Probabilistic Design of Multimedia Embedded Systems

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

In this paper, we propose the novel concept of probabilistic design for multimedia embedded systems, which is motivated by the challenge of how to design, but not overdesign, such systems while systematically incorporating performance requirements of multimedia application, uncertainties in execution time, and tolerance for reasonable execution failures. Unlike most present techniques that are based on either worst- or average-case execution times of application tasks, where the former guarantees the completion of each execution, but often leads to overdesigned systems, and the latter fails to provide any completion guarantees, the proposed probabilistic design method takes advantage of unique features mentioned above of multimedia systems to relax the rigid hardware requirements for software implementation and avoid overdesigning the system. In essence, this relaxation expands the design space and we further develop an off-line on-line minimum effort algorithm for quick exploration of the enlarged design space at early design stages. This is the first step toward our goal of bridging the gap between real-time analysis and embedded software implementation for rapid and economic multimedia system design. It is our belief that the proposed method has great potential in reducing system resource while meeting performance requirements. The experimental results confirm this as we achieve significant saving in system's energy consumption to provide a statistical completion ratio guarantee (i.e., the expected number of completions over a large number of iterations is greater than a given value).

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Design, Performance

This work was performed during Shaoxiong Hua was in his Ph.D. study in the University of Maryland, College Park.
Authors' addresses: Shaoxiong Hua, Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043; email: huas@synopsys.com; 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.

Additional Key Words and Phrases: Energy minimization, soft real-time system, probabilistic design, completion ratio, multiple voltage, hardware/software codesign

ACM Reference Format:

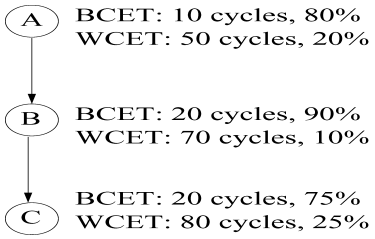
Hua, S., Qu, G., and Bhattacharyya, S. S. 2007. Probabilistic Design of Multimedia Embedded Systems. ACM Trans. Embedd. Comput. Syst. 6, 3, Article 15 (July 2007), 24 pages. DOI = 10.1145/1275986.1275987 http://doi.acm.org/ 10.1145/1275986.1275987

1. INTRODUCTION

With the advances in very large-scale integrated (VLSI) circuit and communication technology, multimedia embedded systems are widely used in many areas from business to education and entertainment to provide information service in applications, such as teleconferences, distant learning, movies, and video games. These systems require the processing of signal, image, and video data streams in a timely fashion to the end user's satisfaction. Such applications are often characterized by the repetitive processing on periodically arriving input data, such as voice samples or video frames, the moderate uncertainty in the execution time of each repetition, and, more important, the tolerance to occasional execution failure (or deadline misses with deadline implicitly determined by the throughput requirement of the input data streams) without being noticed by human visual and auditory systems. 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]. Finally, we mention that in many multimedia digital signal processing (DSP) applications, although the execution time of a task may vary dramatically because of various factors such as cache miss(es) and conditional branches, it is possible to obtain both the execution times in these events and the probabilities at which these events occur [Tia et al 1995; Hu et al. 2001; Kalavade and Moghe 1998; Qu et al. 2000].

Prior design space exploration methods for hardware–software codesign of embedded systems guarantee no deadline misses by considering the worst-case execution time (WCET) of each task [Eikerling et al. 1996; Henkel and Ernst 1998; Madsen et al. 1997]. As the multimedia systems can tolerate some violations of timing constraints, these methods will often lead to overdesigned systems that deliver performance higher than necessary, at the cost of more expensive hardware, higher energy consumption, and other system resources. Existing studies on the estimation of soft real-time system's probabilistic performance for applications with nondeterministic computation time mainly focus on improving system's performance or providing probabilistic performance guarantees [Tia et al 1995; Hu et al. 2001; Kalavade and Moghe 1998]. To the best of our knowledge, there is no reported effort on systematically incorporating application's performance requirements, uncertainties in execution time, and tolerance for reasonable execution failures to guide rapid and economic design of real-time embedded systems.

In this paper, we study the problem of how to integrate such tolerance to deadline misses into the design of multimedia systems. Specifically, we discuss how to implement a single application that needs to be executed repetitively, but



	A	B	C	RET	Prob
1	10	20	20	50	54.0%
2	50	20	20	90	13.5%
3	10	70	20	100	6.0%
4	10	20	80	110	18.0%
5	50	70	20	140	1.5%
6	50	20	80	150	4.5%
7	10	70	80	160	2.0%
8	50	70	80	200	0.5%

Fig. 1. An application with three sequential tasks and all the execution time scenarios.

can tolerate a certain number of execution failures. We propose the concept of probabilistic design for multimedia systems that relaxes the rigid hardware requirements for software implementation to meet the WCET and avoids overdesigning the multimedia systems. We consider the *design space* for a given application as a collection of hardware/software implementation alternatives. Each solution in the design space includes decisions on hardware/software partitioning, hardware configuration (processor selection, memory architecture, RTOS design, IO, and other peripherals, etc.), and software design and optimization. The relaxation of hardware requirements yields a larger design space. For instance, consider the application that consists of three sequential tasks A, B, and C, as shown in Figure 1, and has a deadline of 200 cycles. Each task has two possible execution times, best-case execution time (BCET) and WCET, that occurs with a given probability. All the eight possible execution time scenarios and the probability (Prob) that they occur are listed and sorted by the total real execution time (RET). When the application is repetitively executed, 200 cycles are required by the WCET-based design to guarantee the completion of each iteration. However, if the application only requires 60% of the iterations to be completed, then we can allocate 90 cycles and still meet the performance requirement statistically. This allows us to use a system/processor that is 55% slower and may lead to cost reduction. The proposed probabilistic design considers such slower design, while it is not included in the WCET-based design space.

The expansion of design space is not simply the inclusion of slower designs, it also poses the question of how to utilize these slower designs. For example, in Figure 1, we can let the application execute for 90 cycles and terminate the iteration if it does not finish in 90 cycles. This still guarantees us a 67.5% completion ratio. However, we can also do the following: first, allocate 10, 70, and 20 cycles to tasks A, B, and C, respectively; then we can execute each task in its assigned slot and terminate the iteration (not the individual task) if a task does not complete. In this way, although we will be able to complete the application only when execution time combination 1 or 3 (in Figure 1) happens, we still can provide the 60% completion ratio. This is actually a more efficient design, because it can terminate an iteration as early as the 10th cycle (when task A is not completed), while the former solution will always execute for 90 cycles.

With the introduction of probabilistic design and the enlarged design space, it becomes important to develop efficient design space exploration algorithms. For a given execution time distribution of each task and the tolerance to deadline misses (measured quantitatively by the expected completion ratio defined as the

expected number of completions over a large number of interations), we have developed a set of algorithms to estimate the probabilistic timing performance and to manage system resources in such a way that the system achieves the required completion ratio probabilistically with a reduced amount of system resources.

We will use the system's energy consumption, one of the most critical resources for multimedia embedded systems, as an example to demonstrate how our approach can lead to significant energy-efficient designs. Our basic idea is to exploit the tolerance to deadline misses to create slacks when streamlining the embedded processing associated with the soft real-time applications. More 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 the opportunity of dropping some tasks to introduce slacks during the application's execution for reduced cost or power consumption. Note that the slacks here are fundamentally different from the ones that have been exploited for power/latency minimization (see Section II for a brief survey). Previously, slack occurs in designs based on WCET when the real-time execution time is less than WCET and this is out of the designer's control. In our proposed probabilistic design method, we create slack *intentionally* (e.g., by dropping samples) in order to minimize the system's energy consumption. This becomes possible for multimedia applications that do not require the completion of each data sample (or each execution). Furthermore, information much richer than task's WCET can be made available for many DSP applications. Examples include the execution times in the best case, with cache miss(es), when interrupt occurs, when pipeline stalls, or when a different conditional branch happens. More important, the probabilities at which these events occur can be obtained by simulation or profiling [Tia et al 1995; Hu et al. 2001; Kalavade and Moghe 1998]. With such information, we can develop on-line and off-line schedulers for energy efficiency. However, identifying the best trade-off between performance and power is hard and requires intensive design space explorations [Hsieh et al 2000; Marculescu et al. 2001].

The rest of the paper is organized as follows: Section 2 describes the related work in design space exploration, performance analysis, and low-power design techniques. Section 3 gives the overview of our proposed probabilistic design space exploration methodology. This method consists of two key steps, i.e., the probabilistic timing performance estimation, which is discussed in Section 4; and the offline/on-line resource management of the probablistic performance guarantee, which is described in Section 5. Experimental results for our resource management techniques to reduce the energy consumption are given in Section 6. Finally, we conclude the paper in Section 7.

2. RELATED WORK

The most relevant work is on design space exploration and performance analysis, probabilistic performance estimation, and scheduling techniques for low power.

An integrated hardware–software codesign system should support design space exploration with optimization [Ernst 1998]. There are several works on

performance analysis for design space exploration based on monoprocessor architecture. In PMOSS [Eikerling et al. 1996], the authors presented a methodology for rapid analysis, synthesis, and optimization of embedded systems by providing modularity. Henkel and Ernst [1998] have presented high-level estimation techniques for the hardware effort and hardware/software communication time. They claimed that the proposed techniques are well suited for fast design space exploration. In the LYCOS system [Madsen et al. 1997], the authors used profiling techniques and evaluations of low-level execution time for hardware, software, and communication to estimate the system performance. For the rapid prototyping of hardware–software codesigns, Chatha and Vemuri [1998] introduced their performance evaluation tool to provide fast and accurate performance estimates based on profiling and scheduling. However, all of the above works specify the deadline as one of the design constraints that has to be met. We consider the probabilistic design for soft real-time embedded systems where deadline misses can be tolerated to some extent.

There are several papers on the probabilistic timing performance estimation for soft real-time systems design [Tia et al 1995; Hu et al. 2001; Kalavade and Moghe 1998]. The general assumption is that each task's execution time can be described by a discrete probability density function that can be obtained by applying path analysis and system utilization analysis techniques [Malik et al. 1997]. In Tia et al. [1995], the authors extended the scheduling algorithms and schedulability analysis methods developed for periodic tasks in order to provide probabilistic performance guarantee for semiperiodic tasks when the total maximum utilization of the tasks on each processor is larger than one. They described the transform-task method that transforms each semiperiodic task into periodic followed by a sporadic tasks. The method can provide an absolute guarantee for requests with shorter computation times and a probabilistic guarantee for longer requests. In Kalavade and Moghe [1998], a performance estimation tool that outputs the exact distribution of the processing delay of each application has been introduced. It can help the designers develop multimedia networked systems requiring soft real-time guarantees in a cost efficient manner. Given that the execution time of each task is a discrete random variable, Hu et al. [2001] proposed a state-based probability metric to evaluate the overall probabilistic timing performance of the entire task set. Their experimental results show that the proposed metric well reflects the timing behavior of systems with independent and/or dependent tasks. However, their evaluation method becomes very time consuming when the task has many different execution time values. We make similar assumptions as those in the existing work and use their tools and methods to evaluate the system's probabilistic performance.

Low-power consumption is one of the most important design objectives. Power is proportional to the square of the supply voltage. Therefore, reducing the supply voltage can result in great power saving. Dynamic voltage scaling (DVS), which varies the clock frequency and supply voltage according to the workload at runtime, can achieve the highest possible energy efficiency for time-varying computation load [Burd et al. 2000]. Scheduling techniques have been used at many design levels for the purpose of power reduction. Raje and Sarrafzadeh [1995] showed, at the behavioral level, how to apply multiple

voltages to function units in order to reduce power while satisfying the timing constraints. Johnson and Roy [1997] designed an algorithm that determines the optimal voltages along the datapaths. At the system level, task-scheduling techniques have been exploited to process real-time applications with hard deadlines. Hong et al. proposed heuristics for both off-line and on-line scheduling tasks on a variable-voltage processor system [Hong et al. 1998a, 1988b]. Shin and Choi [1999] used a fixed-priority scheduling method to achieve power reduction by exploiting slack times in real-time systems. Chen and Sarrafzadeh [1999] developed a provably good lower bound algorithm based on maximal-weighted-independent set for the power consumption of dual-supply voltage systems. Quan and Hu [2001] proposed an energy-efficient fixed-priority scheduling policy for real-time systems on variable-voltage processors.* Different from the above scheduling schemes, which are designed to meet application's deadline at all times, the energy-reduction techniques introduced in this paper, which can be applied in the off-line/on-line resource management phase in the probabilistic design methodology, reduce the system's average energy consumption while providing nonperfect completion ratio guarantees statistically. Some tasks are intentionally dropped according to the on-line scheduling algorithm to conserve energy.

Yuan and Nahrstedt [2003] have studied similar problems in mobile multimedia systems. In Yuan and Nahrstedt [2003], they developed an energy-efficient soft real-time CPU scheduler. The scheduler allocates CPU cycles periodically to tasks based on statistical demand, rather than worst case, to provide statistical performance guarantees. The earliest-deadline first scheduler is then applied in order to improve CPU utilization. A list of scaling points are determined for each task at which the task's execution will accelerate. They later extended this work to processors with discrete CPU speed options [Yuan and Nahrstedt 2004]. Their work is at application level, while we consider the data-flow graph representation of a single application that needs to be executed repetitively, but can tolerant a certain number of execution failures. This allows us to develop a simple online algorithm that can terminate an execution early to save energy without exceeding the application's tolerance to execution failures.

3. PROBABILISTIC DESIGN METHODOLOGY OVERVIEW

Many design methods have been developed based on WCET to meet the timing constraints without any deadline misses. These pessimistic methods are suitable for developing systems in a "hard real-time" environment, where any deadline miss will be catastrophic. Although they can take advantage of the fact that the actual execution time of each task frequently deviates from its WCET, sometimes by a large amount, these methods often lead to overdesigned systems, particular for "soft real-time" systems, such as multimedia systems, where occasional deadline misses can be tolerated. In order to avoid overdesigning the soft real-time systems, we propose the concept of "probabilistic design" to design systems that statistically meet the timing constraints of periodic

*Hua and Qu [2003] reported that two or three different voltages will be sufficient for energy efficiency for most application-specific embedded systems.

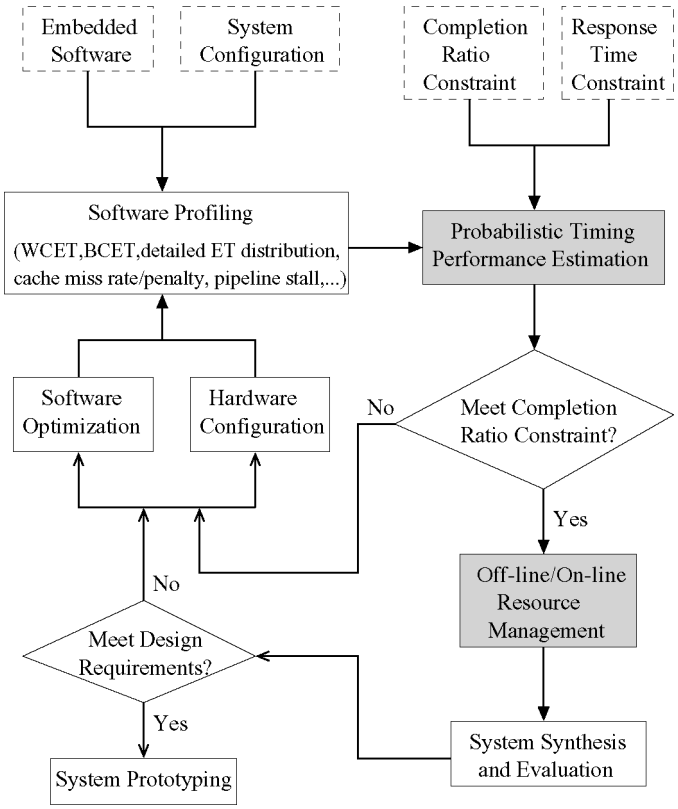


Fig. 2. Design flow in the probabilistic design methodology.

applications. That is, the systems may not guarantee the completion of each execution or iteration, but they will sufficiently produce many successful completions over a large number of iterations to meet the user-specific completion ratio. Or, even better, the probability that any execution will be completed is not lower than the desired completion ratio.

Clearly, the proposed "probabilistic design" will be preferred for many embedded systems, such as portable multimedia systems, where high portability, low-power consumption, and reasonably good performance are equally important. However, the corresponding "probabilistic design space" becomes larger than the traditional pessimistic design space, because it includes designs that fail some iterations while still meeting the desired completion ratio requirement statistically. This increases the design complexity and makes early design space exploration difficult. The "probabilistic design" will thrive only when designers can quickly explore the larger probabilistic design spaces.

Figure 2 depicts our probabilistic design space exploration approach for rapid and economic multimedia system design. This approach involves an integrated process of software profiling, probabilistic analysis, multiprocessor scheduling, DVS management, system- and processor-level modeling, simulation, and evaluation.

We start with the application or embedded software, the system's performance requirements (in terms of constraints such as timing and completion ratio), and a pool of target system architectures to select from. To reduce the design complexity, we divide the application into a set of tasks. Each task consists of tens, hundreds, or even thousands of instructions. These tasks and their data dependency form the popular data-flow graph that represents the application. Then, for each task, we use profiling tools and simulate on the current system configuration to collect the different execution times and the probabilities that they occur. Using this detailed execution information, we can estimate the system timing performance to check whether it is feasible for the current system configuration to achieve the desired performance. Because it is computational expensive to obtain the accurate system performance, our estimation at this phase gives only an upper bound of the system's best performance. If this estimation does not meet the requirement, we change the hardware configuration (for example, use a faster processor or a larger memory) and/or apply software-optimization techniques. Since any change on the target hardware configuration and/or software optimization may affect the application's actual execution, we have to repeat the software profiling process and reevaluate the system's probabilistic performance. This iterative design loop terminates when our estimation meets all the design requirements.

Now that the completion ratio constraint could be met, we move on to the next phase of off-line/on-line resource management to determine a practical way to do so. This is the key step in the proposed probabilistic design where we (1) allocate minimum system resources to each task offline in order to make the desired completion ratio probabilistically achievable, and (2) develop real-time schedulers to manage the resources at runtime such that the required completion ratio can be achieved probabilistically. Finally, we conduct system synthesis, simulation, and evaluation before prototyping the system.

We mention that this approach, like all other approaches based on application's execution time distribution analysis, depends heavily on the accuracy of such distribution. Therefore, it may not be applicable to systems performing multiple ad hoc applications. In the following two sections, we elaborate on the system's probabilistic performance estimation and the offline/on-line resource management (the two shaded blocks in Figure 2).

4. ESTIMATING THE PROBABILISTIC TIMING PERFORMANCE

In order to determine whether a given system implementation can meet the desired completion ratio constraint, we need to estimate the system's probabilistic timing performance. As we have mentioned earlier and will elaborate later in this section, it is computational expensive to obtain the system's accurate timing performance. Furthermore, it is also unnecessary to obtain the accurate estimation at early design stage, because it is going to change when the detailed system configuration is available. Therefore, we use an efficient method to calculate only an upper bound of the completion ratio that the system with current configuration can achieve. This helps us to explore the large probabilistic design space efficiently.

We consider the *task graph* $G = (V, E)$ for a given application, where V is the set of vertices in the graph that represent the application's computation tasks and E is the set of directed edges that represent the data dependencies between vertices. We assume that the execution time of each vertex is described by a discrete probability density function, a general assumption that has been adopted in the real-time system research community [Tia et al. 1995; Hu et al. 2001; Kalavade and Moghe 1998; Malik et al. 1997]. Specifically, for each vertex v_i, we associate a finite set of possible execution times $\{t_{i,1}, t_{i,2}, \ldots, t_{i,k_i}\}$ (under a reference system configuration) and the 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 times will occur at runtime. That is, with probability $p_{i,j}$, vertex v_i requires an execution time of $t_{i,j}$.

The *completion time* of the task graph G (or equivalently the given application) under a fixed execution order $<v_1 v_2 \cdots v_n>$, is the sum of each vertex's runtime execution time e_i: $C(<v_1 v_2 \cdots v_n>) = \sum_{i=1}^{n} e_i$. The *deadline* constraint $\mathcal{M}$ specifies the maximum time allowed to complete the application. The application (or its task graph) will be executed periodically and its period is greater than or equal to the deadline $\mathcal{M}$. We say that an iteration is *successfully completed* if $C(<v_1 v_2 \cdots v_n>) \leq \mathcal{M}$. The performance requirement is measured by a real-valued completion ratio $\mathcal{Q}_0 \in [0, 1]$, which is the minimum ratio of completions that the system has to maintain over a sufficiently large number of iterations. For the hard real-time system, $\mathcal{Q}_0 = 1$; and for the soft real-time system, $\mathcal{Q}_0 < 1$. Let K be the number of successfully completed iterations over a total of $N \gg 1$ iterations; thus, the actual achieved completion ratio is $\mathcal{Q} = \frac{K}{N}$. We say that the completion ratio constraint is satisfied over these N iterations if $\mathcal{Q} \geq \mathcal{Q}_0$.

Suppose that $t_{1,j_1}, t_{2,j_2}, \ldots, t_{n,j_n}$ are the actual execution times of the n vertices, respectively, in one iteration. The probability that this occurs is $\prod_{i=1}^{n} p_{i,j_i}$, where p_{i,j_i} is the probability that t_{i,j_i} happens. Recall that these are under the reference system configuration. For a different system configuration, let t'_{i,j_i} be the time to execute task v_i that requires an execution time t_{i,j_i} under the reference configuration. When the completion time $\sum_{i=1}^{n} t'_{i,j_i}$ is no later than the deadline $\mathcal{M}$, we will have a completion on this new system. Therefore, we have

THEOREM 0. *The maximum achievable completion ratio is given by:*

$$\mathcal{Q}^{\max} = \sum_{\sum_{i=1}^{n} t'_{i,j_i} \leq \mathcal{M}} \prod_{i=1}^{n} p_{i,j_i} \tag{1}$$

where the sum is taken over the execution time combinations that meet the deadline constraint $\mathcal{M}$ and the product computes the probability that each combination occurs.

This is similar to the state-based feasibility probability defined in Hu et al. [2001]. $\mathcal{Q}^{\max}$ helps us to quickly explore the probabilistic design space.

We mention that task graph is a popular and powerful model to represent applications on multiprocessor systems. For a uniprocessor system, it is equivalent to consider the task graph as a sequence of tasks to be performed, where the tasks are ordered in such a way (e.g., any topological order of the corresponding task graph) that if one task depends on the completion of other tasks' execution, then this task is scheduled after those tasks.

Specifically, if $\mathcal{Q}^{\max} < \mathcal{Q}_0$, which means that the completion ratio requirement is not achievable under current system configuration, then we can make the early decision to reconfigure the hardware or optimize the software implementation. More importantly, this decision is the correct decision. From Eq. (1), we see that to achieve $\mathcal{Q}_0$, one has to either increase the cases where the execution time combination meets the deadline (the number of terms to be summed up) or the probability that such combination occurs (the value of the product term). Therefore, we need to make the system faster and/or to find a better implementation of the software that requires less execution time.

The drawback of this estimation is that Eq. (1) is computationally expensive, particularly when there are many tasks and each task has multiple execution times. For example, there will be 3^{50} different execution time combinations on a task graph of 50 vertices, where each vertex has only the best-, average-, and worst-case execution time. Because of the importance of determining whether the required $\mathcal{Q}_0$ is achievable in designing fast probabilistic design space exploration techniques, we have developed the following polynomial heuristic.

Assuming that the task's execution times under the reference configuration are ordered such that $t_{i,1} < t_{i,2} < \cdots < t_{i,k_i}$, we define the prefix sum of the occurrence probability

$$P_{i,l_i} = \sum_{j=1}^{l_i} p_{i,j} \tag{2}$$

which measures the probability that the computation at vertex v_i is not longer than t_{i,l_i}. If we allocate time t_{i,l_i} to task v_i and drop the iteration if its actual execution time is longer, then we achieve a completion ratio

$$\mathcal{Q} = \prod_{i=1}^{n} P_{i,l_i} = \prod_{i=1}^{n} \sum_{j=1}^{l_i} p_{i,j} \tag{3}$$

We use a greedy approach to estimate whether completion ratio $\mathcal{Q}_0$ can be achieved within the deadline $\mathcal{M}$ (see the off-line part of the O^{2}ME algorithm in Figure 4). First, we assign each vertex its WCET. This yields $\mathcal{Q} = 1$ and meet any completion ratio requirement $\mathcal{Q}_0$. If the total assigned completion time $\sum_{i=1}^{n} t_{i,k_i}$ exceeds the deadline constraint, we check whether $\mathcal{Q}_0$ can be met as follows. From Eq. (3), we know that if we cut the time slot of vertex v_i from t_{i,l_i} to $t_{i,(l_i-1)}$, the completion ratio will be reduced by the factor of $\frac{P_{i,(l_i-1)}}{P_{i,l_i}}$ and the total assigned time will be reduced by $t_{i,l_i} - t_{i,(l_i-1)}$. We iteratively cut the time slot of vertex v_j that yields the largest $(t_{j,l_j} - t_{j,(l_j-1)})\frac{P_{j,(l_j-1)}}{P_{j,l_j}}$, as long as it gives a completion ratio larger than $\mathcal{Q}_0$. This greedy-selection approach frees more assigned time slot at the minimum level of completion ratio reduction. When we cannot reduce the completion ratio any further and the total assigned time $\sum_{i=1}^{n} t_{i,l_i}$ is not larger than the deadline $\mathcal{M}$, our heuristic will conclude that the required Q_0 is achievable. Otherwise, it will report that Q_0 cannot be guaranteed, even though, in some cases, $\mathcal{Q}^{\max}$ is actually larger than $\mathcal{Q}_0$.

The complexity of the proposed heuristic is roughly $O((\sum_{i=1}^{n} k_i) \log \sum_{i=1}^{n} k_i)$, which we will explain in detail in the next section in Figure 4. In contrast, computing the exact value of $\mathcal{Q}^{\max}$ by Theorem 0 has an exponential complexity

$O(n\prod_{i=1}^{n} k_i)$ (note that the factor n comes from the $\sum_{i=1}^{n} t_{i,l_i}$ and the $\prod_{i=1}^{n} p_{i,j_i}$ operations in Eq. (1).

5. MANAGING SYSTEM RESOURCES UNDER PROBABILISTIC PERFORMANCE CONSTRAINT

When $\mathcal{Q}^{\max} \geq \mathcal{Q}_0$, it becomes theoretically possible to deliver the probabilistic performance guarantee (in terms of expected completion ratio) with the current system configuration. The resource management phase in our design space exploration method aims to reduce the design cost while providing a practical way to statistically achieve $\mathcal{Q}_0$. It includes: (1) determining the minimum system resource required to provide the probabilistic performance guarantee; and (2) developing on-line scheduling algorithms to guide the system to achieve such guarantee at runtime with the determined minimum resource.

As energy consumption has emerged as one of the most important concerns in the design of embedded systems, particularly for the battery-operated portable systems, we consider, in this section, energy as the resource to manage and present our newly developed off-line/on-line energy reduction techniques with completion ratio guarantee. We achieve the energy saving by the dynamic voltage scaling method on multiple supply voltage and multiple threshold voltage system, which has been identified by the international technology roadmap for semiconductors (ITRS) as the trend of future systems. Specifically, we consider the following problem:

> *For a given task graph, its deadline, its expected completion ratio constraint, and the task execution time distribution, find a scheduling strategy for a multiple voltage system such that the resource (e.g., energy) consumed to statistically satisfy the completion ratio constraint is minimized.*

The solution to this problem is a scheduling strategy that consists of determining the execution order of vertices in the given task graph and selecting the supply voltage for the execution of each vertex.

5.1 Dynamic Voltage Scaling Systems

Although leakage-power dissipation has recently gained more and more attention, dynamic power still dominates in most embedded systems. Dynamic power in a CMOS circuit is proportional to $\alpha C_L V_{dd}^2 f_{clock}$, where αC_L is the effective switched capacitance, V_{dd} is the supply voltage, and f_{clock} is the clock frequency. Reducing the supply voltage can result in substantial power and energy saving. Roughly speaking, system's power dissipation is halved if we reduce V_{dd} by 30% without changing any other system parameters. However, this saving comes at the cost of reduced throughput, slower system clock frequency, or higher gate delay. The gate delay is proportional to $\frac{V_{dd}}{(V_{dd}-V_t)^\beta}$, where V_t is the threshold voltage and $\beta \in (1.0, 2.0]$ is a technology-dependent constant. Dynamic voltage scaling is a technique that varies system's operating voltage and clock frequency on-the-fly, based on the computation load, to provide desired performance with the minimum energy consumption. It has been

demonstrated as one of the most effective low-power system design techniques and has been supported by many modern microprocessors. Examples include Transmeta's Crusoe, AMD's K-6, Intel's XScale, and Pentium III and IV, and some DSPs developed in Bell Labs.

The latest ITRS predicts that "power dissipation for high performance microprocessors will exceed the package limits by 25 times in 15 years" and, as a result, future systems will feature "multiple V_{dd} and multiple V_t being used on a single chip to reduce power while maintaining performance." Low-power design is of particular interest for the soft real-time multimedia embedded systems and we assume that our systems have multiple voltages available at the same time. Such systems can be implemented by using a set of voltage regulators. Each regulator provides certain stable supply voltage. The processor can switch from one supply voltage to another at run time. To consider the energy and delay overhead associated with multiple-voltage systems, note that we use no more than two different voltages during the execution of each vertex [Ishihara and Yasuura 1998] and we can easily compare the energy saving from multiple voltages and the energy/delay overhead they introduce to decide whether multiple voltages should be used on this vertex. Therefore, we assume that voltage scaling occurs simultaneously without any energy and delay overhead.

5.2 Energy Reduction Techniques with Completion Ratio Guarantee

In order to reduce the energy consumption, we need to take advantage of the information we have obtained, such as the execution time distribution of each task, the deadline $\mathcal{M}$, and the completion ratio constraint $\mathcal{Q}_0$. In this section, we first introduce a naïve approach that achieves the highest possible completion ratio $\mathcal{Q}^{\max}$. This method does not consider energy efficiency and we will use the energy consumption by this method as the baseline to show the energy reduction by the other two proposed approaches. The on-line best-effort energy minimization (BEEM) approach maintains the same highest completion ratio with the provably minimum energy consumption. The off-line and on-line minimum-effort approach(O^2ME) further reduces the system energy consumption by intentionally dropping some completable tasks to avoid achieving a completion ratio higher than required. In both the BEEM and the O^2ME approaches, we assume that the actual execution time of each task in each iteration can be known before the execution of that task, which can be predicted on-the-fly by applying some methodologies, such as those mentioned in Yang et al. [2002]. Note that this is different from the unrealistic assumption of knowing each individual task's execution time before the iteration starts.

5.2.1 *A Naïve Best-Effort Approach.* In this approach, the processor will keep on executing tasks, at the highest voltage and, hence, the highest speed, to the completion of the current iteration or when deadline $\mathcal{M}$ is reached. In the latter, if there is any task remains unfinished, we say the current iteration is *failed*; otherwise, we have a *successful completion* or simply *completion*. For any iteration with execution time combination that contributes to $\mathcal{Q}^{\max}$ in Eq. (1), we will have a completion. Clearly, this approach gives the maximum

achievable completion ratio, but it does not take energy consumption into the consideration. We thus refer it as the naïve best-effort approach.

Now we compute the energy consumption by this naïve best-effort approach. Since it will not drop any iteration until the deadline $\mathcal{M}$, the execution time of each failed iteration will be $\mathcal{M}$ and the energy consumption will be $P_{ref}\mathcal{M}$, where P_{ref} is the power dissipation at the reference voltage. For a completion with completion time $\mathcal{C}$, the energy consumed will be $P_{ref}\,\mathcal{C}$. If the naïve best-effort approach completes K out of N iterations and $\mathcal{C}_i$ is the completion time of the ith completion, then the energy consumption on these N iterations is:

$$E = P_{ref} \cdot \left(\sum_{i=1}^{K} \mathcal{C}_i + (N-K)\mathcal{M} \right) \tag{4}$$

This naïve approach achieves the maximum completion ratio $\mathcal{Q}^{\max}$. We can use the value $E\frac{\mathcal{Q}_0}{\mathcal{Q}^{\max}}$ to estimate the energy consumption for providing the given expected completion ratio $\mathcal{Q}_0$. The only reason that we consider this approach is to use this value as a baseline to evaluate the following two energy-efficient approaches.

5.2.2 *On-line Best-Effort Energy Minimization.* With the flexibility of switching voltages at runtime, the system can maintain the same completion ratio at reduced energy consumption. We observe that we could save energy over the above naïve approach in the following two occasions: first, if a completion occurs earlier than the deadline $\mathcal{M}$, we could have operated the processor at a lower voltage; second, if an iteration eventually fails at $\mathcal{M}$, we could have terminated it earlier. Apparently, we want to slow down the processing speed as much as possible and terminate the uncompletable iteration as early as possible to save energy. We now discuss how to achieve the maximum energy saving without sacrificing the completion ratio provided by the naïve approach.

For a given task graph with execution order (e.g., any topological order that guarantees that whenever a task v depends on another task u, task u will be in front of v in the execution order) $v_1, v_2, \ldots, v_n$, we define vertex v_i's *latest completion time* T_{l_i} and *earliest completion time* T_{e_i} recursively (the subscripts l and e stand for *latest* and *earliest*, respectively):

$$T_{l_n} = T_{e_n} = \mathcal{M} \tag{5}$$

$$T_{l_i} = T_{l_{i+1}} - t_{i+1,1} \tag{6}$$

$$T_{e_i} = T_{e_{i+1}} - t_{i+1,k_{i+1}} \tag{7}$$

where $t_{i+1,1}$ and $t_{i+1,k_{i+1}}$ are the BCET and WCET of vertex v_{i+1} at the reference voltage.

T_{l_i} represents the latest time, or a *hard deadline* to complete vertex v_i in order to guarantee a completion with a nonzero probability. That is, if its completion time $t > T_{l_i}$, then the current iteration can never be completed before the deadline $\mathcal{M}$. To see this, let us consider the best case when all the successors

1. current time $t = 0$;
2. for each vertex v_i $(i = 1, 2, \cdots, n)$
3. $e_{i,j,ref}$ = the actual execution time of v_i at the reference voltage in jth iteration;
4. if $(t + e_{i,j,ref} > T_{l_i})$
5. terminate the current iteration;
6. if $(t + e_{i,j,ref} < T_{e_i})$
7. scale voltage so v_i is finished at $t = T_{e_i}$;
8. else
9. run v_i at the reference voltage to finish v_i at $t = t + e_{i,j,ref}$;

Fig. 3. Best-effort energy-minimization (BEEM) scheduling algorithm.

of v_i have their BCET. The completion time in this case will be:

$$\begin{aligned} t + \sum_{q=i+1}^{n} t_{q,1} &> T_{l_i} + t_{i+1,1} + \sum_{q=i+2}^{n} t_{q,1} \\ &= T_{l_{i+1}} + t_{i+2,1} + \sum_{q=i+3}^{n} t_{q,1} \\ &= \cdots \\ &= T_{l_{n-1}} + t_{n,1} \\ &= T_{l_n} = \mathcal{M} \end{aligned}$$

Similarly, we can show that any completion time $t < T_{e_i}$ will result in a completion earlier than deadline $\mathcal{M}$, which is not energy-efficient. The operating voltage for v_i can be reduced as long as v_i's completion is not after T_{e_i}, the *soft deadline*.

Figure 3 depicts the proposed on-line BEEM scheduling algorithm after precomputing the soft/hard deadline pair $\{T_{e_i}, T_{l_i}\}$ for each vertex. The scheduler scales the operating voltage at runtime when the execution time of each vertex is available in order to save energy with a guaranteed best completion ratio. More specifically, we have

THEOREM 1. *BEEM guarantees the highest completion ratio with the minimum energy consumption.*

PROOF. We first show that BEEM achieves the same completion ratio as the naïve approach. To see this, notice that (1) when we terminate the execution at Step 5 in Figure 3, we already know that there is no chance to have a successful completion from the definition of T_{l_i}; (2) when we reduce voltage at Step 7 to finish v_i at T_{e_i}, we are guaranteed a completion from the way we define T_{e_i}. Therefore, the naïve approach cannot do any better; and (3) at Step 9, BEEM and the naïve approach are identical.

Next we show that BEEM is the most energy-efficient on-line scheduler by contradiction. First, BEEM detects a failed iteration at the earliest time at Step 4 and thus keeps the "wasted" energy at the minimum level. Therefore, no other deterministic scheduler can consume less energy than BEEM on failed iterations. Second, we argue that any scheduler that is more energy-efficient than BEEM cannot maintain the highest completion ratio. If a more

energy-efficient on-line scheduler exists, it will fall behind the execution of BEEM at some time t. Now consider an iteration constructed in the following way: the execution times of the tasks before time t are identical to the current iteration (so the new scheduler falls behind), the execution times of the rest tasks are set up in a way that BEEM needs to run at the highest voltage to complete the iteration exactly at the deadline. Note that the new scheduler is already behind at time t and even if it runs at the highest voltage for the rest of the time, it cannot catch up BEEM will and eventually fail this iteration, which is completable by BEEM. □

5.2.3 *Off-line/On-line Minimum-Effort Approach.* Both the naïve approach and BEEM achieve the best possible completion ratio $\mathcal{Q}^{\max}$. They will be energy inefficient when the required completion ratio is lower than $\mathcal{Q}^{\max}$. The proposed offline/on-line minimum effort (O^2ME) algorithm leverages the available task execution time information beyond BCET and WCET to further reduce energy consumption. It consists of an off-line execution time allocation phase, where each vertex v_i is assigned a pair $(T_{s,i}, T_{e,i})$, and an on-line voltage scaling phase, where the decision of whether and how to execute a task will be made. In short, a task v_i will be scheduled for execution in a time interval of length $T_{e,i}$ if its execution time at the reference voltage is less than $T_{s,i}$. This is shown in Figure 4.

Before we elaborate the details of the proposed algorithm, we show how to calculate the expected energy consumption per iteration by the O^2ME algorithm. We have defined $e_{i,j,ref}$ as the execution time of vertex v_i at the reference voltage in the jth iteration. $E_{i,j}(T_{e,i})$, the minimum energy to complete workload $e_{i,j,ref}$ in time slot $T_{e,i}$, is given as follows: on an ideal variable voltage processor, $E_{i,j}(T_{e,i})$ is the energy consumed by running at voltage V_{ideal} throughout the entire assigned slot $T_{e,i}$, where V_{ideal} is the voltage level that enables the processor to accumulate the workload $e_{i,j,ref}$ at the end of the slot [Qu 2001]; on a multiple voltage processor with only a finite set of voltage levels $V_1 < V_2 < \cdots$, $E_{i,j}(T_{e,i})$ is the energy consumed by running at V_k for a certain amount of time and then switching to the next higher level V_{k+1} to complete $e_{i,j,ref}$, where $V_k < v_{ideal} < V_{k+1}$ and the switching point can be conveniently calculated from $T_{e,i}$ and $e_{i,j,ref}$ [Ishihara and Yasuura 1998; Qu 2001].

Recall that $p_{i,j}$ is the probability that vertex v_i requires execution time $t_{i,j}$ and $P_{i,l_i} = \sum_{j=1}^{l_i} p_{i,j}$ is the probability that v_i can be completed by time t_{i,l_i}. The completion of an iteration requires the completion of all the vertices. Therefore, if we execute vertex v_i only when it has a real execution time less than or equal to $T_{s,i}$, the probability that an iteration can be completed is

$$\mathcal{Q} = \prod_{i=1}^{n} P_{i,l_i} = \prod_{i=1}^{n} \sum_{j=1}^{l_i} p_{i,j} \tag{8}$$

BEEM uses the provably most energy-efficient ways for voltage scaling on an ideal [Qu 2001] or multiple-voltage processor [Ishihara and Yasuura 1998] in Step 7 (see next subsection for details). The only way to consume less energy and finish the same workload is to have a longer execution time.

```
/*off-line part */
1. find a topological order of the vertices;
2. for each vertex v_i,   l_i = k_i;                          /* assign WCET to v_i*/
3. Q = 1;
4. while (Q > Q_0)
5. { pick v_j that has the maximum (t_{j,l_j} - t_{j,(l_j-1)}) · P_{j,(l_j-1)} / P_{j,l_j};
6.    if (Q · P_{j,(l_j-1)} / P_{j,l_j} > Q_0)
7.       { Q = Q · P_{j,(l_j-1)} / P_{j,l_j};    l_j = l_j - 1; }
8.    else mark v_j unpickable;
9. }
10. C' = Σ t_{i,l_i};                          /* calculate the total assigned time C' */
11. if (C' > M)    exit;                                  /* Q_0 cannot be met */
12. for each vertex v_i
13.    T_{s,i} = t_{i,l_i};   T_{e,i} = T_{s,i} · M / C';
/* on-line part: repeated for each iteration j = 1, 2, ··· */
14. current time t = 0;
15. for each vertex v_i
16.    e_{i,j,ref} = the execution time of v_i at the reference voltage in jth iteration;
17.    if (e_{i,j,ref} > T_{s,i})    terminate the current iteration;
18.    else    scale voltage so v_i is completed at t = t + T_{e,i};
```

Fig. 4. The off-line/on-line minimum-effort (O^2ME) algorithm for energy reduction with completion ratio guarantee.

where l_i satisfies $t_{i,l_i} \leq T_{s,i} < t_{i,l_i+1}$. Further, if we assign $T_{e,i} (\geq T_{s,i})$ CPU units to v_i, the expected energy consumption for a completion is:

$$E_{completion} = \sum_{i=1}^{n} \sum_{j=1}^{l_i} p_{i,j} E_{i,j}(T_{e,i}) \tag{9}$$

The on-line part of the O^2ME approach terminates an iteration early at vertex v_q when it requires an execution time longer than $T_{s,q}$ (Step 17 in Figure 4), which happens with probability $1-P_{q,l_q}$. Note that this also implies the successful execution at vertices $v_1, v_2, \ldots, v_{q-1}$. These "wasted" computations, because the iteration fails, consume energy in the amount of

$$E_{failure}(v_q) = (1 - P_{q,l_q}) \sum_{i=1}^{q-1} \sum_{j=1}^{l_i} p_{i,j} E_{i,j}(T_{e,i}) \tag{10}$$

Hence, the O^2ME's expected energy consumption per iteration is

$$E(\{(T_{s,1}, T_{e,1}), \ldots, (T_{s,n}, T_{e,n})\}) = E_{completion} + \sum_{q=2}^{n} E_{failure}(v_q)$$

The goal of the off-line part in the O^2ME approach is to *minimize* $E(\{(T_{s,1}, T_{e,1}), \ldots, (T_{s,n}, T_{e,n})\})$ *by finding pairs* $(T_{s,i}, T_{e,i})$ *such that* $T_{s,i} \leq T_{e,i}$, $\sum_{i=1}^{n} T_{e,i} \leq \mathcal{M}$, *and* $\mathcal{Q} \geq \mathcal{Q}_0$. The first inequality guarantees that the task is completable at Step 18; the second one requires total allocated CPU units not to

exceed the deadline $\mathcal{M}$; and the last one enforces that the required completion ratio $\mathcal{Q}_0$ is met. Obviously, energy will be minimized only when *minimum effort* is paid. That is, the scheduler should not attempt to achieve completion ratio higher than the required $\mathcal{Q}_0$. We develop a greedy heuristic, which we have described in the previous section for probabilistic performance estimation, to solve this problem. Recall that t_{i,k_i} is the WCET of vertex v_i, Step 2 assigns each vertex a slot t_{i,l_i} equal to its WCET. This guarantees the completion of every iteration (Step 3). Steps 4–9 attempt to trade the extra completion ratio over the required $\mathcal{Q}_0$ for slack time. More specifically, if we assign vertex v_j time $t_{j,(l_j-1)}$ rather than t_{j,l_j}, we save $t_{j,l_j} - t_{j,(l_j-1)}$ in the total time assigned at the cost of reducing the completion ratio by the factor of $\frac{P_{i,(l_i-1)}}{P_{i,l_i}}$. We use the product of $(t_{j,l_j} - t_{j,(l_j-1)})$ and $\frac{P_{i,(l_i-1)}}{P_{i,l_i}}$ as the objective function in Step 5 to select the best vertex. If this results in a completion ratio higher than $\mathcal{Q}_0$ (Step 6), we continue this greedy selection. Otherwise, reducing the assigned time to v_j will violate the completion ratio requirement $\mathcal{Q}_0$ and we mark that this vertex's current assigned time t_{i,l_i} cannot be reduced (Step 8). When this greedy selection cannot lower the completion ratio $\mathcal{Q}$ any closer to $\mathcal{Q}_0$, we exit the while loop in Step 9. We then calculate the total assigned time in Step 10, whose difference from the deadline $\mathcal{M}$ is the slack. We distribute this slack time proportionally in Step 13 to obtain $T_{e,i}$.

In the on-line part, we reset the time for each iteration in Step 14. The for loop in Steps 15–18 execute each vertex sequentially. If a vertex's execution is larger than $T_{s,i}$, we terminate the iteration immediately to save energy by not attempting to achieve a completion ratio higher than required (Step 17). Otherwise, we stretch v_i's execution time to $T_{e,i}$ and scale voltage accordingly to reduce the energy consumption on completing task v_i (Step 18).

O^{2}ME distinguishes itself from the best-effort approaches as follows: (1) it assigns each vertex a fixed execution slot while the execution slot for a vertex in the best-effort approaches is determined by the on-line scheduler at runtime. This makes O^{2}ME simple to be implemented into the real-time operating system (RTOS). (2) it fully utilizes both the statistical execution time information and the completion ratio requirement, while the best-effort approaches consider only the BCET and WCET. This is reflected in their different drop and voltage scaling policies. (3) When the best-effort approach BEEM drops an iteration, it guarantees that this iteration cannot be completed. O^{2}ME does not have such guarantee, but its selection of $T_{s,i}$'s guarantees that the required completion ratio $\mathcal{Q}_0$ will be met statistically when the number of iterations becomes sufficiently large. (4) O^{2}ME, in general, drops an iteration earlier than BEEM and, hence, reduces the energy "wasted" on the "failed" iterations. (5) BEEM scales voltage conservatively based on WCET, while O^{2}ME scales more aggressively, based on $T_{e,i}$, to give more energy saving on completed iterations. Unfortunately, it is hard to analyze O^{2}ME's performance in energy reduction, because it depends on the structure of the data-flow graph and the execution time distribution of each vertex. In the next section, we will use experimental results to show O^{2}ME's energy efficiency. We summarize the proposed O^{2}ME approach by

THEOREM 2. *O^2ME approach provides a statistical guarantee to the completion ratio requirement with the potential of further energy reduction over best-effort approaches.*

6. EXPERIMENTAL RESULTS

The proposed probabilistic design method relaxes the rigid hardware requirements (to meet the WCET scenario) for software implementation and avoids overdesigning the systems. It is a promising innovation beyond conventional design methods based on WCET, when a perfect completion ratio is not vital. In this case, we can effectively trade the performance for other more important system resources, such as energy consumption. We are currently working on prototyping multimedia embedded systems to investigate how the probabilistic timing performance requirement can help us in more economic designs. In this section, we validate the energy efficiency of the proposed off-line/on-line minimum-effort (O^2ME) algorithm over the best-effort approaches and study how parameters, such as size of the application, deadline, and the completion ratio requirement, affect O^2ME's energy efficiency.

6.1 Experiment Setup

To reach the above simulation goals, we conduct experiments on a variety of benchmarks, including both task graphs constructed, based on real-life DSP applications and random task graphs, generated by the TGFF package. The task graphs from real-life DSP applications include: two different implementations of the Fast Fourier Transform (FFT1 and FFT2); Laplace transform (Laplace); a quadrature mirror filter bank (qmf4); the Karplus–Strong music synthesis algorithm with 10 voices (karp10); a measurement application (meas); an upside-down binary tree representing the sum of products computation (sum1); and other task graphs reported in the early literatures [Al-Mouhamed 1990; McCreary et al. 1994; Wu and Gajski 1990; Yang and Gerasoulis 1994]. The randomized task graphs are generated from TGFF (task graph for free) generator [Dick et al. 1998]. The characteristics of these task graphs are listed in Table I.

For each benchmark task graph, we assign each vertex three possible execution times $e_0 < e_1 < e_2$ with probabilities $p_0 \gg p_1 > p_2$, respectively. (In real life design, such execution time distribution information should be collected by profiling tools or by simulation.). One can treat e_0 as the BCET, which occurs much more frequently than others in most DSP applications. We consider a processor with four different voltages ranging from 1.2 to 3.3 V and neglect the overhead of voltage switching for the reasons that we have discussed earlier. The deadline $\mathcal{M}$ varies from the sum of BCET to the sum of WCET, and the required completion ratio $\mathcal{Q}$ varies from 0.4 to 1.0 with a step of 0.05. The following reports are based on the deadlines that are set to be around three times of the sum of BCET (columns 3 and 6 in Table I).

For each pair of $(\mathcal{M}, \mathcal{Q})$, we simulate the execution of each application with the naïve, BEEM, and O^2ME approaches, respectively, and track the completion ratio and energy consumption. Both best-effort approaches (naïve and BEEM) achieve $\mathcal{Q}^{\max}$ that may be higher than the required completion ratio $\mathcal{Q}_0$, but

Table I. Benchmark Task Graphs with Their Numbers of Vertices (n) and Deadline Constraints ($\mathcal{M}$)

Benchmark	n	$\mathcal{M}$	Benchmark	n	$\mathcal{M}$
DSC-7-7	7	39	TGFF1	39	9822
DSC-7-8	7	39	TGFF2	51	11825
meas	12	849	TGFF3	60	16835
qmf4	14	480	TGFF4	74	20146
sum1	15	261	TGFF5	84	23109
Laplace	16	4320	TGFF6	91	25376
almu	17	216	TGFF7	107	30158
karp10	21	1782	TGFF8	117	34060
FFT1	28	2280	TGFF9	131	38015
FFT2	28	1440	TGFF10	147	41903
			TGFF11	163	46039
			TGFF12	174	48837
			TGFF13	188	51624
			TGFF14	199	54271
			TGFF15	212	58372

O^2ME targets only the completion ratio $\mathcal{Q}_0$. The energy consumption comparison will then be in favor of O^2ME as it has completed fewer iterations. To get a fair comparison (at least not in favor of the proposed O^2ME algorithm), we force the naïve and BEEM approaches to achieve $\mathcal{Q}_0$ exactly as follows. We group 100 consecutive iterations and stop the execution once $\lceil 100\ \mathcal{Q}_0 \rceil$ iterations are completed in the same group. This gives us exactly (or slightly above) the required 100,000 $\mathcal{Q}_0$ completions.

6.2 Results and Discussion

We now report a few representative sets of results for the following setup (if not specified otherwise): $\mathcal{Q}_0 = 0.800$, $\mathcal{M}$ equals to four times of the sum of each task's BCET; the four voltages are 3.3, 2.6, 1.9, and 1.2 V.

We first consider the energy consumption by different approaches on real-life benchmarks with fixed-deadline and completion-ratio constraints. Table II reports the energy consumption on each benchmark by the approaches of naïve, BEEM, and O^2ME (columns 2, 3, and 5, respectively). As we have mentioned earlier, both best-effort approaches (naïve approach and BEEM), although they are able to complete every single iteration, achieve exactly the required completion ratio $\mathcal{Q}_0$. The ratio achieved by O^2ME is given in the last column.

Compared with the naïve approach, BEEM provides the same completion ratio with an average of nearly 38% energy saving. The O^2ME approach, as predicted in Theorem 2, is more energy efficient. Specifically, it consistently saves significant amount of energy (an average of 54%) over the naïve approach. It also outperforms the BEEM algorithm, the provably most energy-efficient best-effort approach, by an average of 26%. The actual completion ratio achieved by O^2ME is higher than the required 0.800 on every benchmark and their average is slightly less than 0.814. This implies that the completion ratio guarantee is provided and O^2ME has converted a majority of the unnecessary completions into energy savings, considering that the maximal achievable completion ratio is close to 100% for most of these benchmarks.

Table II. Average Energy Consumption per Iteration for Naïve, BEEM, and O^{2}ME Approaches to Achieve $\mathcal{Q}_0 = 0.800$ with Deadline Constraints $\mathcal{M}$[a]

	Naïve	BEEM		O^{2}ME			
Benchmark	Energy	Energy	Saving Over Naïve	Energy	Saving Over Naïve	Saving Over BEEM	$\mathcal{Q}$
DSC-7-7	12.18	8.19	32.77%	5.32	56.29%	34.99%	0.809
DSC-7-8	12.18	7.65	37.13%	5.18	57.43%	32.28%	0.809
meas	260.10	145.82	43.94%	115.37	55.64%	20.88%	0.8026
qmf4	147.65	72.84	50.67%	67.64	54.19%	7.13%	0.832
sum1	80.30	57.08	28.9%	39.60	50.59%	30.63%	0.8192
Laplace	1322.2	724.39	45.21%	571.42	56.78%	21.11%	0.8222
almu	66.03	35.89	45.64%	28.60	56.68%	20.31%	0.8132
karp10	547.03	382.83	30.02%	264.57	51.64%	30.89%	0.8229
FFT1	699.69	539.02	22.96%	371.00	46.98%	31.17%	0.8043
FFT2	441.91	259.08	41.37%	199.14	54.94%	23.14%	0.8043
Average	/	/	37.86%	/	54.12%	26.16%	0.8139

[a]Unit, the dissipation in one CPU unit at the reference voltage.

Table III. Average Energy Consumption per Iteration for Naïve, BEEM, and O^{2}ME Approaches to Achieve $\mathcal{Q}_0 = 0.800$ with Deadline Constraints $\mathcal{M}$ Specified in Table I[a]

	Naïve	BEEM		O^{2}ME			
Benchmark	Energy	Energy	Saving Over Naïve	Energy	Saving Over Naïve	Saving Over BEEM	$\mathcal{Q}$
TGFF1	3031.4	1724.9	43.10%	1353.6	55.35%	21.53%	0.8028
TGFF2	3541.2	1884.5	46.78%	1573.0	55.58%	16.53%	0.8001
TGFF3	4870.5	2517.5	48.31%	2140.2	56.06%	14.99%	0.8073
TGFF4	5653.3	2748.2	51.39%	2422.6	57.15%	11.85%	0.8005
TGFF5	6362.4	2954.0	53.57%	2691.1	57.70%	8.90%	0.8046
TGFF6	7249.1	3314.0	54.28%	3013.2	58.43%	9.08%	0.8011
TGFF7	8031.6	3621.7	54.91%	3283.5	59.12%	9.34%	0.8001
TGFF8	8918.7	4012.4	55.01%	3582.4	59.83%	10.72%	0.8001
TGFF9	9564.3	4274.9	55.30%	3784.4	60.43%	11.48%	0.8000
TGFF10	10335.9	4514.2	56.32%	4032.1	60.99%	10.68%	0.8010
TGFF11	11073.2	4737.9	57.21%	4303.7	61.13%	9.16%	0.8001
TGFF12	11863.0	5102.6	56.99%	4638.7	60.90%	9.09%	0.8003
TGFF13	12073.5	5149.6	57.35%	4788.2	60.34%	7.02%	0.8014
TGFF14	12738.4	5352.9	57.98%	4853.6	61.90%	9.33%	0.8002
TGFF15	13273.8	5573.4	58.01%	5104.7	61.54%	8.41%	0.8004
Average	/	/	53.77%	/	59.10%	11.21%	0.8013

[a]Unit, the dissipation in one CPU unit at the reference voltage.

Table III gives the results on random benchmarks with various task graph sizes, where we observe the following: (1) BEEM gives an average 53.77% energy saving over the naïve approach and its energy saving increases as the size of task graph increases. This is mainly because of its ability to stop an iteration earlier when it detects a completion is impossible (see line 4, Figure 3). (2) O^{2}ME achieves the required completion ratio 0.800 and the actual completion ratios are fairly close to 0.800, with an average of 0.8013. This is impressive, because the O^{2}ME approach does not count the number of completions. Instead, it drops iterations if a certain vertex's execution time exceeds a given amount (see line 17, Figure 4). (3) O^{2}ME does complete more than required, in most

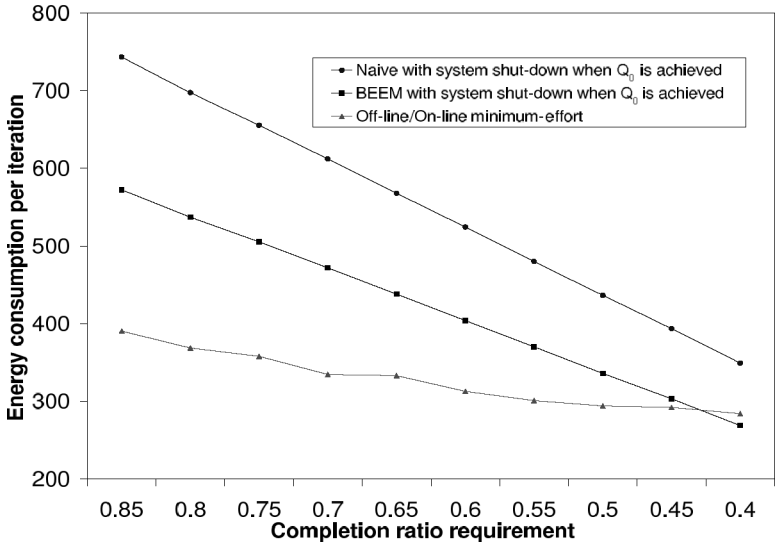


Fig. 5. Different completion ratio requirement $\mathcal{Q}_0$'s impact to the average energy consumption per iteration on benchmark FFT1.

cases. This is because in the off-line part of the O^{2}ME algorithm (see the *while* loop, in Figure 4), our greedy heuristic may not achieve the perfect desired completion ratio $\mathcal{Q}_0$. In general, the larger the task graph, the closer we are to $\mathcal{Q}_0$. (4) O^{2}ME achieves energy reduction of 59.10 and 11.21% over the naïve and BEEM best-effort approaches, respectively. However, the energy saving over BEEM is not monotonically increasing to the size of the task graph. This is also caused by the greedy heuristic in the O^{2}ME algorithm. For example, if it gives a relatively large completion ratio or when it decides to complete the vertices that are scheduled to be executed early.

Second, we consider the impact of different completion ratio requirements to the energy consumed by BEEM and O^{2}ME. Both best-effort approaches have the counting mechanism (to count the number of completions), so their energy consumption should decrease linearly as the completion ratio requirement decreases. Figure 5 depicts this effect for the FFT1 benchmark with a fixed deadline around three times of the sum of BCET. In this case, the best achievable completion ratio by O^{2}ME is slightly larger than 0.85. We see that different completion ratio has much less impact on O^{2}ME's energy consumption, which consists of two parts: the portion on complete iterations and the portion on failed iterations as given in Eqs. (9) and (10). When the completion ratio requirement decreases, the first part decreases, but the second part increases because more and more iterations are intentionally dropped. This becomes clear when the completion ratio is low (around 0.43). O^{2}ME consumes even more energy than BEEM because of the "wasted energy" on failed iterations. On the other end, both best-effort approaches are capable of reaching completion ratio very close to 1 (perfection), which O^{2}ME cannot achieve. This limitation of O^{2}ME is caused by its off-line greedy execution slot reallocation heuristics. When completion

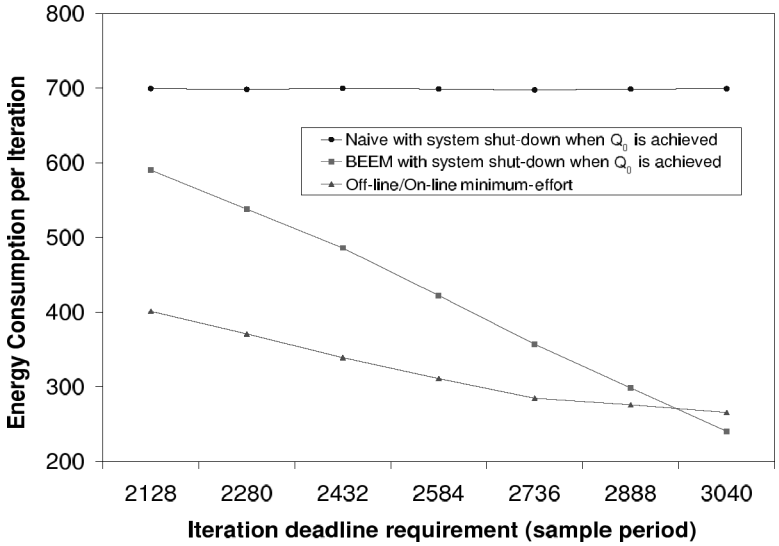


Fig. 6. Different deadline requirement $\mathcal{M}$'s impact to the average energy consumption per iteration on benchmark FFT1.

ratio requirement is high, all vertices will compete for execution slots and O^{2}ME cannot meet the deadline (Step 11 in Figure 4). We conclude that O^{2}ME outperforms best-effort approaches when the completion ratio is moderately high.

Finally, we discuss deadline's impact to energy consumption for the proposed techniques. The naïve approach operates at the highest voltage until the required $\mathcal{Q}_0$ is reached. Therefore, its energy consumption remains constant regardless of the deadline. In BEEM, the hard/soft deadline pair (T_{l_i}, T_{e_i}) (Eqs. 5–7) is directly related to the iteration deadline $\mathcal{M}$. In fact, the last vertex has both T_{l_i} and T_{e_i} set to $\mathcal{M}$. The pair for other vertices are defined recursively from this. When $\mathcal{M}$ increases, T_{e_i} becomes larger and the system can slow down to save energy without dropping the task. Therefore, BEEM's energy consumption highly depends on the deadline. Similar to the above analysis of completion ratio's impact to O^{2}ME's energy consumption, we see that the deadline also has a positive impact to O^{2}ME, but not as dramatic as it does to BEEM. All these have been verified by simulation, as shown in Figure 6 on the same FFT1 example.

7. CONCLUSIONS

This paper presents the novel concept of probabilistic design for multimedia embedded systems and a methodology to quickly explore such design spaces at the early design stage in order to rapidly achieve economic (multimedia) system design. By taking advantage of multimedia DSP application's unique features, namely, application performance requirements, uncertainties in execution time, and tolerance for reasonable execution failures, our method systematically relaxes the rigid hardware requirements for software implementation and avoids overdesigning the system. There are two key steps in our probabilistic design

methodology, which are the probabilistic timing performance estimation and the off-line/on-line resource management. As an example, we show how to design multimedia systems with reduced resource (energy consumption in our case) while providing the desired performance (completion ratio) probabilistically. Experimental results show that our proposed method achieves better designs with significant energy (resource) savings. Our ongoing work includes applying the proposed probabilistic design method to build multimedia embedded systems, measuring the overall energy consumption, and evaluating the systems performance at user level.

REFERENCES

AL-MOUHAMED, M. A. 1990. Lower bound on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Trans. on Software Engineering*, vol. 16, no. 12.

BOLOT, J. AND VEGA-GARCIA, A. 1996. Control mechanisms for packet audio in the internet. *Proceedings of IEEE Infocom*.

BURD, T. D., PERING, T., STRATAKOS, A., AND BRODERSEN, R. 2000. A dynamic voltage-scaled microprocessor system. *IEEE International Solid-State Circuits Conference*. 294–295, 466.

CHATHA, K. S. AND VEMURI, R. 1998. Performance evaluation tool for rapid prototyping of hardware-software codesigns. *9th International Workshop on Rapid System Prototyping* (June). 218–224.

CHEN, C. AND SARRAFZADEH, M. 1999. Probably good algorithm for low power consumption with dual supply voltages. *IEEE/ACM International Conference on Computer Aided Design*. 76–79.

DICK, R. P., RHODES, D. L., AND WOLF, W. 1998. TGFF: Task graphs for free. In *Proceedings of International Workshop Hardware/Software Codesign*. 97–101.

EIKERLING, H. J., HARDT, W., GERLACH, J., AND ROSENSTIEL, W. 1996. A methodology for rapid analysis and optimization of embedded systems. *International IEEE Symposium and Workshop on ECBS* (March). 252–259.

ERNST, R. 1998. Codesign of embedded systems: Status and trends. *IEEE Design & Test of Computers 15*, 2, 45–54.

HENKEL, J. AND ERNST, R. 1998. High-level estimation techniques for usage in hardware/software co-design. *Asia and South Pacific Automation Conference* (Feb.). 353–360.

HONG, I., KIROVSKI, D., QU, G., POTKONJAK, M., AND SRIVASTAVA, M. 1998a. Power minimization of variable voltage core-based systems. *35th ACM/IEEE Design Automation Conference*. 176–181.

HONG, I., POTKONJAK, M., AND SRIVASTAVA, M. B. 1998b. On-line scheduling of hard real-time tasks on variable voltage processor. *IEEE/ACM International Conference on Computer Aided Design*. 653–656.

HSIEH, H., BALARIN, F., LAVAGNO, L., AND SANGIOVANNI-VINCENTELLI, A. L. 2000. Efficient methods for embedded system design space exploration. *37th ACM/IEEE Design Automation Conference*, (June). 607–612.

HU, X., ZHOU, T., AND SHA, E. H.-M. 2001. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Trans. VLSI Systems 9*, 6 (Dec.), 833–844.

HUA, S. AND QU, G. 2003. Approaching the maximum energy saving on embedded systems with multiple voltages. *International Conference on Computer-Aided Design (ICCAD'03)*. 26–29.

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

JOHNSON, M. C. AND ROY, K. 1997. Scheduling and optimal voltage selection for low power multi-voltage dsp datapaths. In *Proceedings of 1997 IEEE International Symposium on Circuits and Systems*. 2152–2155.

KALAVADE, A. AND MOGHE, P. 1998. A tool for performance estimation of networked embedded end-systems. *Proc. Design Automation Conference* (June). 257–262.

KARAM, M. J. AND TOBAGI, F. A. 2001. Analysis of the delay and jitter of voice traffic over the internet. *Infocom*.

MADSEN, J., GRODE, J., KNUDSEN, P. V., PETERSEN, M. E. AND HAXTHAUSEN, A. E. 1997. LYCOS: The lyngby co-synthesis system. *Journal for Design Automation of Embedded Systems* 2, 2 (Mar.). 195–235.

MALIK, S., MARTONOSI, M., AND LI, Y. S. 1997. Static timing analysis of embedded software. *Design Automation Conference* (June). 147–152.

MARCULESCU, R., NANDI, A., LAVAGNO, L., AND SANGIOVANNI-VINCENTELLI, A. L. 2001. System-level power/performance analysis of portable multimedia systems communicating over wireless channels. *IEEE/ACM International Conference on Computer-Aided Design* (Nov.). 207–214.

MCCREARY, C. L. ET AL. 1994. A comparison of heuristics for scheduling dags on multiprocessors. In *Proceedings of the International parallel 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.

QU, G., KAWABE, N., USAMI, K., AND POTKONJAK, M. 2000. Function-level power estimation methodology for microprocessors. *37th ACM/IEEE Design Automation Conference Proceedings* (June). 810–813.

QUAN, G. AND HU, X. 2001. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. *38th IEEE/ACM Design Automation Conference*. 828–833.

RAJE, S. AND SARRAFZADEH, M. 1995. Variable voltage scheduling. *International Symposium on Low Power Electronics and Design*. 9–14.

SHIN, Y. AND CHOI, K. 1999. Power conscious fixed priority scheduling for hard real-time systems. *36th ACM/IEEE Design Automation Conference*. 134–139.

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 Symposium*. 164–173.

WU, M. AND GAJSKI, D. D. 1990. Hypertool: A programming aid for message-passing systems. *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 1.

YANG, P., MARCHAL, P., WONG, C., HIMPE, S., CATTHOOR, F., DAVID, P., VOUNCKX, J., AND LAUWEREINS, R. 2002. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *Proceedings of the 15th International Symposium on System Synthesis*. 112–119.

YANG, T. AND GERASOULIS, A. 1994. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. on Parallel and Distributed Systems*, vol. 2. 951–967.

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.

YUAN, W. AND NAHRSTEDT, K. 2004. Practical voltage scaling for mobile multimedia devices. *ACM Multimedia*. 924–931.

Received September 2003; revised September 2004 and July 2005; accepted March 2006