Energy Efficient Task Assignment with Guaranteed Probability Satisfying Timing Constraints for Embedded Systems

Jianwei Niu, *Senior Member, IEEE*, Chuang Liu, Yuhang Gao, and Meikang Qiu, *Senior Member, IEEE*

Abstract—The trade-off between system performance and energy efficiency (service time) is critical for battery-based embedded systems. Most of the previous work focuses on saving energy in a deterministic way by taking the average or worst scenario into account. However, such deterministic approaches usually are inappropriate in modeling energy consumption because of uncertainties in conditional instructions on processors and time-varying external environments (e.g., fluctuant network bandwidth and different user inputs). By adopting a probabilistic approach, this paper proposes a model and a set of algorithms to address the *Processor and Voltage Assignment with Probability* (PVAP) problem of data-dependent aperiodic tasks in real-time embedded systems, ensuring that all the tasks can be done under the time constraint with a guaranteed probability. We adopt a task DAG (Directed Acyclic Graph) to model the PVAP problem. We first use a processor scheduling algorithm to map the task DAG onto a set of voltage-variable processors, and then use our dynamic programming algorithm to assign a proper voltage to each task. Finally, to escape from local optima, a local search with restarts searches the optimal solution from candidate solutions by updating the objective function, until the stop criteria are reached or a time bound is elapsed. The experimental results demonstrate that for probability 1.0, our approach yields slightly better results than the well-known algorithms like ASAP/ALAP (As Soon As Possible/As Late As Possible) and ILP (Integer Linear Programming) with/without DVS (Dynamic Voltage Scaling). However, for probabilities 0.8 and 0.9, our approach significantly outperforms those algorithms (maximum improvement of 50.3 percent).

Index Terms—Probabilistic scheduling, real-time embedded system, energy efficiency, task assignment

1 Introduction

With embedded devices' increasing capabilities and fast development of mobile/ubiquitious computing, it is essential to elongate standby time for battery-based embedded systems. Also, users of embedded systems desire quick response, energy efficiency and high system reliability. For example, it is important that real-time tasks such as robot control and image processing perform well under real-time constraints. Powerful processors are desirable for these systems. However, generally, more powerful processors consume more power. Therefore, the trade-off between system performance and energy efficiency is of vital importance to battery-based embedded systems. To improve energy efficiency and satisfy the real-time constraints, much work has been done on real-time voltage and frequency scaling techniques [1], [2].

The performance of embedded systems (e.g., smart phones) is also influenced by uncertain environmental factors (e.g., network bandwidth and user input), conditional instructions on processors and power allocation strategies. Although many static assignment techniques can find the best assignment for data-dependent tasks, existing methods are not able to efficiently deal with such uncertainties. Contrary to deterministic methods, we regard the execution time and energy cost of a task as variable due to conditional instructions and different data inputs. We can obtain the probability distribution of the execution time of a task by sampling or profiling [3], [4], [5]. By adopting a probabilistic approach, we can obtain solutions that not only work for hard real-time systems, but also provide more choices of lower total costs for soft real-time applications, satisfying timing constraints with a guaranteed confidence probability (i.e., all the tasks can be done under timing constraints with a guaranteed probability, such as 0.9).

In this paper, we try to tackle the following problem: assuming that multiple data-dependent acyclic tasks need to be executed on a set of processors, how can we minimize the energy consumption while satisfying all the timing constraints and precedence constraints? We assume that processors utilize *Dynamic Voltage Scaling* (DVS) to satisfy demands of energy efficiency in embedded systems, respecting the fact that many well-known processors, such as StrongARM SA1100 [6] and the Intel XScale [7], adopt this technology. The key to solve the problem is to determine how to assign a proper processor and voltage to each data dependent acyclic task in this multi-processor embedded system, while satisfying all the timing and precedence constraints.

By adopting a probabilistic approach, we provide a solution to the problem, and conduct our experiments on six different benchmarks. The experimental results show

- *J. Niu, C. Liu, and Y. Gao are with the State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing 100191, China. E-mail: niujianwei@buaa.edu.cn.*
- *M. Qiu is with the Dept. of Computer Engineering, San Jose State University, San Jose, CA 95192 USA.*

Manuscript received 17 July 2013; revised 15 Sept. 2013; accepted 16 Sept. 2013. Date of publication 30 Sept. 2013; date of current version 16 July 2014. Recommended for acceptance by V. Misic.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.251

that our algorithm can effectively minimize energy consumption while satisfying the timing and precedence constraints with guaranteed confidence probabilities.

The rest of the paper is organized as follows. In Section 2, the related research work in this field is reviewed. Then, we adopt a task DAG (Directed Acyclic Graph) to model the PVAP problem in Section 3. In Section 4, we propose the PVAP_DP and PVAP_LSR algorithms to solve the PVAP problem. The experimental results are presented in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

DVS allows embedded devices to dynamically adjust CPUs' voltages to desired levels to achieve high energy efficiency. There are many DVS techniques that improve energy efficiency for embedded and distributed systems [8]. The main metrics used to categorize these techniques are as follows: periodic/aperiodic tasks, independent/dependent tasks, single/multiple processors, and deterministic/probabilistic scheduling.

Much research has been conducted on DVS for independent real-time tasks with hard deadlines in recent years. The authors of [1] proposed a pseudo-polynomial dynamic programming algorithm to get the optimal solution for a set of periodic real-time tasks in a uni-processor system with discrete voltage levels. For multiprocessor real-time systems, respecting the fact that the actual execution time of a task is often less than the worst-case execution time, the scheduling techniques in [9] allow the remaining tasks to run at lower voltage levels by reusing the time unused by their precedent tasks. These techniques can improve energy efficiency for both independent and dependent tasks.

There is a lot of research work on dependent tasks in distributed and embedded systems using DVS techniques. The authors of [10] proposed an approach to schedule periodic dependent tasks on multi-core processors with discrete voltage levels. This approach first utilizes the retiming technique to transform the periodic dependent tasks into independent tasks. To get the optimal solution for single-core processors, a dynamic programming based pseudo-polynomial algorithm is used. To improve energy efficiency for multi-core processors, this approach iteratively adjusts task scheduling and voltage selection. By adopting a strategy similar to that presented in [10], the authors of [11] addressed scheduling for acyclic dependent tasks on multi-core processors. This approach first generates the initial schedule without energy constraints utilizing any efficient scheduling algorithm, and then it iteratively adjusts voltage to a higher level for more important tasks, while not violating energy constraints. The authors of [2] solved the similar problem proposed in [10] but for acyclic tasks. They proposed an energy-conscious scheduling heuristic which utilizes an objective function to achieve a tradeoff between the quality of schedules and energy consumption. However, because each scheduling decision tends to be confined to local optima, this approach may fail to obtain the optimal solution. To escape from local optima, the authors of [12] used a combined global/local search strategy. This strategy uses a genetic algorithm with simulated heating for global search and Monte Carlo techniques for local search.

In embedded systems, the execution times of some tasks are variable due to conditional instructions (and/or operations) and different inputs [9], [13], [14], [15]. These embedded systems may operate in time-varying environments. It is possible to obtain the statistical distribution of the execution time for each task by sampling or profiling [3], [4], [5]. For energy minimization under the deadline constraints, static scheduling algorithms were proposed based on worst-case or average-case execution time [16] for each task. These methods are pessimistic and often lead to over-designed systems with higher cost. In contrast to deterministic algorithms, probabilistic algorithms [17] adopt a probabilistic approach to avoid over-designed embedded systems by considering uncertainties in execution time while satisfying performance requirements with a certain probability.

Some work has been done on the probabilistic scheduling for embedded systems. The authors of [18] took into account the uncertain execution times, and then introduced probabilistic retiming and probabilistic rotation scheduling for periodic tasks to shorten the timespan of an acyclic task graph. The authors of [19] considered how to utilize the unused time for tasks that complete earlier than expected. The scheduling algorithm proposed by [19] dynamically re-maps tasks to proper processors, and determines the ordering to execute tasks within a processor, and then allocates the unused time to the remaining tasks for improving energy efficiency while meeting the timing constraints. The authors of [20] proposed two optimal algorithms, one for uniprocessor and one for multiprocessor DSP systems, to minimize the expected total energy consumption while satisfying the timing constraint with a guaranteed confidence probability by assuming that the optimal processor assignment is already known. The authors of [21] proposed dynamic programming algorithms to solve the issue of Functional Unit (FU) assignment for heterogeneous embedded systems without considering the constraint of the number of FUs (processors).

Among previous work, [2], [21], and [16] all consider variations in power consumption among different tasks. However, the work in [2] does not propose an effective measure to avoid local optima. The work in [21] and [16] also targets multi-processor embedded systems. However, the work in [21] provides a general framework without fully considering the FU constraint. Furthermore, although the work in [16] takes into account the processor resource constraint, it only considers average-case scenarios and adopts an ineffective task assignment heuristic.

To overcome these limitations, we present an LSR algorithm to find a near-optimal and power-efficient scheduling for DVS enabled embedded systems. First, an efficient processor assignment algorithm is applied to generate a processor assignment. Then, we use a voltage assignment algorithm to minimize energy consumption by adopting a probabilistic approach. Finally, to avoid local optima, a local search with restarts is applied. In short, our major contribution in this work is that our approach can provide some near-optimal solutions with lower total costs for soft real-time applications, satisfying timing constraints with a guaranteed probability.

3 Model and Problem Description

Due to conditional instructions and time-varying external environment (e.g., fluctuating network bandwidth, different user inputs and DVS), tasks of embedded systems (e.g., smart phones) may have different execution times in different cases. In other words, a task may have many probabilistic executions under different conditions. For example, it may take a longer time to execute an image-processing task in a smart phone when the size of input image is larger. The following notation is used to model the probabilistic execution:

Let i be the ID number of a task and $r(i)$ be the execution time of task i. $c_{i,r(i)}, p_{i,r(i)}$ are the cost and probability of task i finished in $r(i)$ TUs, respectively. In this paper, cost is the energy consumed by the processors. pe_m^n is a processor and voltage assignment, which means processor pe_m runs at voltage vol_n (m is the ID of the processor and n is the ID of the voltage level). Here we define B_i as a linked list of (cost, probability, processor and voltage assignment) triplets $(c_{i,r(i)}, p_{i,r(i)}, pe_m^n)$. In more detail, B_i is a linked list that stores only the information of task i. Triplet $(c_{i,r(i)}, p_{i,r(i)}, pe_m^n)$ means: when task i selects processor pe_m at voltage vol_n, it will be finished in $r(i)$ TUs with $c_{i,r(i)}$ EUs and confidence probability $p_{i,r(i)}$. $B = \{B_1, \ldots, B_i, \ldots, B_N\}$, where N is the number of tasks in an embedded system.

We define $T_{pe_m^n}(v)$ as the execution time of each node $v \epsilon V$ when running at voltage level vol_n on processor pe_m. We can obtain the B_i list for each task according to the time CDF $\mathrm{F}(t)$ which gives the accumulated probability for $T_{pe_m^n}(v) \leq t$. First, we can obtain the processor power with different voltages [22]. Second, the time probability density function of each task is obtained by building a historical table and using statistical profiling [22], [23]. Thus, the energy consumption of a task execution is estimated by multiplying processor power and execution time, and then the B_i list can be built up.

In order to investigate the aperiodic real-time task assignment with precedence constraints, a DAG is used to model a set of data dependent tasks. Each node in a DAG is a task, and an edge in the DAG is a data dependency. Generally, memory is shared in a multi-processor system, and it will not make much difference to the cost of data transmission whether two data-dependent tasks are assigned to the same processor. Therefore, in this paper we ignore the difference whether two data-dependent tasks are executed in the same processor or not.

The following notation is used in the mathematical formulation of the problem:

$DAG\ G = \langle V, E \rangle$ is used to represent all the tasks and their data dependencies. $V = \{v_1, \ldots, v_i, \ldots, v_N\}$ is the set of task nodes. N is the number of tasks (nodes) and v_i represents the task i. $E = \{e_{11}, \ldots, e_{ij}, \ldots, e_{NN}\}$ is the set of edges. e_{ij} is equal to 1 if a data dependency exists between v_i and v_j, and 0 otherwise.

Regarding processors with DVS techniques, we have the following assumptions. $PE = \{pe_1, \ldots, pe_i, \ldots, pe_M\}$ is a set of processor units. M is the number of processors in an embedded system. pe_i is the i_{th} processor. $VOL = \{VOL_1, \ldots, VOL_i, \ldots, VOL_M\}$; $VOL_i = \{vol_{i,1}, \ldots, vol_{i,j}, \ldots, vol_{i,L}\}$ is the voltage level set for pe_i; L is the number of available voltage levels for a specific processor.

Next, we discuss how to calculate the probabilistic execution of a DAG. The probabilistic execution of a whole DAG is the combination of probabilistic executions for all nodes in the task graph. Assuming the probabilistic execution $b(c_{i,r(i)}, p_{i,r(i)}, pe_m^n) \in B_i$ is selected for task v_i, and s_i is the start time for task v_i, the total execution time $T_A(G)$, energy consumption $C_A(G)$, confidence probability $P_A(G)$ under the processor and voltage assignment A for a given $DAG\ G$ can be calculated as follows:

$$T_A(G) = \max\{s_i + r(i)\} \quad \forall v_i \in V \tag{1}$$

$$C_A(G) = \sum_{i=1}^{N} c_{i,r(i)} \tag{2}$$

$$P_A(G) = \prod_{i=1}^{N} p_{i,r(i)}. \tag{3}$$

Since task v_i depends upon its predecessors' data outputs, task v_i can be executed if and only if its predecessor tasks are finished. Therefore, the start time s_i of task v_i is equal to the maximum of all its precedence timing constraints.

After the mathematical formulation of the task and processor model, we give the definition of the PVAP (*Processor and Voltage Assignment with Probability*) problem as follows: given a finite processor set *PE*, a voltage level set *VOL*, a probabilistic execution set *B*, a $DAG\ G = \langle V, E \rangle$, a timing constraint T and a confidence probability P, the problem is to determine a proper processor and a voltage level for each task v_i, which provides the minimum energy consumption under timing constraint *T* and the precedence constraint with a guaranteed confidence probability P. To make our idea easily understood, we provide an example of energy-efficient task assignment satisfying the given time constraints with a probabilistic approach in the online supplemental materials which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.251. In the following section, we will discuss how to solve the PVAP problem in detail.

4 Solution to the PVAP Problem

This section presents a variable-voltage probabilistic scheduling solution for aperiodic task graphs with data dependence and CPU resource (number) constraint for the embedded system. The solution is divided into three phases. In the first phase, an initial processor assignment is obtained using an efficient scheduling algorithm (e.g., *As Soon As Possible* (ASAP) [24]). In the second phase, based on this newly generated processor assignment, we propose a voltage assignment algorithm to minimize energy consumption under timing constraints with a guaranteed probability. In the third phase, *Local Search with Restarts* (LSR) [25], [26], [27] is applied to escape local optima since the initial processor assignment may not optimally select a processor for each task. The local search algorithm searches the optimal solution from candidate solutions (the neighboring solution space) by updating the objective function, until the stop criteria are reached or the time bound is elapsed. These three phases will be discussed in the following subsections respectively.

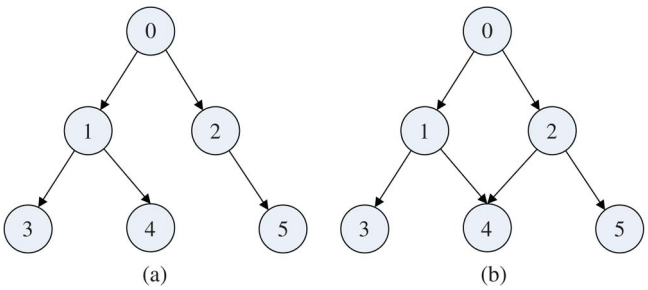


Fig. 1. Common node problem. (a) A task tree whose PVAP problem exhibits the nature of optimal substructure and over-lapping subproblems. (b) A DAG with common node problem.

4.1 Initial Processor Assignment

Since the number of processors in an embedded system is finite, we need to carefully select a proper processor for each task in order to efficiently utilize these processor resources. Local search is applied in our processor and voltage assignment algorithm. To avoid local-optima problem, we reiterate the local search with different initial processor assignments. In this work, the ALAP (As Late As Possible) based list scheduling [24], ASAP based list scheduling, and ILP (Integer Linear Programming) scheduling [16], [28] are used to generate a valid processor assignment.

Our implementation of the list scheduling algorithms (ASAP/ALAP) is described as follows. First, a topological sequence of the task graph is obtained using the topological sorting algorithm. This topological sequence is regarded as the priority list. Second, the execution time for each task is computed assuming that all processors run at their highest frequencies and all tasks run at their worst cases (i.e., the longest execution time on each processor). Then, an efficient scheduling algorithm (ASAP/ALAP) is used to schedule each task in the priority list at the earliest/latest opportunity. We insert fake edges between consecutive tasks running on the same processor to maintain task order. Finally, after each task in a task graph is assigned to a proper processor using ASAP/ALAP, we obtain a new DAG for further solving the PVAP problem. More detailed discussion of ASAP and ALAP algorithms is available in the online supplemental materials available online.

Our implementation of the ILP algorithm is described as follows. First, the execution time for each task is computed assuming that all processors run at their highest frequencies and all tasks run at their worst cases. Then, the mathematical formulation of PVAP problem for ILP is done by adopting a method similar to that of [16], [28]. Finally, we use the nonlinear programming solver of the LINGO 9.0 software to obtain a processor assignment.

4.2 Voltage Assignment

In this subsection, we discuss how to use the *Dynamic Programming* (DP) to solve the voltage assignment in the PVAP problem.

Assume G_i is the subgraph of G rooted at node v_i, containing all the nodes that can be reached by node v_i. G'_i is the subgraph rooted at node v_i, containing all the nodes reached down by node v_i except v_i. $T_A(G_i)$, $C_A(G_i)$ are the total execution time and total energy consumption of G_i under assignment A, and $P_A(G_i)$ is the corresponding probability of G_i finished with $T_A(G_i)$ TUs under assignment A. S_i is defined as the solution space of the PVAP problem for G_i. S_i is a linked list of $s_{i,j}$ (energy consumption, probability) pair $(c_{i,j}, p_{i,j})$, sorted in the ascending order of probability. Here we define $(c_{i,j}, p_{i,j})$ as follows: $p_{i,j}$ is the probability of G_i finishing in j TUs, and $c_{i,j}$ is the minimum energy consumption of $C_A(G_i)$ computed by all voltage assignments satisfying $T_A(G_i) \leq j$ (j is the timing constraint) with $P_A(G_i) \geq p_{i,j}$. Similar to S_i, S'_i is defined as the solution space of the PVAP problem for G'_i. S'_i is also a linked list of $s'_{i,j}$ (energy consumption, probability) pair $(c_{i,j}, p_{i,j})$.

Then, the PVAP problem becomes a question of obtaining S_i for a given graph G. For example, to get S_0 in Fig. 1a, it is necessary to first get through its subgraphs G_1 and G_2. Therefore, in order to obtain S_0, we need to combine the results of these two subgraphs as follows:

1. Let $K = \{S_1, S_2\}$; // S_1 and S_2 are the solution spaces of G_1 and G_2, respectively;
2. $S'_0 = CombineSubSolution(K)$;
3. $S_0 = S'_0 \odot B_0$;
4. remove redundant and infeasible solutions in S_0

In the above steps, Algorithm $CombineSubSolution$ (described in the online supplemental materials available online) combines the optimal assignments of multiple subgraphs to estimate S'_i. In step 3, we introduce the ***operator*** ''$\odot$'', which deals with the transition from S'_i to S_i. The operation ''$\odot$'' between S'_i and B_i is defined as follows: Given S'_i and B_i, after applying ''$\odot$'' between S'_i and B_i, for each $s'_{i,j_1} = (c_{i,j_1}, p_{i,j_1}) \in S'_i$ and each $(c_{i,j_2}, p_{i,j_2}, pe^n_m) \in B_i$, we can obtain $s_{i,j} = (c_{i,j}, p_{i,j})$, whose $p_{i,j} = p_{i,j_1} * p_{i,j_2}$, $c_{i,j} = c_{i,j_1} + c_{i,j_2}$, and $j = j_1 + j_2$.

In step 4, both redundant and infeasible solutions in S_0 are deleted. A redundant (energy, probability) pair is defined as follows: Given two (energy, probability) pairs s_{i,j_1} and s_{i,j_2}, if $p_{i,j_1} \leq p_{i,j_2}$, $c_{i,j_1} \geq c_{i,j_2}$, and $j_1 \geq j_2$, then s_{i,j_1} is called a redundant solution. The solutions which fail to satisfy time and probability constraints are called infeasible pairs.

By adopting the above-mentioned method, we can compute the S_i values in the bottom-up ordering of Fig. 1a. When we get to the root node 0, all the information that we need to compute S_0 is available. Hence, in the bottom-up order, we can compute S_i in a single pass by reusing the solutions of subgraphs. We start with the smallest of subgraphs. Then, the algorithm moves on to solve a larger subproblem until the entire graph is traversed. The solutions of the PVAP problem for subgraphs are stored because these solutions are reused when solving a bigger subgraph.

The optimal solution S_i depends upon the optimal solution of $S_{i_1}, \ldots, S_{i_w}$, where $v_{i_1}, \ldots, v_{i_w}$ are the child nodes of v_i. $S_i, S_{i_1}, \ldots S_{i_w}$, are the solution spaces of the PVAP problem for $G_i, G_{i_1}, \ldots, G_{i_w}$, respectively. Therefore, the PVAP problem for the whole tree can be broken down into multiple PVAP problems for its subtrees. By removing redundant and infeasible solutions in S_i, we obtain a series of candidate solutions that satisfy the timing and probability constraints of each subgraph. Each $s_{i,j}$ pair $(c_{i,j}, p_{i,j})$ in S_i is a possible solution that gives the minimum energy

TABLE 1
Benchmark Descriptions

Benchmarks	Tasks	Edges	Width	Common nodes
TGFF-1	47	46	4	0
TGFF-2	38	41	4	4
TGFF-3	24	28	4	3
TGFF-4	69	72	4	3
consumer	7	8	3	1
auto-industry	9	9	2	1

consumption of subgraph G_i satisfying $T(G_i) \leq j$ (j is the timing constraint) with $P(G_i) \geq p_{i,j}$. Hence we can obtain the optimal solution of the whole tree that satisfies the timing and probability constraint with the minimum energy consumption. Therefore, for a tree, the PVAP problem exhibits the nature of optimal substructure and over-lapping subproblems [29].

However, for a general DAG with an arbitrary number of edges and nodes, there may exist the common node problem, which will result in voltage selection conflicts. A common node is a node that appears in two or more subgraphs. For example, in Fig. 1b, v_4 belongs to both subgraphs G_1 and G_2. It is possible that S_1 selects vol_0 for v_4 to achieve energy efficiency while S_2 prefers vol_1. Therefore, even though both solutions (i.e., S_1 and S_2) for G_1 and G_2 are optimal, S_0 fails to get an optimal assignment because of voltage selection conflicts. In order to solve this problem, we enumerate all the possible combinations for all common nodes. For example, we first select vol_0 for task v_4 to solve the PVAP problem of Fig. 1b. Then, we select vol_1 for task v_4 to solve the PVAP. From all the possible combinations, the PVAP_DP algorithm is able to select the best voltage assignment.

Next we use the PVAP_DP algorithm to solve the voltage assignment problem for a general DAG, as shown in Algorithm 1. First, to solve the smaller subproblems, we need to get the reverse topological ordering in line 1 of Algorithm 1.

Algorithm 1: ***PVAP_DP*** Algorithm

Input: A processor assignment DAG G, a probabilistic execution set B, the timing constraint T, guaranteed probability P, a voltage level set VOL

Output: An optimal voltage assignment S to provide energy minimization while satisfying T respecting guaranteed probability P

1 $Seq \leftarrow \{v_1, \ldots, v_N\}$ //$v_1, \ldots, v_N$ is the reverse topological ordering of the DAG G;
2 $S \leftarrow \emptyset$;
3 $C_{mp} \leftarrow$ {common nodes};
4 $Q \leftarrow$ possible voltage assignments for nodes in C_{mp};
5 **for** *each* $q \in Q$ **do**
6 $S_1 \leftarrow B_1$;
7 **for** $v_i \in Seq$ $i \in [2, N]$ **do**
8 $K \leftarrow \{S_{i_1}, S_{i_2} \ldots S_{i_w}\}$; //$v_{i_1}, v_{i_2}, \ldots, v_{i_w}$ are all child nodes of node v_i
9 $S'_i \leftarrow \begin{cases} \phi & w = 0 \\ CombineSubSolution(K) & w \geq 1; \end{cases}$
10 $S_i \leftarrow S'_i \odot B_i$;
11 remove redundant and infeasible solutions in S_i;
12 **end**
13 $S \leftarrow S \cup S_N$;
14 **end**
15 remove redundant and infeasible solutions in S;
16 return S;

Second, to solve the common node problem, we enumerate the possible voltage assignments for the multi-parent nodes in line 4. Then, we solve a collection of subproblems in the bottom-up direction. Since the first node (here we assume it is v_1) from the reverse topological order has no children, the feasible solutions S_1 are the probabilistic execution list B_1.

To obtain S_i, we first need to get S'_i by combining solutions of $G_{i_1}, G_{i_2} \ldots G_{i_w}$, as shown in line 9 of Algorithm 1. And then, for each $s'_{i,j} \in S'_i$ and each $b \in B_i$, the operation $\odot$ is applied between b and $s'_{i,j}$ to obtain S_i. The redundant and infeasible solutions need to be deleted as shown in line 11 of Algorithm 1. When the inner loop from line 7 to line 12 is finished, we can obtain the optimal voltage assignment S_N for all nodes under the current processor selection and the given voltage assignment of the common nodes. The optimal voltage assignments S_N generated in each inner loop are merged in line 13. When the outer loop from line 5 to line 14 is completed, we can obtain the optimal voltage assignment for all nodes under the current processor selection. After the outer loop is terminated, we remove redundant and infeasible solutions from set S in line 15.

4.3 Local Search with Restarts

In this paper, Local Search with Restarts (LSR) is adopted to search the optimal solution among all the candidate solutions. LSR is presented in Algorithm 2.

Before explaining Algorithm 2, first we need to introduce how to generate the candidate solution space. In Algorithm 2, we adopt a perturb algorithm to perturb the initial scheduling and generate the candidate solutions. First, the topological ordering Seq is obtained using the topological sorting for a DAG. Then, a perturbed ordering Seq' is obtained by perturbing the Seq list. Third, we use ASAP to schedule each task in Seq' to get a task scheduling g' (a graph) under the worst case. If the newly generated graph g' is valid, g' is returned. Otherwise, the loop continues until a valid task scheduling is found. More detailed discussion of the perturb algorithm is available in the online supplemental material.

In Algorithm 2, first, an initial processor scheduling (a DAG) is generated using an efficient scheduling algorithm (e.g., ALAP) in line 6. Second, based on the newly generated DAG, we adopt Algorithm PVAP_DP with local search to minimize energy consumption, as shown in lines 7 to 16 of Algorithm 2. In this step, from the initial scheduling, in the inner loop in line 7, a local search algorithm searches for better processor and voltage assignment in the space of candidate solutions, until the stop criteria are reached, or a time bound is elapsed. The space of candidate solutions is built by perturbing the initial scheduling. However, for the lack of the global information, it is hard for this local search to escape from local optima.

TABLE 2
Voltage Level and Power for ARM Processor and Intel Processor

PE.	Vol.	Freq.	Pow.	Vol.	Freq.	Pow.	Vol.	Freq.	Pow.
Unit	(VU)	(MHz)	(EU)	(VU)	(MHz)	(EU)	(VU)	(MHz)	(EU)
ARM	1.0	700	5.0	1.1	1000	9.8	1.2	1200	17.0
Intel	0.9	900	8.0	1.0	1200	13.6	1.1	1600	19.0

Algorithm 2: ***PVAP_LSR*** Algorithm

Input: A DAG G, a probabilistic execution set B, the timing constraint T, guaranteed probability P, a processor set PE, a voltage level set VOL
Output: A voltage and processor assignment A^* to provide energy minimization satisfying T, P

1 $r \leftarrow 0$; //r is the restart count of local search.
2 $\alpha \leftarrow 5$;//α is an empirical constant. We set it to 5 to balance the computational complexity and algorithm performance.
3 $C^* \leftarrow \infty$; //C^* stores the minimum energy consumption of G.
4 $A^* \leftarrow \phi$; //A^* stores the best voltage and processor assignment ever found.
5 **while** $r < \alpha$ **do**
6 Get the static schedule g (a DAG) from input DAG G using any efficient scheduling algorithm (e.g., ALAP) and let $g' \leftarrow g$;
7 **while** *stop criteria are not reached* $\|$ *time bound is not elapsed* **do**
8 Using algorithm $PVAP\_DP(g', B, T, P, VOL)$ to get the near-optimal assignment of voltage levels for the schedule graph g';
9 $C \leftarrow$ the minimum energy consumption computed by *PVAP_DP* satisfying T under P;
10 $A \leftarrow$ the processor and voltage assignment generated by *PVAP_DP*;
11 **if** $C \leq C^*$ **then**
12 $C^* \leftarrow C$;
13 $A^* \leftarrow A$;
14 **end**
15 Using algorithm Perturb(g) to get another valid schedule g' (a DAG);
16 **end**
17 r++;
18 **end**
19 return A^*;

Third, to avoid local optima, LSR is applied in the outer loop from line 5 to line 18 of Algorithm 2. In line 6, different processor schedulings are used to generate the initial processor assignments such that we can restart local search with different initial schedulings. The outer loop in line 5 is terminated when the number of restarts is reached. The number of restarts is an empirical constant depending on different application scenarios, and we set it to 5 according to our experience in this paper.

5 Experiments

This section evaluates the performance of our proposed PVAP_LSR algorithm. Firstly, we present the benchmarks used in this paper. Secondly, we compare our PVAP_LSR algorithm against the ILP method. Thirdly, the performance of our PVAP_LSR algorithm is evaluated in comparison with the ASAP and ALAP based scheduling algorithms. The ALAP and ASAP scheduling algorithms are widely adopted in various processor and voltage assignment algorithms. The ASAP/ALAP based scheduling algorithm used in our experiments is a combination of ASAP/ALAP and PVAP_DP. The PVAP_LSR algorithm is a combination of LSR and PVAP_DP. In this way, we can evaluate the LSR performance. Moreover, we also compare the PVAP_LSR algorithm with other well known search algorithms like Tabu Search and Simulated Annealing algorithm. The detailed results of comparing with Tabu Search and Simulated Annealing is available in the online supplemental material.

5.1 Benchmarks

We experiment on six benchmarks as shown in Table 1. TGFF1-TGFF4 are obtained using a randomized task-graph generator [30]. Among them, TGFF-1 is a slim graph while TGFF-2 is a fat graph. Basically, TGFF-1 has a very long critical path and there are not many independent nodes; TGFF-2 has a relatively shorter critical path and there are many independent nodes. The consumer and auto-industry

TABLE 3
Energy Comparison among ILP with DVS, ILP without DVS and PVAP_LSR for TGFF-3

TC/ PEs	ILP1	ILP2	PVAP_LSR								
			0.8			0.9			1.0		
	Eng.	Eng.	Eng.	%I1	%I2	Eng.	%I1	%I2	Eng.	%I1	%I2
750/2	21460	18254	15078	29.7	17.4	15784	26.4	13.5	16156	24.7	**11.5**
800/2	20800	15019	12383	40.5	17.6	13026	37.4	13.3	13537	**34.9**	9.9
850/2	20278	14319	12072	**40.5**	15.7	12669	**37.5**	11.5	13241	34.7	7.5
900/2	18953	13867	11821	37.6	14.8	12423	34.5	10.4	13010	31.4	6.2
650/3	20987	16112	13164	37.3	**18.3**	13875	33.9	13.9	14520	30.8	9.9
700/3	20301	15266	12660	37.6	17.1	13129	35.3	**14.0**	13778	32.1	9.7
750/3	19364	14459	12277	36.6	15.1	12858	33.6	11.1	13484	30.4	6.7
800/3	19209	14186	11982	37.6	15.5	12486	35.0	12.0	13173	31.4	7.1
Average Improvements			–	37.2	16.4	–	34.2	12.5	–	31.3	8.6

TABLE 4
Energy Comparison among ILP with DVS, ILP without DVS and PVAP_LSR for Auto-Industry

TC/PEs	ILP1	ILP2	PVAP_LSR								
			0.8			0.9			1.0		
	Eng.	Eng.	Eng.	%I1	%I2	Eng.	%I1	%I2	Eng.	%I1	%I2
320/2	8320	7718	6661	19.9	13.7	7026	15.6	9.0	7728	7.1	-0.1
360/2	8250	6998	5805	29.6	17.0	6321	23.4	9.7	7004	15.1	-0.1
400/2	8250	6409	5024	39.1	21.6	5541	32.8	13.5	6209	24.7	3.1
440/2	8250	5965	4446	46.1	**25.5**	4964	39.8	**16.8**	5621	31.9	**5.8**
480/2	8250	5401	4098	**50.3**	24.1	4617	**44.0**	14.5	5267	**36.2**	2.5
320/3	8169	7587	6665	18.4	12.2	7185	12.0	5.3	7832	4.1	-3.2
360/3	7819	7193	6194	20.8	13.9	6605	15.5	8.2	7253	7.2	-0.8
400/3	7448	6683	5717	23.2	14.5	6121	17.8	8.4	6774	9.0	-1.4
440/3	7510	6003	5071	32.5	15.5	5464	27.2	9.0	6126	18.4	-2.0
480/3	7118	5421	4635	34.9	14.5	5013	29.6	7.5	5688	20.1	-4.9
Average Improvements			–	31.5	17.3	–	25.8	10.2	–	17.4	-0.1

benchmarks are obtained from Embedded Systems Synthesis Benchmarks (E3S) [31]. E3S is largely based on the data from the EDN Embedded Microprocessor Benchmark Consortium (EEMBC) [32]. The consumer benchmark is from consumer electronic applications, such as JPEG compression and decompression. The auto-industry benchmark is from auto-industry applications, such as Fast Fourier Transform and matrix arithmetic.

First, we estimate the CPU power at different frequencies by profiling. In this paper, two kinds of processors are used: Intel processor and ARM processor. All of them are voltage-scalable processors which are allowed to adjust voltage dynamically. Three voltage levels, their corresponding frequencies and powers are shown in Table 2. Columns ''PE'', ''Vol'', ''Freq'' and ''Pow'' represent the processor, voltage level, frequency and power, respectively. Second, to simplify the experiments, we assume that the distribution of execution time of each node is Gaussian (in practice, the distribution can be obtained as done in [23], [22]). Finally, we use a cumulative distribution function to get the B_i list. We obtain all the energy consumptions of each benchmark under different timing constraints. In the following tables and figures, the unit of energy consumption is Energy Unit (EU); the unit of timing constraint or execution time of tasks is Time Unit (TU); column ''TC/PEs'' stands for ''timing constraint/the number of processors'' and Column ''Eng.'' represents the energy consumption. The average improvement of PVAP_LSR over others is shown in the last row of each table. In our experiment, the choice of number of processors to be deployed is kept flexible; it can be either 2 or 3. The former implies that all the tasks are executed on 2 processors, one being an Intel and the other one being an ARM processor. The latter means that all the tasks are executed on 3 processors, two of them being Intel and the third one being an ARM processor.

5.2 Comparison with ILP

In this subsection, we compare our algorithms with ILP. The experimental results for the TGFF-3 and auto-industry are shown in Tables 3 and 4, respectively. In Tables 3 and 4, columns ''ILP1'', ''ILP2'', and ''PVAP_LSR'' represent the results obtained by ILP without DVS from [16], [28], ILP with DVS, and our PVAP_LSR algorithm, respectively. For ILP with DVS, it may take hours, even days to obtain the final results. In our experiments, we set the maximum execution time as four hours for ILP with DVS. Under this time constraint, ILP with DVS cannot generate the optimal solution in most cases. Columns ''%I1'' and ''%I2'' represent the improvement of our algorithms over the ILP without DVS and ILP with DVS, respectively. In this paper, the entries with ''$\times$'' in all the tables indicate that the corresponding assignment fails to generate a valid schedule.

The experimental results show that our PVAP_LSR algorithm can improve energy efficiency while having a guaranteed confidence probability. As shown in Table 3, for TGFF-3, the PVAP_LSR achieves an average (maximum)

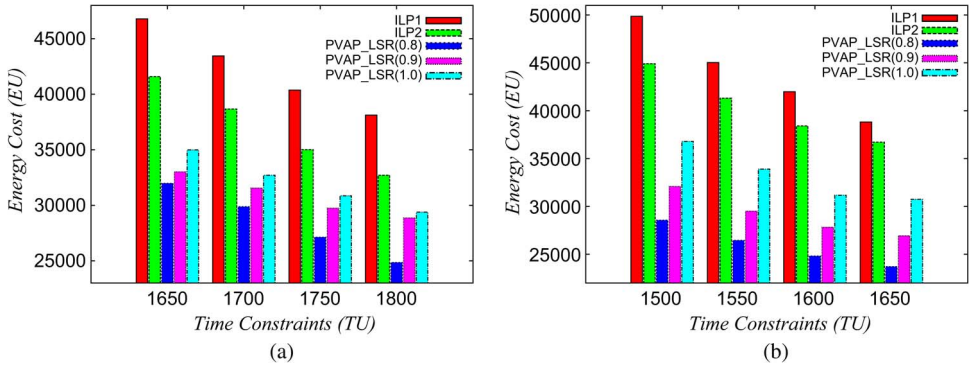


Fig. 2. Energy comparison among ILP1, ILP2 based schedules and PVAP_LSR for TGFF-1. (a) Two processors. (b) Three processors.

TABLE 5
Energy Comparison among ASAP, ALAP Based Schedules and PVAP_LSR for TGFF-2

TC/ PEs	ASAP	ALAP	PVAP_LSR								
			0.8			0.9			1.0		
	Eng.	Eng.	Eng.	%A1	%A2	Eng.	%A1	%A2	Eng.	%A1	%A2
1100/2	×	×	×	×	×	×	×	×	28671	×	×
1150/2	26493	25784	22330	15.7	13.4	22874	13.7	11.3	23525	11.2	8.8
1200/2	25012	24980	21332	14.7	14.6	21729	13.1	13.0	22419	10.4	10.3
1250/2	23071	23845	19988	13.4	16.2	20388	11.6	14.5	21121	8.5	11.4
1300/2	21905	22080	19335	11.7	12.4	19754	9.8	10.5	20389	6.9	7.7
1000/3	24845	26028	19493	**21.5**	**25.1**	20047	**19.3**	**23.0**	20648	**16.9**	**20.7**
1050/3	23993	24177	21243	11.5	12.1	21657	9.7	10.4	22324	7.0	7.7
1100/3	23749	23059	20103	15.4	12.8	20520	13.6	11.0	21254	10.5	7.8
1150/3	22946	22474	19557	14.8	13.0	19978	12.9	11.1	20648	10.0	8.1
1200/3	22447	22681	18966	15.5	16.4	19390	13.6	14.5	20137	10.3	11.2
Average Improvements			–	14.9	15.1	–	13.0	13.3	–	10.2	10.4

power reduction of 16.4 percent (18.3 percent), 12.5 percent (14.0 percent), and 8.6 percent (11.5 percent) with 0.8, 0.9, and 1.0 confidence probabilities against the ILP2 algorithm, respectively. Our PVAP_LSR algorithm also obtains an average (maximum) power reduction of 37.2 percent (40.5 percent), 34.2 percent (37.5 percent), and 31.3 percent (34.9 percent) with 0.8, 0.9, and 1.0 confidence probabilities against the ILP1 algorithm, respectively.

As shown in Table 4, for auto-industry, the PVAP_LSR achieves an average (maximum) power reduction of 17.3 percent (25.5 percent), 10.2 percent (16.8 percent), and −0.1 percent (5.8 percent) with 0.8, 0.9, and 1.0 confidence probabilities against the ILP2 algorithm, respectively. Our PVAP_LSR algorithm also obtains an average (maximum) power reduction of 37.2 percent (50.3 percent), 34.2 percent (44.0 percent), and 31.3 percent (36.2 percent) with 0.8, 0.9, and 1.0 confidence probabilities against the ILP1 algorithm, respectively.

For the TGFF-1 benchmark, Fig. 2 reveals the improvement of our algorithm over ILP1 and ILP2. As shown in Fig. 2, compared with ILP1 and ILP2, the PVAP_LSR algorithm achieves better energy efficiency under all time constraints.

There are two reasons for this high energy effiency. First, the PVAP_LSR algorithm adopts a probabilistic way. Therefore, it has more choices and optimization room. Second, the ILP algorithm has a time limit of four hours. Therefore, the ILP algorithm may not find the near-optimal solution within the time limit.

5.3 Comparison with ALAP, ASAP Based Algorithms

In this subsection, we compare our approach against other two algorithms: ASAP (for processor assignment, ASAP based list scheduling is used; for voltage assignment, the PVAP_DP algorithm is used) and ALAP (for processor assignment, ALAP based list scheduling is used; for voltage assignment, the PVAP_DP algorithm is used).

The experimental results based on TGFF-2 and consumer benchmarks are shown in Tables 5 and 6, respectively. In these two tables, columns ''A1'', ''A2'', and ''PVAP_LSR'' represent the results obtained by the ASAP, ALAP, and our PVAP_LSR algorithm, respectively. Columns '' %A1'' and '' %A2'' represent the improvement of our algorithms over the the ASAP and ALAP algorithm.

In Table 5, for the TGFF-2 benchmark, we can see that PVAP_LSR obtains the best variable-voltage schedule in terms of power consumption in all the cases. PVAP_LSR achieves an average (maximum) power reduction of 15.1 percent (25.1 percent), 13.3 percent (23.0 percent), and 10.4 percent (20.7 percent) with 0.8, 0.9, and 1.0 confidence probabilities against the ALAP based schedule, respectively. The PVAP_LSR algorithm also obtains an average (maximum) power reduction of 14.9 percent (21.5 percent), 13.0 percent (19.3 percent), and 10.2 percent (16.9 percent) with 0.8, 0.9, and 1.0 confidence probabilities against ASAP based schedule, respectively.

In Table 6, for the consumer benchmark, we can see that PVAP_LSR is able to generate the best variable-voltage

TABLE 6
Energy Comparison among ASAP, ALAP Based Schedules and PVAP_LSR for Consumer

TC/ PEs	ASAP	ALAP	PVAP_LSR								
			0.8			0.9			1.0		
	Eng.	Eng.	Eng.	%A1	%A2	Eng.	%A1	%A2	Eng.	%A1	%A2
240/2	×	×	3954	×	×	4180	×	×	4435	×	×
270/2	5070	5287	3533	**30.3**	**33.2**	3880	**23.5**	26.6	4119	**18.8**	**22.1**
300/2	4123	4320	3242	21.4	25.0	3516	14.7	18.6	3800	7.8	12.0
330/2	3752	3794	3206	14.6	15.5	3419	8.9	9.9	3752	0.0	1.1
360/2	3600	3574	3006	16.5	15.9	3289	8.6	8.0	3510	2.5	1.8
240/3	5188	5674	3933	24.2	30.7	4155	19.9	**26.8**	4435	14.5	21.8
270/3	4555	4438	3740	17.9	15.7	3913	14.1	11.8	4256	6.6	4.1
300/3	4335	4528	3435	20.8	24.1	3693	14.8	18.4	3944	9.0	12.9
330/3	4012	3971	3196	20.3	19.5	3426	14.6	13.7	3750	6.5	5.6
360/3	3658	3870	3002	17.9	22.4	3264	10.8	15.7	3510	4.0	9.3
Average Improvements			–	20.4	22.4	–	14.4	16.6	–	7.8	10.1

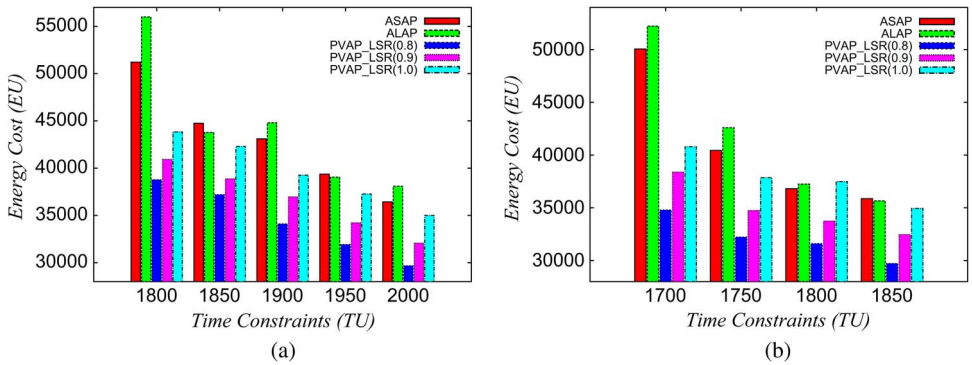


Fig. 3. Energy comparison among ALAP, ASAP based schedules and PVAP_LSR for TGFF-4. (a) Two processors. (b) Three processors.

schedule in terms of power consumption in all the cases. PVAP_LSR achieves an average (maximum) power reduction of 22.4 percent (33.2 percent), 16.6 percent (26.8 percent), and 10.1 percent (22.1 percent) with 0.8, 0.9, and 1.0 confidence probabilities against ALAP based schedule, respectively. PVAP_LSR also obtains an average (maximum) power reduction of 20.4 percent (30.3 percent), 14.4 percent (23.5 percent), and 7.8 percent (18.8 percent) with 0.8, 0.9, and 1.0 confidence probabilities against the ASAP based schedule, respectively.

In Fig. 3, we present the improvement of PVAP_LSR over the ASAP and ALAP schedulings for the TGFF-4 benchmark. As shown in Fig. 3, compared with ASAP and ALAP, the PVAP_LSR algorithm achieves better energy efficiency under all the timing constraints.

6 Conclusion

Existing studies do not pay much attention to the uncertainties in such factors as execution time and energy consumption for real-time embedded systems. In this paper, targeting soft real-time embedded systems with limited resources, we propose a set of processor and voltage assignment algorithms to solve the problem of variable-voltage scheduling of aperiodic tasks with precedence constraints, by adopting a probabilistic approach. The experimental results show that compared with currently popular methods, our proposed approach can improve the energy efficiency of embedded systems and provide users with more choices to achieve energy efficiency, while the timing constraints were satisfied with a guaranteed confidence probability. This is especially useful for soft real-time applications.

Our future work is two-fold: 1) we will combine local and global search to find the best processor and voltage assignment for an acyclic task graph, by adopting a probabilistic approach. For example, we can combine an incremental local optimization heuristic with limited discrepancy search; and 2) we will also further consider the priorities of tasks in our task scheduling algorithms.

Acknowledgment

This work was supported in part by the 973 Program (2013CB035503), National Natural Science Foundation of China (61170296, 61190125), the R&D Program (2013BAH35F01), and NSF CNS-1359557.

References

[1] X. Zhong and C. Xu, "System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation," in *Proc. IEEE/ACM ICCAD*, 2006, pp. 516-521.

[2] Y. Lee and Y. Zomaya, "Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling," in *Proc. 9th IEEE/ACM Int'l Symp. CCGRID*, 2009, pp. 92-99.

[3] T. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. Wu, and J. Liu, "Probabilistic Performance Guarantee for Real-Rime Tasks with Varying Computation Times," in *Proc. Real-Time Technol. Appl. Symp.*, 1995, pp. 164-173.

[4] X. Zhong and C.-Z. Xu, "Energy-Aware Modeling and Scheduling for Dynamic Voltage Scaling with Statistical Real-Time Guarantee," *IEEE Trans. Comput.*, vol. 56, no. 3, pp. 358-372, Mar. 2007.

[5] F. Wang, Y. Chen, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-Aware Task and Communication Mapping for MPSoC Architecture," *IEEE Trans. Comp.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 295-307, Feb. 2011.

[6] A. Acquaviva, L. Benini, and B. Riccó, "Energy Characterization of Embedded Real-Time Operating Systems," *SIGARCH Comput. Architecture News.*, vol. 29, no. 5, pp. 13-18, Dec. 2001.

[7] A. Weissel and F. Bellosa, "Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management," in *Proc. Int'l Conf. CASES*, 2002, pp. 238-246.

[8] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1540-1552, Nov. 2008.

[9] D. Zhu, R. Melhem, and B. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 7, pp. 686-700, July 2003.

[10] H. Liu, Z. Shao, M. Wang, J. Du, C. Xue, and Z. Jia, "Combining Coarse-Grained Software Pipelining with DVS for Scheduling Real-Time Periodic Dependent Tasks on Multi-Core Embedded Systems," *J. Signal Process. Syst.*, vol. 57, no. 2, pp. 249-262, Nov. 2009.

[11] I. Ahmad, R. Arora, D. White, V. Metsis, and R. Ingram, "Energy-Constrained Scheduling of DAGs on Multi-Core Processors," *Contemporary Comput.*, vol. 40, pp. 592-603, Aug. 2009.

[12] N. Bambha, S. Bhattacharyya, J. Teich, and E. Zitzler, "Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors," in *Proc. 9th Int'l Symp. CODES*, 2001, pp. 243-248.

[13] C. Xian, Y. Lu, and Z. Li, "Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time," in *Proc. 44th Annu. DAC*, 2007, pp. 664-669.

[14] S. Liu, Q. Wu, and Q. Qiu, "An Adaptive Scheduling and Voltage/Frequency Selection Algorithm for Real-Time Energy Harvesting Systems," in *Proc. 46th Annu. DAC*, 2009, pp. 782-787.

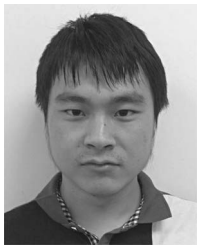
[15] M. Lombardi and M. Milano, "Stochastic Allocation and Scheduling for Conditional Task Graphs in MPSoCs," in *Proc. 12th Int'l Conf. Principles Pract. Constraint Programm.*, 2006, pp. 299-313.

[16] J. Cong and K. Gururaj, "Energy Efficient Multiprocessor Task Scheduling under Input-Dependent Variation," in *Proc. Conf. DATE*, 2009, pp. 411-416.

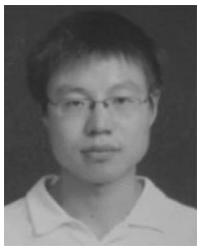
[17] S. Hua, G. Qu, and S. Bhattacharyya, "Exploring the Probabilistic Design Space of Multimedia Systems," in *Proc. 14th IEEE Int'l Workshop RSP*, 2003, pp. 233-240.

[18] S. Tongsima, E. Sha, C. Chantrapornchai, D. Surma, and N. Passos, "Probabilistic Loop Scheduling for Applications with Uncertain Execution Time," *IEEE Trans. Comput.*, vol. 49, no. 1, pp. 65-80, Jan. 2000.
[19] J. Kang and S. Ranka, "Energy-Efficient Dynamic Scheduling on Parallel Machines," in *Proc. HIPC*, vol. 5374, *ser. Lecture Notes in Computer Science*, 2008, pp. 208-219.
[20] M. Qiu, Z. Jia, C. Xue, Z. Shao, and E.H.-M. Sha, "Voltage Assignment with Guaranteed Probability Satisfying Timing Constraint for Real-Time Multiproceesor DSP," *J. VLSI Signal Process. Syst.*, vol. 46, no. 1, pp. 55-73, Jan. 2007.
[21] M. Qiu and E. Sha, "Cost Minimization While Satisfying Hard/Soft Timing Constraints for Heterogeneous Embedded Systems," *ACM Trans. Design Autom. Electron. Syst.*, vol. 14, no. 2, pp. 1-30, Apr. 2009.
[22] Y. Lu, T. Nolte, J. Kraft, and C. Norstrom, "A Statistical Approach to Response-Time Analysis of Complex Embedded Real-Time Systems," in *Proc. IEEE 16th Int'l Conf. Embedded RTCSA*, 2010, pp. 153-160.
[23] A. Abdallah, W. Wolf, and G. Hellestrand, "Statistical Characterization of Execution Time through Simulation," in *Proc. Int'l Workshop Intell. Solutions Embedded Syst.*, 2008, pp. 1-13.
[24] D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*. New York, NY, USA: Springer-Verlag, 2009.
[25] H. Li and A. Lim, "Local Search with Annealing-Like Restarts to Solve the VRPTW," *Eur. J. Oper. Res.*, vol. 150, no. 1, pp. 115-127, Oct. 2003.
[26] L. Michel, A. See, and P. Hentenryck, "Parallel and Distributed Local Search in COMET," *Comput. Oper. Res.*, vol. 36, no. 8, pp. 2357-2375, Aug. 2009.
[27] O. Mengshoel, D. Wilkins, and D. Roth, "Initialization and Restart in Stochastic Local Search: Computing a Most Probable Explanation in Bayesian Networks," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 2, pp. 235-247, Feb. 2011.
[28] Y. Zhang, X. Hu, and D. Chen, "Task Scheduling and Voltage Selection for Energy Minimization," in *Proc. 39th Annu. Design Autom. Conf.*, 2002, pp. 183-188.
[29] E. Bellman, *Dynamic Programming*. New York, NY, USA: Dover, 2003.
[30] K. Vallerio, *Task Graphs for Free (TGFF v3.0)*, Apr. 2008. [Online]. Available: http://ziyang.eecs.northwestern.edu/dickrp/tgff/manual.pdf
[31] R.P. Dick, *Embedded System Synthesis Benchmarks Suites (e3s)*, Nov. 2011.
[32] J. Poovey, T. Conte, M. Levy, and S. Gal-On, "A Benchmark Characterization of the EEMBC Benchmark Suite," *IEEE Micro.*, vol. 29, no. 5, pp. 18-29, Sept. 2009.

Jianwei Niu received the PhD degree in computer science from Beijing University of Aeronautics and Astronautics (BUAA, now Beihang University), China, in 2002. He was a visiting scholar at School of Computer Science, Carnegie Mellon University, USA, from Jan. 2010 to Feb. 2011. He is a professor in the School of Computer Science and Engineering, BUAA. He has published more than 100 referred papers, and filed more than 30 patents in mobile and pervasive computing. He served as the Program Chair of IEEE SEC 2008, Executive Co-chair of TPC of CPSCom 2013, TPC members of InfoCom, Percom, ICC, WCNC, Globecom, LCN, and etc. He has served as associate editor of *International Journal of Ad Hoc and Ubiquitous Computing*, associate editor of *Journal of Internet Technology*, editor of *Journal of Network and Computer Applications* (Elsevier). He received the New Century Excellent Researcher Award from Ministry of Education of China 2009, the first prize of technical invention of the Ministry of Education of China 2012, Innovation Award from Nokia Research Center, and won the best paper award in IEEE ICC 2013, WCNC 2013, ICACT 2013, CWSN 2012 and GreenCom 2010. His current research interests include mobile and pervasive computing, mobile video analysis. He is a senior member of the IEEE.



Chuang Liu received the BE degree from School of Information Science and Engineering, Lanzhou University, China. He is now pursuing the ME degree from Beihang University, China. His research interests include embedded systems and operating systems.



Yuhang Gao received the BE degree from China University of Mining and Technology, and the ME degree from Beihang University, China. His research interests include embedded systems, computer security, and operating systems.

Meikang Qiu received the BE and ME degrees from Shanghai Jiao Tong University, China, and the MS and PhD degrees of computer science from University of Texas at Dallas, in 2003 and 2007, respectively. Currently, he is an Associate Professor of Computer Engineering at San Jose State University. He has worked at Chinese Helicopter R&D Institute, IBM, etc. He is an ACM Senior member. His research interests include cyber security, embedded systems, cloud computing, smart grid, microprocessor, data analytics, etc. A lot of novel results have been produced and most of them have already been reported to research community through high-quality journal (such as *IEEE Transactions on Computer*, *ACM Transactions on Design Automation*, *IEEE Transactions on VLSI*, and *JPDC*) and conference papers (ACM/IEEE DATE, ISSS+CODES and DAC). He has published 3 books, 170+ peer-reviewed journal and conference papers (including 70 journal articles, 100 conference papers), and 3 patents. His research is supported by NSF. He has won Naval Summer Faculty Award in 2012 and Air Force Summer Faculty Award in 2009. His paper about cloud computing has been ranked 1st in Top 25 Hottest Articles published in *JPDC* (*Journal of Parallel and Distributed Computing*, Elsevier) the whole year of 2012. He is the receipt of *ACM Transactions on Design Automation of Electrical Systems* (*TODAES*) 2011 Best Paper Award selected from a 3-year window. In recent four years, he has won another 4 Conference Best Paper Awards (IEEE/ACM ICESS'12, IEEE GreenCom'10, IEEE EUC'10, IEEE CSE'09). He is a Senior Member of the IEEE.