in the same group have been completed.

Table IV reports the average energy consumption per iteration by different algorithms on each benchmark with deadline constraint $\mathcal{M}$ and completion ratio constraint $\mathcal{Q}_0(0.900)$. From the table, we can see that both BEEM1 and BEEM2 provide the same completion ratio with an average of nearly 29% and 26% energy saving over naïve. Compared with BEEM2, BEEM1 saves more energy because it assumes that the actual execution time can be known a priori. However, without this assumption the QGEM approach can still save more energy than BEEM2 in most benchmarks. Specifically, it provides 36% and 12% energy saving over naïve and BEEM2 and achieves 0.9111 average completion ratio which is higher than the required completion ratio 0.9000. It is mentioned that for FFT2 benchmark, QGEM has negative energy saving compared to BEEM2, because the deadline $\mathcal{M}$ is so long that BEEM2 can scale down the voltage to execute most of the tasks and save energy.

Figure 4 depicts the completion ratio requirement's impact to energy efficiency of different algorithms with same deadline $\mathcal{M}(9705)$. We can see that with the decrement of $\mathcal{Q}_0$, the energy consumption of each algorithm is decreased. However, different from naïve, BEEM1, and BEEM2, the energy consumption of QGEM does not change dramatically. Therefore, although under high completion ratio requirement ($\mathcal{Q}_0 > 0.75$ in Figure 4), using QGEM consumes the least energy, it may consume more energy than BEEM1, BEEM2, and even naïve when $\mathcal{Q}_0$ is low.

Table IV. Average Energy Consumption per Iteration by Naïve, BEEM1, BEEM2, and QGEM to Achieve $\mathcal{Q}_0 = 0.900$ with Deadline Constraints $\mathcal{M}$[a]

	Naïve	BEEM1	BEEM2	QGEM		
Benchmark	E	Saving vs. Naïve (%)	Saving vs. Naïve (%)	Saving vs. Naïve (%)	Saving vs. BEEM2 (%)	$\mathcal{Q}$
FFT1	1040.4	6.78	6.07	35.71	31.56	0.9118
FFT2	2122.4	18.15	18.15	15.96	−2.67	0.9104
Laplace	1799.7	42.75	32.63	45.12	18.53	0.9232
karp10	592.8	23.44	15.84	50.54	41.23	0.9392
TGFF1	4437.7	33.98	30.75	38.94	11.82	0.9090
TGFF2	6102.8	34.20	31.27	34.36	4.49	0.9185
TGFF3	8541.2	29.73	27.01	33.13	8.39	0.9034
TGFF4	8838.9	32.08	30.68	38.67	11.53	0.9109
TGFF5	11137.6	29.38	27.85	34.56	9.31	0.9065
TGFF6	10799.3	33.23	32.25	41.16	13.16	0.9057
TGFF7	13608.3	31.15	29.71	36.23	9.28	0.9027
TGFF8	15674.0	28.30	27.07	34.51	10.21	0.9074
TGFF9	15165.7	31.00	30.31	37.81	10.77	0.9084
TGFF10	21925.8	30.09	29.04	31.69	3.71	0.9029
TGFF11	22984.4	25.61	24.95	31.76	9.08	0.9100
TGFF12	25220.2	29.89	29.08	33.35	6.02	0.9074
Average		28.73	26.42	35.84	12.28	0.9111

[a] E, the baseline energy by the naïve scheduler from Table III; $\mathcal{Q}$, the actual completion ratio achieved by QGEM without forcing the processors stop at $\mathcal{Q}_0$.

The deadline requirement's impact to the energy consumption is shown in Figure 5, with the same $\mathcal{Q}_0$(0.900). Because the naïve approach operates at the highest voltage untill the required $\mathcal{Q}_0$ is reached, when the highest possible completion ratio of the system is close to 1, its energy consumption keeps constant, regardless of the change of the deadline $\mathcal{M}$. However, in BEEM1 and BEEM2, the latest completion time T_l^v and the earliest completion time T_e^v for each vertex v depend on $\mathcal{M}$ (see 3–5), and the energy consumption will be reduced dramatically with the increment of $\mathcal{M}$. For QGEM, the increment of deadline also has positive effect on the energy saving, while it is not as dramatic as it does to BEEM1 and BEEM2. Similar to the completion ratio requirement's impact, we conclude that QGEM consumes less energy than BEEM1 and BEEM2 in the short deadline (with the condition that $\mathcal{Q}_0$ is achievable), while consuming more energy when the deadline is long.

From Table IV and Figures 4 and 5, we can conclude that QGEM save more energy than BEEM1 and BEEM2 when $\mathcal{Q}_0$ is high and $\mathcal{M}$ is not too long. Actually this conclusion is valid regardless of the number of multiple processors. Figure 6 shows the energy consumption of different algorithms under different deadlines and different number of processors. With the increment of the number of processors, its latency will be reduced. Thus, for the same deadline(e.g., 7275), it is not relatively long and QGEM saves more energy than BEEM1 and BEEM2 for the system with small number of processors (e.g., four processors). However, for the system with large number of processors (e.g., $>$ five processors), QGEM will consume more energy than BEEM1 and BEEM2.

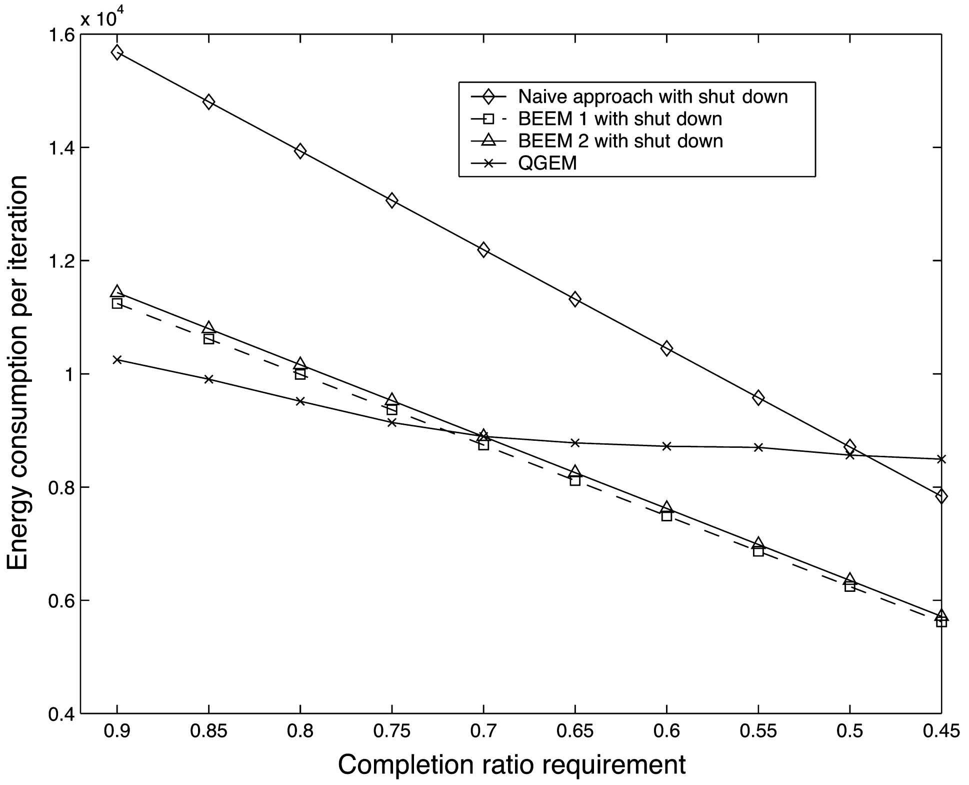

Fig. 4. Different completion ratio requirement's impact to the average energy consumption per iteration on benchmark TGFF8 with three processors.

6. CONCLUSIONS

Many embedded applications, such as multimedia and DSP applications, have a high performance requirement yet are able to tolerate certain level of execution failures. We investigate how to trade this tolerance for energy efficiency, another increasingly important concern in the implementation of embedded software. In particular, we consider systems with multiple supply voltages that enable dynamic voltage scaling, arguably the most effective energy reduction technique. We present several on-line scheduling algorithms that scale operating voltage based on some parameters pre-determined offline. All the algorithms have low run-time complexity yet achieve significant energy saving while providing the required performance, measured by the completion ratio.

The algorithms proposed in this paper have several limitations on which we are currently working:

- Leakage. As technology scales, leakage power dissipation is gaining more and more attention recently. The circuit's leakage power dissipation depends on operating voltage and threshold voltage among many other factors. One

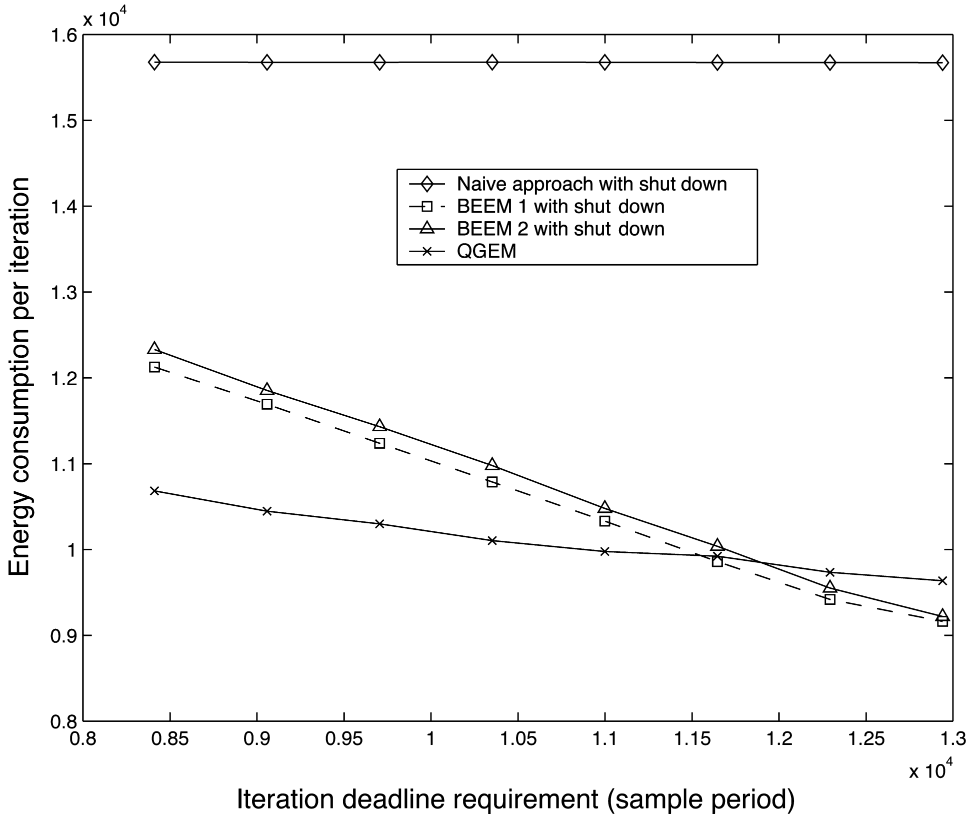

Fig. 5. Different deadline requirement's impact to the average energy consumption per iteration on benchmark TGFF8 with three processors.

can refer to, for example, Kim et al. [2003], for detailed discussion on leakage current and leakage power models. As we dynamically scale the operating voltage, the leakage will become affected and this will impact the on-line scheduling policy if we target the total (dynamic and leakage) energy minimization. This is currently under investigation.

- Voltage scaling overhead. In our discussion, the overhead (particular, the additional energy and time it takes for the circuit to reach the steady state at the new voltage level) is not considered directly. However, this overhead, once known, can be conveniently integrated into our approach. This is because that in our dynamic voltage scaling approach for multiple voltages, there is, at most, one voltage switch for each task [Qu 2001; Ishihara and Yasuura 1998]. One can compare the energy reduction with the overhead to decide whether the voltage should be scaled or not.
- Scheduling. We have mentioned that we are using the dynamic level scheduling method [Sih and Lee 1993] to schedule all the tasks on different processors first. Our proposed algorithms are then applied to reduce the energy consumption. Note that this scheduler is not driven by energy. It will be

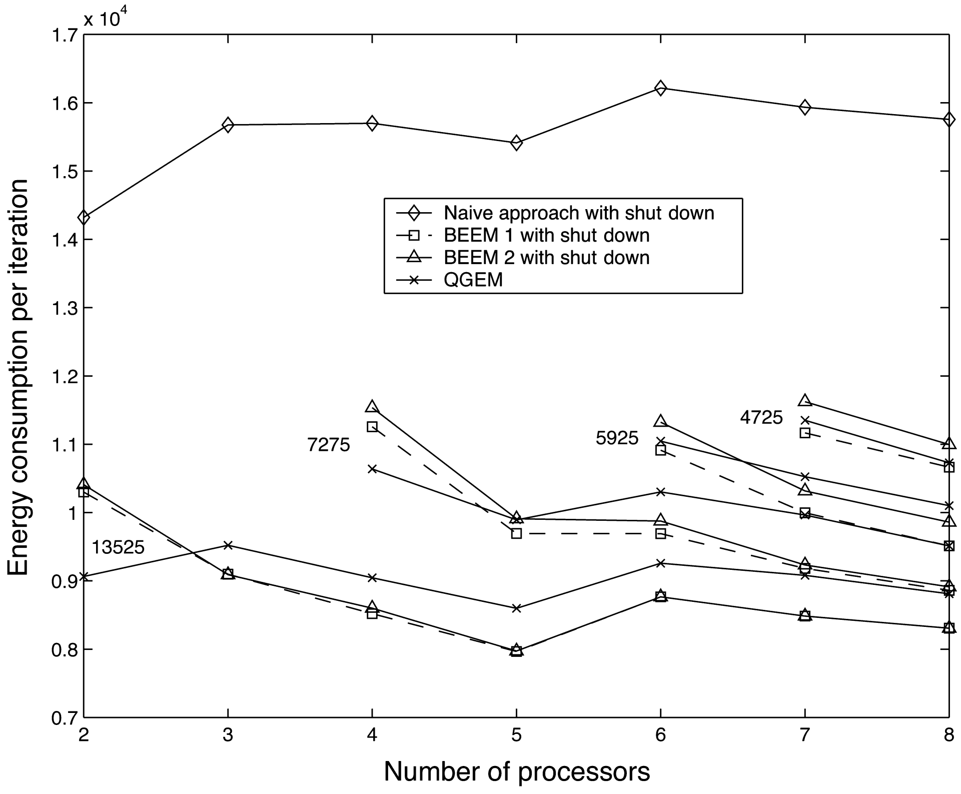

Fig. 6. The average energy consumption per iteration on benchmark TGFF8 with different number of processors and different deadlines (13525, 7275, 5925, and 4725).

interesting to study the impact of such scheduler to the system's total energy consumption.

REFERENCES

Bambha, N. K. and Bhattacharyya, S. S. 2000. A joint power/performance optimization technique for multiprocessor systems using a period graph construct. In *Proceedings of the International Symposium on System Synthesis*. 91–97.

Bolot, J. and Vega-Garcia, A. 1996. Control mechanisms for packet audio in the internet. In *Proceedings of IEEE Infocom*. 232–239.

Burd, T. D., Pering, T., Stratakos, A., and Broderson, R. 2000. A dynamic voltage scaled microprocessor system. *IEEE J. Solid-State Circuits 35*, 11 (Nov.), 1571–1580.

Chandrakasan, A. P., Sheng, S., and Broderson, R. W. 1992. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, *27*, 4, 473–484.

Dick, R. P., Rhodes, D. L., and Wolf, W. 1998. TGFF: Task graphs for free. In *Proc. Int. Workshop Hardware/Software Codesign*. 97–101.

Garey, M. R. and Johnson, D. S. 1979. *Computer and intractability: A guide to the theory of NP-Completeness.* Freeman, W. H., San Francisco, CA.

Grbic, A., Brown, S., and Caranci, S. et al. 1998. Design and implementation of the NUMAchine multiprocessor. *35th ACM/IEEE Design Automation Conference*. 65–69.

Gruian F. and Kuchcinski, K. 2001. LEneS: task scheduling for low-energy systems using variable supply voltage processors. In *Proc. of Asia and South Pacific Design Automation Conference*. 449–455.

Hong, I., Potkonjak, M., and Srivastava, M. B. 1998. On-line scheduling of hard real-time tasks on variable voltage processor. *IEEE/ACM International Conference on Computer-Aided Design*. 653–656.

Hua, S. and Qu, G. 2005a. Power minimization techniques on distributed real-time systems by global and local slack management. In *Proc. of Asia South Pacific Design Automation Conference*. 830–835.

Hua, S. and Qu, G. 2005b. Voltage setup problem for embedded systems with multiple voltages. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, *13*, 7, 869–872.

Hua, S., Qu, G., and Bhattacharyya, S. S. 2003. Energy reduction techniques for multimedia applications with tolerance to deadline misses. *40th ACM/IEEE Design Automation Conference*. 131–136.

Ishihara, T. and Yasuura, H. 1998. Voltage scheduling problem for dynamically variable voltage processors. *International Symposium on Low Power Electronics and Design*. 197–202.

Janka, R. S. and Wills, L. M. 2000. A novel codesign methodology for real-time embedded COTS multiprocessor-based signal processing systems. In *Proceedings of the International Workshop on Hardware/Software Co-Design*. 157–161.

Karkowski, I. and Corporaal, H. 1998. Design space exploration algorithm for heterogeneous multiprocessor embedded system design. *35th ACM/IEEE Design Automation Conference*. 82–87.

Karam, M. J. and Tobagi, F. A. 2001. Analysis of the delay and jitter of voice traffic over the internet. In *Proceedings of IEEE Infocom*. 824–833.

Kim, N., Austin, T., Blaauw, D., Mudge, T., Flautner, K., Hu, J., Irwin, M. J., Kandemir, M., and Narayanan, V. 2003. Leakage current: Moore's law meets static power. *Computer*, *36*, 12, 68–75.

Lee, S., Austin, T., Blaauw, D., and Mudge, T. 2004. Reducing pipeline energy demands with local DVS and dynamic retiming. *International Symposium on Low Power Electronics and Design (ISLPED)*. 319–324.

Luo, J. and Jha, N. K. 2000. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. *IEEE/ACM International Conf. on Computer-Aided Design*. 357–364.

Luo, J. and Jha, N. K. 2001. Battery-aware static scheduling for distributed real-time embedded systems. *38th ACM/IEEE Design Automation Conference*. 444–449.

Luo, J. and Jha, N. K. 2002. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *Proc. of Asia and South Pacific Design Automation Conference*. 719–726.

McCreary, C. L., Khan, A. A., Thompson, J. J., and McArdle, M. E. 1994. A comparison of heuristics for scheduling DAGs on multiprocessors. In *Proceedings of the International Parallel Processing Symposium*. 446–451.

Mishra, R., Rastogi, N., Zhu, D., Mosse, D. and Melhem, R. 2003. Energy aware scheduling for distributed real-time systems. *International Parallel and Distributed Processing Symposium*.

Qu, G. 2001. What is the limit of energy saving by dynamic voltage scaling?" *IEEE/ACM International Conference on Computer-Aided Design*. 560–563.

Scherrer, D. and Eberle, H. 1998. A scalable real-time signal processor for object-oriented data flow applications. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems*. 183–189.

Schmitz, M. T. and Al-Hashimi, B. M. 2001. Considering power variations of DVS processing elements for energy minimisation in distributed systems. *Proceedings of 14th International Symposium on System Synthesis*. 250–255.

Sih, G. C. and Lee, E. A. 1993. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Tran. on Parallel and Distributed Systems*, *4*, 2.

Sutton, R. A., Srini, V. P., and Rabey, J. M. 1998. A multiprocessor DSP system using PADDI-2. *35th ACM/IEEE Design Automation Conference*. 62–65.

Tia, T. S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.-C., and Liu, J. W. -S. 1995. Probabilistic performance guarantee for real-time tasks with varying computation times. *Proc. Real-Time Technology and Applications Symp.* 164–173.

YUAN, W. AND NAHRSTEDT, K. 2003. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. *19th ACM Symposium on Operating Systems Principles (SOSP'03)*. 149–163.

ZHU, D., MELHEM, R., AND CHILDERS, B. 2001. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE 22nd Real-Time Systems Symposium*. 84–94.

Received February 2005; accepted June 2005