Energy-Efficient Multi-processor Implementation of Embedded Software

Shaoxiong Hua, Gang Qu, and Shuvra S. Bhattacharyya

Electrical and Computer Engineering Department and Institute for Advanced Computer Studies University of Maryland, College Park, MD 20742, USA
{shua,gangqu,ssb}@eng.umd.edu

Abstract. This paper develops energy-driven completion ratio guaranteed scheduling techniques for the implementation of embedded software on multi-processor system with multiple supply voltages. We leverage application's performance requirements, uncertainties in execution time, and tolerance for reasonable execution failures to scale processors' supply voltages at run-time to reduce energy consumption. Specifically, we study how to trade the difference between the highest achievable completion ratio $\mathcal{Q}^{max}$ and the required completion ratio $\mathcal{Q}_0$ for energy saving. We propose several on-line scheduling policies, which are all capable of providing $\mathcal{Q}_0$, based on the knowledge about application's execution time. We show that significant energy saving is achievable when only the worst/best case execution time are known and further energy reduction is possible with the probabilistic distribution of execution time. The proposed algorithms have been implemented and their energy-efficiency have been verified by simulations over real-life DSP applications and the TGFF random benchmark suite.

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 over-designing the system (e.g., more hardware and more energy consumed than necessary). We discuss the problem of how to implement multi-processor 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 [2], while tolerance for losses in low bit-rate voice applications may be significantly lower [13]. Such tolerance gives rise to slacks that can be exploited when streamlining the embedded processing associated with such applications. Specifically, when

R. Alur and I. Lee (Eds.): EMSOFT 2003, LNCS 2855, pp. 257–273, 2003.

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 examined voltage scaling for multi-processor with known computation time and hard deadline constraints [1]. Luo and Jha presented a power-conscious algorithm [14] and static battery-aware scheduling algorithms for distributed real-time battery-powered systems [15]. Zhu et al. introduced the concept of slack sharing on multi-processor systems to reduce energy consumption [26]. 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 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 [24]. 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 [3,19]. 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 multiple voltage processors. Schmitz and Al-Hashimi 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 [21]. Gruian and Kuchcinski 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 [8]. Luo and Jha 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 [16]. In

[18], Mishra et al. 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. However, these algorithms use the slacks to reduce energy but they do not drop tasks to create more slacks. Different from them, some energy reduction techniques on single processor have been proposed by Hua et al. for multimedia applications with tolerance to deadline misses while providing a statistical completion ratio guarantee [10].

Finally, we mention that early efforts on multi-processor design range from the design space exploration algorithm [12] to the implementation of such systems [7,23]. And scalable architectures and co-design approaches have been developed for the design of multi-processor DSP systems (e.g., see [11,20]). These approaches, however, do not provide systematic techniques to handle voltage scaling, non-deterministic computation time, or completion ratio tolerance. Performance-driven static scheduling algorithms that allocate task graphs to multi-processors [22] 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.

A Motivational Example

We consider a simple case when a multiple-voltage processor executes three tasks $\mathcal{A}, \mathcal{B}, \mathcal{C}$ in that order repetitively. Table 1(a) gives each task's only two possible execution time and the probabilities that they occur. Table 1(b) shows the normalized power consumption and processing speed of the processor at three different voltages.

Table 1. Characteristics of the tasks and the processor. (a): each entry shows the best/worst case execution time at v_1 and the probability this execution time occurs at run time. (b): *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.

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

(a) Three tasks.

voltage	power	delay
$v_1 = 3.3V$	1	1
$v_2 = 2.4V$	0.30	1.8
$v_3 = 1.8V$	0.09	3.4

(b) Processor parameters.

Suppose that each iteration of "$\mathcal{A} \rightarrow \mathcal{B} \rightarrow \mathcal{C}$" must be completed in 10 CPU units and we can tolerate 40% of the 10,000 iterations to miss their deadlines. 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) Assign deadline pairs (0,6), (5,8), and (10,10) to $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ respectively. For each task, terminate the current iteration if the task cannot be completed by its second and longer deadline at v_1; otherwise, run at the lowest voltage without violating its first and shorter deadline or run at v_1 to its completion.

(III) In each iteration, assign 1, 7, and 2 (a total of 10) CPU units to $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ respectively. Each task can only be executed within its assigned slot: if it cannot be finished at v_1, terminate; otherwise run at the lowest voltage to completion.

Assuming that the execution time of each task follows the above probability, for each algorithm, we obtain the completion ratio $\mathcal{Q}$, each iteration's average processing time (at different voltages) and power consumption (Table 2). We mention that 1) algorithm I gives the highest possible completion ratio; 2) algorithm II achieves the same ratio with less energy consumption; and 3) algorithm III trades unnecessary completion for further energy reduction. Although algorithm I is a straightforward best-effort approach, the settings for algorithms II and III are not trivial: *Why the deadline pairs are determined for $\mathcal{A}$ and $\mathcal{B}$? Is it a coincidence that such setting achieves the same completion ratio as algorithm I? How to set execution slot for each task in algorithm III to guarantee the 60% completion ratio, in particular if we cannot find 80% and 75% whose product gives the desired completion ratio?*

Table 2. Expected completion ratio and energy consumption for the three algorithms. $t@v_1$, $t@v_2$, 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 $E \cdot 60\%/\mathcal{Q}$, corresponds to the case of shutting the system down once 6,000 iterations are completed.

	$\mathcal{Q}$	$t@v_1$	$t@v_2$	$t@v_3$	E	E@($\mathcal{Q} = 60\%$)
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

In this paper, i) we first formulate the energy minimization problem with deadline miss tolerance on multi-processor (DSP) systems; ii) we then develop on-line scheduling techniques to convert deadline miss tolerances into energy reduction via DVS; iii) this departs us from the conservative view of over-implementing the embedded software in order to meet deadlines under WCET; iv) our result is an algorithmic framework that integrates considerations of iterative multiprocessor scheduling, voltage scaling, non-deterministic computation time, and completion ratio requirement, and provides robust, energy-efficient multi-processor implementation of embedded software for embedded DSP applications.

2 Problem Formulation

We consider the *task graph* $G = (V, E)$ for a given application. Each vertex in the graph represents one computation and directed edges represent the data dependencies between vertices. For each vertex v_i, 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 $\{p_{i,1}, p_{i,2}, \cdots, p_{i,k_i} | \sum_{l=1}^{k_i} p_{i,l} = 1\}$ 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 then 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 to vertex v_i based on its WCET t_{i,k_i}.

A directed edge $(v_i, v_j) \in E$ shows that the computation at vertex v_j requires data from vertex v_i. For each edge (v_i, v_j), there is a cost for *inter-processor 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. 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* for 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 on a multi-processor system periodically 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.

Finally, we assume that there are multiple supply voltage levels available at the same time for each processor in the multi-processor 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 [3] 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 [4]. 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 [19].

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 multi-processor multi-voltage system (a means of (1) assigning vertices to processors, (2) determining the execution order of vertices on the same processor, and (3) selecting the supply voltage for each processor) 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 [9,19]. On the other hand, there exist intensive studies on multi-processor task scheduling problem with other optimization objectives such as completion time or IPC cost [17,22]. 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 $\mathcal{Q}_0$.

3 Energy-Driven Voltage Scaling Techniques with Completion Ratio Constraint

In this section, we first obtain, with a simple algorithm, the best completion ratio on multi-processor 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.

3.1 $\mathcal{Q}^{max}$: 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 multi-processor 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 [6].). 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;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 word, 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 over-serving the application.

For system with multiple operating voltages, we mention that energy could have been saved over the above naïve approach in the following scenario: i) if we knew that an iteration would be completed earlier than the deadline $\mathcal{M}$, we could have processed with a lower voltage; and ii) if we knew that an iteration cannot be completed and 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*. However, additional information about the task's execution time (e.g. WCET, BCET, and/or the probabilistic distribution) is required to answer these questions. In the rest of this section, we propose on-line voltage scaling techniques to reduce energy with the help of such information.

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

The best-effort energy minimization (BEEM) technique proposed by Hua et al. gives the minimum energy consumption on a single processor system to provide the highest achievable completion ratio [10]. We extend this approach and propose algorithm BEEM1 for the multi-processor system and BEEM2 that does not assume tasks' execution time are known as a priori.

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

$$T \leq t + \sum_{j=0}^{k-1}(w_{u_j,u_{j+1}} + WCET_{u_{j+1}}) < T_e^{v_i} + \sum_{j=0}^{k-1}(w_{u_j,u_{j+1}} + WCET_{u_{j+1}})$$
$$= T_e^{u_0} + w_{u_0,u_1} + WCET_{u_1} + \sum_{j=1}^{k-1}(w_{u_j,u_{j+1}} + WCET_{u_{j+1}})$$
$$\leq T_e^{u_1} + \sum_{j=1}^{k-1}(w_{u_j,u_{j+1}} + WCET_{u_{j+1}}) \leq \cdots \leq T_e^{v} = \mathcal{M}$$

This implies that even when the WCET happens 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 ($u_0 = v_i, u_1, \cdots, u_k = v$), 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 happens at time

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

∎

Combining these two lemmas and the naïve approach that achieves the highest possible completion ratio $\mathcal{Q}^{max}$, we have:

Theorem 3. (BEEM1) If we know the execution time t_e^v of vertex v, the following algorithm achieves $\mathcal{Q}^{max}$ with the minimum energy consumption.

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;

However, it is unrealistic to have each task's real execution time known as a priori, we hereby propose algorithm BEEM2, another version of BEEM that does not require tasks' real-time execution time to make decisions, yet still achieves the highest completion ratio $\mathcal{Q}^{max}$:

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. 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 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 $\mathcal{Q}_0 < \mathcal{Q}^{max}$.

3.3 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 on-line completion ratio $\mathcal{Q}$ guaranteed energy minimization (QGEM) algorithm, which consists of three steps:

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 $\mathcal{Q}_0$ (Fig. 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 $\mathcal{M}$ (Fig. 2). *Lines 3-9* repetitively scale t_q^i for all the tasks.

/* 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 $\mathcal{Q}_0$.

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 re-evaluate 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.

Now for vertex v_i, we have the pair (t_s^i, t_q^i) which represent the minimum amount of work and maximal execution time we commit to v_i. Define, recursively, the expected drop-time for v_i to be

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

Step 3 defines the on-line voltage scheduling policy for the QGEM approach in Fig. 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 iteration right after the execution of this task. From the way we determine t_s^i (in Fig. 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 Fig. 3) and the use of low voltage to fully utilize the time from now to a task's expected drop-time (*line 2* in Fig. 3). We will confirm our claim on QGEM's completion ratio guarantee and demonstrate its energy efficiency by simulation in the next section.

```
/* 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 = M/L - 1;
5. while ( r ≥ ε )                                      /* to prevent an endless loop */
6. { t_q^i = t_q^i · (1 + r);                           /* scale the time allocated to each vertex */
7.    determine the completion time L;
8.    r = 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 < M)
14.    { for each vertex v_i with done(v_i) = 0
15.        t_q^i = t_q^i · (1 + δ);                     /* δ is a small positive number */
16.      determine the completion time L;
17.    }                  /* L may exceed deadline 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 + δ);
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.

4 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* [5], which is a randomized task graph generator. 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 multi-processors.

Before we apply our approaches to the benchmark graphs, we need to schedule all of tasks to available processors based on the performance such as latency. Here

we use the dynamic level scheduling (DLS) [22] 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. Secondly, 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 number 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 $p_0 >> p_1 > p_2$ respectively. All processors support real time voltage scheduling and power management (such as shut down) mechanism. Four different voltage levels, 3.3V, 2.6V, 1.9V, and 1.2V are available with threshold voltage 0.5V. 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 3 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.

Fig. 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 doesn't change dramatically. Therefore, although under high completion

Table 3. Average energy consumption per iteration by naïve, BEEM1, BEEM2 and QGEM to achieve $\mathcal{Q}_0 = 0.900$ with deadline constraints $\mathcal{M}$. (n: number of vertices in the benchmark task graph; m: number of processors; $\mathcal{Q}$: the actual completion ratio achieved by QGEM without forcing the processors stop at $\mathcal{Q}_0$; energy is in the unit of the dissipation in one CPU unit at the reference voltage 3.3V.)

Benchmark	n	m	No. IPCs	$\mathcal{M}$	naïve	BEEM1	BEEM2	QGEM		
					energy	saving vs. naïve	saving vs. naïve	saving vs. naïve	saving vs. BEEM2	$\mathcal{Q}$
FFT1	28	2	15	1275	1040.4	6.78%	6.07%	35.71%	31.56%	0.9118
FFT2	28	2	16	2445	2122.4	18.15%	18.15%	15.96%	-2.67%	0.9104
Laplace	16	2	13	2550	1799.7	42.75%	32.63%	45.12%	18.53%	0.9232
karp10	21	2	12	993	592.8	23.44%	15.84%	50.54%	41.23%	0.9392
TGFF1	39	2	20	4956	4437.7	33.98%	30.75%	38.94%	11.82%	0.9090
TGFF2	51	3	36	4449	6102.8	34.20%	31.27%	34.36%	4.49%	0.9185
TGFF3	60	3	51	5487	8541.2	29.73%	27.01%	33.13%	8.39%	0.9034
TGFF4	74	2	49	9216	8838.9	32.08%	30.68%	38.67%	11.53%	0.9109
TGFF5	84	3	74	6990	11137.6	29.38%	27.85%	34.56%	9.31%	0.9065
TGFF6	91	2	59	11631	10799.3	33.23%	32.25%	41.16%	13.16%	0.9057
TGFF7	107	3	89	9129	13608.3	31.15%	29.71%	36.23%	9.28%	0.9027
TGFF8	117	3	111	9705	15674.0	28.30%	27.07%	34.51%	10.21%	0.9074
TGFF9	131	2	85	15225	15165.7	31.00%	30.31%	37.81%	10.77%	0.9084
TGFF10	147	4	163	10124	21925.8	30.09%	29.04%	31.69%	3.71%	0.9029
TGFF11	163	3	159	13068	22984.4	25.61%	24.95%	31.76%	9.08%	0.9100
TGFF12	174	4	169	12183	25220.2	29.89%	29.08%	33.35%	6.02%	0.9074
average						28.73%	26.42%	35.84%	12.28%	0.9111

ratio requirement ($\mathcal{Q}_0$ >0.75 in Fig. 4), using QGEM consumes the least energy, it may consume more energy than BEEM1, BEEM2 even naïve when $\mathcal{Q}_0$ is low.

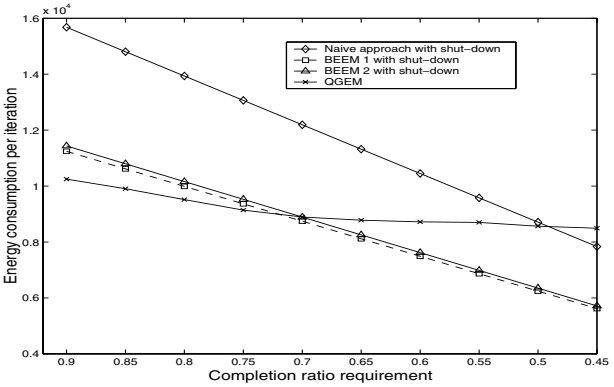

Fig. 4. Different completion ratio requirement's impact to the average energy consumption per iteration on benchmark TGFF8 with 3 processors.

The deadline requirement's impact to the energy consumption is shown in Fig. 5 with the same $\mathcal{Q}_0(0.900)$. Because the naïve approach operates at the highest voltage till 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.

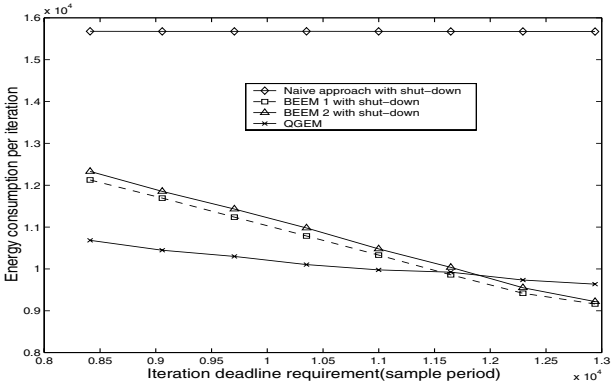

Fig. 5. Different deadline requirement's impact to the average energy consumption per iteration on benchmark TGFF8 with 3 processors.

From Table 3 and Fig. 4-5, 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. Fig. 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. So 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., 4 processors), however, for the system with large number of processors(e.g., >5 processors), QGEM will consume more energy than BEEM1 and BEEM2.

5 Conclusions

Many embedded applications, such as multimedia and DSP applications, have high performance requirement yet are able to tolerate certain level of execution

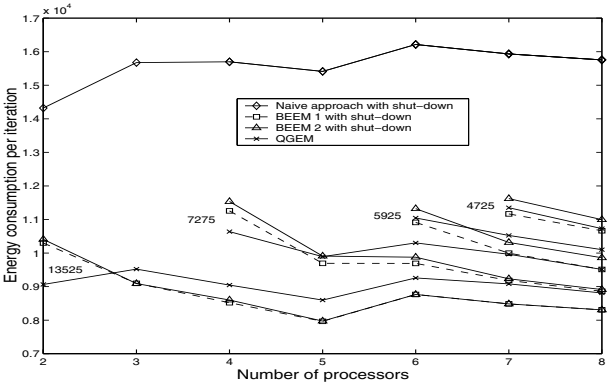

Fig. 6. The average energy consumption per iteration on benchmark TGFF8 with different number of processors and different deadlines(13525, 7275, 5925 and 4725).

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.

References

1. N. K. Bambha and S. S. Bhattacharyya. "A Joint Power/Performance Optimization Technique for Multiprocessor Systems Using a Period Graph Construct", *Proceedings of the International Symposium on System Synthesis*, pp. 91–97, 2000.
2. J. Bolot and A. Vega-Garcia. "Control Mechanisms for Packet Audio in the Internet", *Proceedings of IEEE Infocom*, pp. 232–239, March 1996.
3. T.D. Burd, T. Pering, A. Stratakos, and R. Brodersen. "A Dynamic Voltage Scaled Microprocessor System", *IEEE J. Solid-State Circuits*, Vol. 35, No.11, pp. 1571–1580, November 2000.
4. A.P.Chandrakasan, S.Sheng, and R.W.Broderson. "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473–484, 1992.
5. R. P. Dick, D. L. Rhodes, and W. Wolf. "TGFF: Task Graphs for Free", *Proc. Int. Workshop Hardware/Software Codesign*, pp. 97–101, Mar. 1998.
6. M.R.Garey and D.S.Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, NY, 1979.
7. A. Grbic, S. Brown, S. Caranci, et al. "Design and Implementation of the NUMAchine Multiprocessor", *35th ACM/IEEE Design Automation Conference*, pp. 65–69, June 1998.
8. F. Gruian and K. Kuchcinski. "LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors" *Proc. of Asia and South Pacific Design Automation Conference*, pp. 449–455, 2001

9. I. Hong, M. Potkonjak, and M.B. Srivastava. "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor", *IEEE/ACM International Conference on Computer Aided Design*, pp. 653–656, 1998.
10. S. Hua, G. Qu, and S. S. Bhattacharyya. "Energy Reduction Techniques for Multimedia Applications with Tolerance to Deadline Misses", *40th ACM/IEEE Design Automation Conference*, pp. 131–136, June 2003.
11. R. S. Janka and L. M. Wills. "A Novel Codesign Methodology for Real-Time Embedded COTS Multiprocessor-Based Signal Processing Systems", *Proceedings of the International Workshop on Hardware/Software Co-Design*, pp. 157–161, 2000.
12. I. Karkowski and H. Corporaal. "Design Space Exploration Algorithm For Heterogeneous Multi-processor Embedded System Design", *35th ACM/IEEE Design Automation Conference*, pp. 82–87, June 1998.
13. M. J. Karam and F. A. Tobagi. "Analysis of the Delay and Jitter of Voice Traffic Over the Internet", *Proceedings of IEEE Infocom*, pp. 824–833, April 2001.
14. J. Luo and N. K. Jha. "Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems", *IEEE/ACM International Conference on Computer-Aided Design*, pp. 357–364, November 2000.
15. J. Luo and N. K. Jha. "Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems", *38th ACM/IEEE Design Automation Conference*, pp. 444–449, June 2001.
16. J. Luo and N. K. Jha. "Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems", *Proc. of Asia and South Pacific Design Automation Conference*, pp. 719–726, Jan. 2002.
17. C.L. McCreary, A.A. Khan, J.J. Thompson, and M.E. McArdle. "A Comparison of Heuristics for Scheduling DAGs on Multiprocessors", *Proceedings of the International Parallel Processing Symposium*, pp. 446–451, April 1994.
18. R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem. "Energy Aware Scheduling for Distributed Real-Time Systems", *International Parallel and Distributed Processing Symposium*, Apr. 2003.
19. G. Qu. "What is the Limit of Energy Saving by Dynamic Voltage Scaling?" *IEEE/ACM International Conference on Computer-Aided Design*, pp. 560–563, November 2001.
20. D. Scherrer and H. Eberle. "A Scalable Real-time Signal Processor for Object-oriented Data Flow Applications", *Proceedings of the International Conference on Parallel and Distributed Computing Systems*, pp. 183–189, September 1998.
21. M. T. Schmitz and B. M. Al-Hashimi. "Considering Power Variations of DVS processing elements for energy minimisation in distributed systems", *Proceedings of 14th International Symposium on System Synthesis*, pp. 250–255, 2001.
22. G.C. Sih and E.A. Lee. "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", *IEEE Tran. on Parallel and Distributed Systems*, Vol. 4, No. 2, February 1993.
23. R.A. Sutton, V.P. Srini, and J.M. Rabey. "A Multiprocessor DSP System Using PADDI-2", *35th ACM/IEEE Design Automation Conference*, pp. 62–65, June 1998.
24. T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.W.-S. Liu. "Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times", *Proc. Real-Time Technology and Applications Symposium*, pp. 164–173, 1995
25. V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. "Reduced Power in High-Performance Microprocessors", *35th ACM/IEEE Design Automation Conference* pp. 732–737, June 1998

26. D. Zhu, R. Melhem and B. Childers. "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems", *IEEE 22nd Real-Time Systems Symposium* pp. 84–94, 2001