Energy Minimization with Soft Real-time and DVS for Uniprocessor and Multiprocessor Embedded Systems

Meikang Qiu\(^1\) Chun Xue\(^1\) Zili Shao\(^2\) Edwin H.-M. Sha\(^1\)
\(^1\)Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA
\{mxq012100, cxx016000, edsha\}@utdallas.edu

\(^2\)Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
cszshao@comp.polyu.edu.hk

Abstract

Energy-saving is extremely important in real-time embedded systems. Dynamic Voltage Scaling (DVS) is one of the prime techniques used to achieve energy-saving. Due to the uncertainties in execution times of some tasks of systems, this paper models each varied execution time as a random variable. By using probabilistic approach, we propose two optimal algorithms, one for uniprocessor and one for multiprocessor to explore soft real-time embedded systems and avoid over-designing them. Our goal is to minimize the expected total energy consumption while satisfying the timing constraint with a guaranteed confidence probability. The solutions can be applied to both hard and soft real-time systems. The experimental results show that our approach achieves significant energy-saving than previous work.

1 Introduction

Power and energy reductions are critical for real-time embedded systems. In practice, many of these systems can tolerate occasional deadline misses and some tasks in them may not have fixed execution time. Such tasks usually contain conditional instructions and/or operations that could have different execution times for different inputs \([2, 8]\).

It is possible to obtain the execution time distribution for each task by sampling and knowing detailed timing information about the system or by profiling the target hardware.

Also some multimedia applications, such as image, audio, and video data streams, often tolerate occasional deadline misses without being noticed by human visual and auditory systems. For example, in packet audio applications, loss rates between 1\% - 10\% can be tolerated.

Prior approaches for hardware/software codesign of embedded systems guarantee no deadline missing by considering worst-case execution time of each task \([3, 6]\). These approaches are pessimistic and only suitable for hard real-time systems, where any deadline miss will be catastrophic. They are not suitable for soft real-time systems, which can tolerate occasional violations of timing constraints. Using probabilistic approach \([2, 8]\), we can take advantage of this feature and deliver higher performance with less energy consumption. Probabilistic approach can be applied to both hard real-time systems and soft real-time systems. Hard real-time is a special case of soft real-time when the probability equals to 1.

Dynamic voltage scaling (DVS) is one of the most effective techniques to reduce energy consumption \([1, 4, 5, 7]\). In many embedded systems, the supply voltage can be changed by mode-set instructions according to the workload. With the trend of multiple processors being widely used in embedded systems, it is important to study DVS techniques for multiprocessor systems.

In this paper, we use probabilistic approach \([2, 8]\) and DVS to avoid over-designing systems. We propose two novel optimal algorithms, one for uniprocessor and one for multiprocessor embedded systems, to minimize expected value of total energy consumption while satisfying timing constraints with guaranteed probabilities for real-time applications. Hua et al. \([2]\) proposed a heuristic algorithm for uniprocessor. Their data flow graph (DFG) is a simple path. We call the offline part of it as HUA algorithm for convenience. In this paper, we also apply the greedy method of HUA algorithm to multiprocessor and call the new algorithm as Heu.

Our contributions are listed as the following:

1) For uniprocessor, our algorithm \(VAP_S\) gives the optimal solution and achieves significant energy saving than HUA algorithm.

2) For the general problem, that is, multiprocessor, our algorithm \(VAP_M\) gives the optimal solution and achieves significant average energy reductions than the Heu algorithm.

3) Our algorithms are not only optimal, but also provide more choices of smaller expected total energy consumption with guaranteed confidence probabilities satisfying timing constraints. In many situations, algorithms HUA and Heu...
cannot find a solution, while ours can find satisfied results.

4) Our algorithms are practical and quick. In practice, when the number of multi-parent nodes and multi-child nodes in the given Probabilistic Data Flow Graph (PDFG) graph is small, and the timing constraint is polynomial to the size of PDFG, the running times of these algorithms are very small and our experiments are always finished in very short time.

The rest of the paper is organized as following: The models and basic concepts are introduced in Section 2. In Section 3, we give a motivational example. In Section 4, we propose our algorithms. The experimental results are shown in Section 5, and the conclusion is shown in Section 6.

2 Models and Concepts

In this section, we introduce the system model, the energy model, and VAP problem for multiprocessor systems that will be used in the later sections.

System Model: We focus on real-time applications on uniprocessor and multiprocessor embedded systems. Probabilistic Data-Flow Graph (PDFG) is used to model an embedded systems application. A PDFG $G = \langle U, ED, T, V \rangle$ is a directed cyclic graph (DAG), where $U = \{u_1, \ldots, u_n\}$ is a set of nodes representing tasks; $V = \{V_1, \ldots, V_M\}$ is a voltage set; the execution time $T_{v_j}(u)$ is a random variable; $ED \subseteq U \times U$ is the edge set that defines the precedence relations among nodes in $U$. There is a timing constraint $L$ and it must be satisfied for executing the whole PDFG. In multiprocessor systems, each processor has multiple discrete levels of voltages and its voltage level can be changed independently by voltage-level-setting instructions without the influence for other processors.

Energy Model: Dynamic power, which is the dominant source of power dissipation in CMOS circuit, is proportional to $N_e(u) \times C_s \times V^2_{dd}$, where $N_e(u)$ represent the number of computation cycles for node $u$, $C_s$ is the effective switched capacitance, and $V_{dd}$ is the supply voltage. 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 cycle period time (gate delay).

We use the same energy model as in [4, 5, 7]. Let $T$ represent the execution time of a node and $E$ stand for energy consumption. The cycle period time $T_c$ is proportional to $\frac{V_{dd}}{V_{dd} - V_{th}}$, where $V_{th}$ is the threshold voltage and $\alpha \in (1.0, 2.0]$ is a technology dependent constant. Given the number of cycles $N$ of node $u$, the supply voltage $V_{dd}$ and the threshold voltage $V_{th}$, its computation time $T(u)$ and the energy $E(u)$ for node $u$ are calculated as follows:

$$T(u) = N_e(u) \times T_c = N_e(u) \times \frac{k \times V_{dd}}{(V_{dd} - V_{th})^\alpha} \quad (1)$$

$$E(u) = N_e(u) \times C_s \times V^2_{dd} \quad (2)$$

In Equation (1), $k$ is a device related parameter. From Equations (2) and (3), we can see that the lower voltage will prolong the execution time of a node but reduces its energy consumption.

VAP problem: Assume there are maximum $M$ different voltages in a voltage set $V = \{V_1, V_2, \ldots, V_M\}$. For each voltage, there are maximum $K$ execution time variations, although each node may have different execution time variations. An assignment for a PDFG $G$ is to assign a voltage level to each node. In a PDFG $G$, each varied execution time is modeled as a probabilistic random variable, $T_{v_j}(u)$, $1 \leq j \leq M$, representing the execution times of each node $u \in U$ when running at voltage level $V_j$. $P_{v_j}(u)$ and $E_{v_j}(u)$ is corresponding probability and expected value of energy consumption.

We define the voltage assignment with probability (VAP) problem as follows: Given $M$ different voltage levels: $V_1, V_2, \ldots, V_M$, a PDFG $G = \langle U, ED \rangle$ with $T_{v_j}(u)$, $P_{v_j}(u)$, and $E_{v_j}(u)$ for each node $u \in U$ executed on each voltage level $V_j$, a timing constraint $L$ and a confidence probability $P$, find an assignment of voltage level for $G$ that gives the minimum expected total energy consumption $E$ with confidence probability $P$ under timing constraint $L$.

3 Motivational Examples

First we give an example for multiprocessor embedded systems, which is shown in Figure 1. Figure 1(a) shows the input PDFG, and (b) shows the times, expected energy consumption, and probabilities of each node. Each node has two different voltage levels to choose from, and is executed on them with probabilistic execution times.

The number of computation cycles ($N_e$) for a task is proportional to the execution time. The energy consumption ($E$) depends on not only the voltage level $V$, but also the number of computation cycles $N_e$. We use the expected value of energy consumption $(Exp(E))$ as the energy consumption $E$ under a certain voltage level $V$. Under different voltage levels, a task has different expected energy consumptions. The higher the voltage level is, the faster the execution time is, and the more expected energy is consumed. According to the energy model of DVS [7], the computation time is proportional to $V_{dd}/(V_{dd} - V_{th})^2$, where $V_{dd}$ is the supply voltage, $V_{th}$ is the threshold voltage; the energy consumption is proportional to $V^2_{dd}$. So here we assume the computation time of a node under the low voltage ($V_2$) is twice as much as it is under the high voltage ($V_1$); the energy consumption of a node under the high voltage ($V_1$) is four times as much as it is under the low voltage ($V_2$).

We use a schedule to preprocess the input PDFG. After preprocessing, we get the scheduling graph in Figure 1(c), which is a DAG. There are two processors, $P_{R1}$ and $P_{R2}$. Node 5 is a multi-child node, which has two children: 3 and
Node 2 is a multi-parent node, and has two parents: 3 and 4. From Figure 1(b) we can compute the time cumulative distribution functions (CDFs) of each node at different voltage levels.

4. Node 2 is a multi-parent node, and has two parents: 3 and 4. From Figure 1(b) we can compute the time cumulative distribution functions (CDFs) of each node at different voltage levels.

For Figure 1, the minimum total energy consumptions with computed confidence probabilities under the timing constraint are shown in Table 1. For each row of the table, the energy consumption with computed confidence probability under timing constraint 11. The energy saving improvement in each (P, E) pair. Table 2 shows the assignments of our algorithm. Assignment A(u) represents the voltage selection of each node u. Using our algorithm, we achieve minimum total energy consumption 22 with probability 0.80 satisfying timing constraint 11.

4 The Algorithms

4.1 Definitions and Lemma

To solve the VAP problem, we use dynamic programming method traveling the graph in bottom up or top down fashion. For the ease of explanation, we will index the nodes based on bottom up sequence. For example, Figure 1(a) shows a tree indexed by bottom up sequence. The sequence is: u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_n. Define a root node to be a node without any parent and a leaf node to be a node without any child. A multi-child node is a node with more than one child. For example, in Figure 1(a), node 3 and 5 are multi-child nodes. Similarly, a multi-parent node is a node with more than one parent.

Given the timing constraint L, a PDFG G, and an assignment A, we first give several definitions as follows:

1) $G^1$: The sub-graph rooted at node $v_i$, containing all the nodes reached by node $v_i$. In our algorithm, each step will add one node which becomes the root of its sub-graph. For example, in Figure 1 (a), $G^3$ is the tree containing nodes 1, 2, and 3.

2) $E_A(G^1)$ and $T_A(G^1)$: The total energy consumption and total execution time of $G^1$ under the assignment A. In our algorithm, each step will achieve the minimum total energy consumption of $G^1$ with computed confidence probabilities under various timing constraints.

3) Define the (Probability, Energy) pair $(P_{i,j}, E_{i,j})$ as follows: $E_{i,j}$ is the minimum energy consumption of $E_A(G^2)$ computed by all assignments $A$ satisfying $T_A(G^2) \leq j$ with probability $\geq P_{i,j}$.

We introduce the operator “$\oplus$” in this paper. For two (P, E) pairs $H_1$ and $H_2$, if $H_1$ is $(P^1_{i,j}, E^1_{i,j})$, and $H_2$ is $(P^2_{i,j}, E^2_{i,j})$, then, after the $\oplus$ operation between $H_1$ and $H_2$, we get pair $(P', E')$, where $P' = P^1_{i,j} \cdot P^2_{i,j}$ and $E' = E^1_{i,j} + E^2_{i,j}$. We denote this operation as “$H_1 \oplus H_2$”.

In every step in our algorithm, one more node will be included for consideration. The information of this node is stored in local table $B_{i,j}$. A local table store only data of probabilities and energy of a node itself. Table $B_{i,j}$ is the

<table>
<thead>
<tr>
<th>Node id</th>
<th>Time</th>
<th>V Level</th>
<th>Prob.</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>V1</td>
<td>1.00</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>V2</td>
<td>1.00</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>V2</td>
<td>0.80</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>V2</td>
<td>1.00</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>V2</td>
<td>1.00</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>0.80</td>
<td>61</td>
</tr>
</tbody>
</table>

Table 2. Under timing constraint 11, the different assignments between VAP_M and HUA.
local table storing only the data of node \( v_i \). In more detail, \( B_{i,j} \) is a local table of linked lists that store pair \( (P_{i,j}, E_{i,j}) \) sorted by \( P_{i,j} \) in an ascending order; \( E_{i,j} \) is the energy consumption only for node \( u_i \) at time \( j \), and \( P_{i,j} \) is the corresponding probability. The building procedures of \( B_{i,j} \) are as follows. First, sort the execution time variations in an ascending order. Then, accumulate the probabilities of same type. Finally, let \( L_{i,j} \) be the linked list in each entry of \( B_{i,j} \), insert \( L_{i,j} \) into \( L_{i,j+1} \) while redundant pairs canceled out based on Lemma 4.1.

In our algorithm, table \( D_{i,j} \) will be built. Each entry of table \( D_{i,j} \) will store a linked list of \((\text{Probability, Energy})\) pairs sorted by probability in ascending order. In our algorithm, \( D_{i,j} \) is the table in which each entry has a linked list that store pair \( (P_{i,j}, E_{i,j}) \) sorted by \( P_{i,j} \) in an ascending order. Here, \( i \) represents a number node, and \( j \) represents time. For example, a linked list can be \( (0.1, 2)\rightarrow (0.3, 3)\rightarrow (0.8, 6)\rightarrow (1.0, 12) \). Usually, there are redundant pairs in a linked list. We can use the following Lemma to cancel redundant pairs.

**Lemma 4.1** Given \( (P_{i,j}^1, E_{i,j}^1) \) and \( (P_{i,j}^2, E_{i,j}^2) \) in the same list:
1. If \( P_{i,j}^1 = P_{i,j}^2 \), then the pair with minimum \( E_{i,j} \) is selected to be kept.
2. If \( P_{i,j}^1 < P_{i,j}^2 \) and \( E_{i,j}^1 \geq E_{i,j}^2 \), then \( E_{i,j}^2 \) is selected to be kept.

### 4.2 The VAP_S Algorithm

The Algorithm for uniprocessor system is shown in VAP_S. Using dynamic programming, it can give the optimal solution for the VAP problem when there is only one processor.

**The VAP_S Algorithm**

In algorithm VAP_S, first build a local table \( B_{i,j} \) for each node. Next, in step 2 of the algorithm, when \( i = 1 \), there is only one node. We set the initial value, and let \( D_{1,j} = B_{1,j} \). Then using dynamic programming method, build the table \( D_{i,j} \). For each node \( u_i \) under each time \( j \), we try all the times \( k \) \((1 \leq k \leq j)\) in table \( B_{i,j} \). We use "\( \oplus \)" on the two tables \( B_{i,k} \) and \( D_{i-1,j-k} \). Since \( k + (j-k) = j \), the total time of nodes from \( u_1 \) to \( u_i \) is \( j \). The "\( \oplus \)" operation add the energy consumptions of two tables together and multiply the probabilities of two tables with each other. Finally, we use Lemma 4.1 to cancel the conflicting \((\text{Probability, Energy})\) pairs. The new energy consumption in each pair obtained in table \( D_{i,j} \) is the energy consumption of current node \( u_i \) at time \( k \) plus the energy consumption in each pair obtained in \( D_{i-1,j-k} \). Since we have used Lemma 4.1 canceling redundant pairs, the energy consumption of each pair in \( D_{i,j} \) is the minimum total energy consumption for graph \( G' \) with confidence probability \( P_{i,j} \) under timing constraint \( j \).

### Algorithm 4.1 Optimal algorithm for the VAP problem when there is a uniprocessor (VAP_S)

**Input:** \( M \) different voltage levels, a DAG, and the timing constraint \( L \).

**Output:** An optimal voltage assignment

1. Build a local table \( B_{i,j} \) for each node of PDFG. \([V'] \leftarrow N\), where \([V']\) is the number of nodes.
2. Let \( D_{i,j} = B_{i,j} \) for each node \( u_i, i > 1 \) do
   a. For each time \( j \) do
      i. If \( D_{i,j} \) then
         1. For each time \( k \) in \( B_{i,k} \) do
            a. If \( D_{i-1,j-k} \) then
               1. \( D_{i,j} = D_{i-1,j-k} \uplus B_{i,k} \)
          else
             continue
   end if
   end for
   insert \( D_{i,j} \) to \( D_{i,j} \) and remove redundant pairs using Lemma 4.1.
end for
end for
3. return \( D_{N,j} \)

### 4.3 The Algorithms For Multiprocessor

In this subsection, we first give the algorithm of schedule-graph construction. Next, we give a heuristic algorithm \( Heu \), then we propose our novel and optimal algorithm, VAP_M, for multiprocessor embedded systems. We will compare them in the experiments section.

**Algorithm 4.2 Algorithm to get scheduling graph (VAP_SG)**

**Input:** a task graph PDFG

**Output:** a scheduling graph

1. build a graph to show the order using list scheduling.
2. show the dependency in the graph.
3. remove all redundant edges according Lemma 4.2.

For an input PDFG with multiple processors, given the order of nodes and expected energy consumption of each node, the basic steps of our schedule are shown in algorithm VAP_SG. In the graph built in step 1 of VAP_SG, if there is a edge from \( u_i \) to \( u_j \), this means that \( u_i \) is scheduled before \( u_j \) in the same processor or \( u_j \) depends on \( u_i \) in the original PDFG. The new graph is a DAG that represents the order of nodes and dependencies.

**Lemma 4.2** For two nodes \( u_i \) and \( u_j \), there is an edge \( e_{i,j} \), if we can find another separate path \( u_i \rightarrow u_j \), then the edge \( e_{i,j} \) can be deleted.

**The Heu Algorithm**

We first design an heuristic algorithm for multiprocessor systems according to the HUA algorithm in [2], we call this algorithm as \( Heu \). The PDFG now is a DAG and no longer
Algorithm 4.3 Heuristic algorithm for the VAP problem when there are multi-processors and the PDFG is DAG (Heu)

**Input:** $M$ different voltage levels, a DAG, and the timing constraint $L$.

**Output:** a voltage assignment to minimize energy with guaranteed probability $\theta$ satisfying $L$.

1: Scheduling graph construction.

2: For each vertex $u_i$, let $l_i = k_i$; \(1^\text{st}\) assign worst-case time to $u_i$.

3: $P = 1$.

4: while $(P > \theta)$

5: \{ pick $u_i$ that has the maximum $\left(\frac{t_{ui} - t_{(i-1)i}}{t_{(i-1)i}} \right)$, \}$P_i$.

6: $P = P \times \frac{P_{u_i}}{P_{(i-1)i}}$.

7: if $(P > \theta)$

8: $l_i = l_i - 1$.

9: $T = \sum t_{ui}$; \(1^\text{st}\) calculate the total execution time $T$.

10: if $(T \geq L)$ exit; \(1^\text{st}\) if $\theta$ cannot be met.

11: for each vertex $u_i$, set $S_i = t_{ui} \cdot \theta$.

\[
D_{ij} = D_{i,j-k} \oplus B_{i,k}
\]

9. For each possible fixed assignment, we get a $D_{N,j}$.

10. Then use the Lemma 4.1 to remove redundant pairs. $D_{N,j}$ is the table that stored minimum total energy consumption for the graph $G^2$ with confidence probability $P_{i,j}$ under timing constraint $j$.

Now we explain our optimal algorithm $VAP_M$ in details. In $VAP_M$, we exhaust all the possible assignments of multi-parent or multi-child nodes. Without loss of generality, assume we are using bottom up approach. If the total number of nodes with multi-parent is $t$, and there are maximum $K$ variations for the execution times of all nodes, then we will give each of these $t$ nodes a fixed assignment. We will exhaust all of the $K^t$ possible fixed assignments. Algorithm $VAP_M$ gives the optimal solution when the given PDFG is a DAG. In the following, we give the Theorem 4.1 and Theorem 4.2 about this. Due to page limit, we will not give proofs in this paper.

**Theorem 4.1** In each possible fixed assignment, for each pair $(P_{i,j}, E_{i,j})$ in $D_{i,j}$ \((1 \leq i \leq N)\) obtained by algorithm $VAP_M$, $E_{i,j}$ is the minimum total energy consumption for the graph $G^2$ with confidence probability $P_{i,j}$ under timing constraint $j$.

**Theorem 4.2** For each pair $(P_{i,j}, E_{i,j})$ in $D_{N,j}$ \((1 \leq j \leq L)\) obtained by algorithm $VAP_M$, $E_{i,j}$ is the minimum total energy consumption for the given DAG $G$ with confidence probability $P_{i,j}$ under timing constraint $j$.

In algorithm $VAP_M$, there are $K^t$ loops and each loop needs $O(|V|^2 \cdot L \cdot m \cdot K)$ running time. The complexity of Algorithm $VAP_M$ is $O(K^{t+1} \cdot |V|^2 \cdot L \cdot m \cdot M)$. Let $t_{mp}$ be the number of nodes with multi-parent, and $t_{mc}$ be the number of nodes with multi-child, then $t = min(t_{mp}, t_{mc})$. $|V|$ is the number of nodes, $L$ is the given timing constraint, $M$ is the maximum number of voltage levels for each node, and $K$ is the maximum number of execution time variation for each node. The experiments show that algorithm $VAP_M$ runs efficiently.

5 Experiments

This section presents the experimental results of our algorithms. We conduct experiments on a set of benchmarks including volterra filter, 4-stage lattice filter, 8-stage lattice filter, differential equation solver RLS-laguerrre lattice filter, and elliptic filter. Three different voltage levels, $V_1$, $V_2$, and $V_3$, are used in the system, in which a processor under $V_1$ is the quickest with the highest energy consumption and a processor under $V_3$ is the slowest with the lowest energy.
consumption. There are two processors: PR1 and PR2. We compare our VAP_M algorithm with the Heu algorithm. The distribution of execution times of each node is Gaussian. For each benchmark, the first timing constraint we use is the minimum execution time. The experiments are performed on a Dell PC with a P4 2.1 G processor and 512 MB memory running Red Hat Linux 7.3.

The experimental results of RLS-Laguerre filter are shown in Table 3. In the table, column “TC” represents the given timing constraint, “Heu” represents the heuristic algorithm Heu, and “Ours” represents our optimal algorithm VAP_M. The minimum total energy consumption obtained from different algorithms, VAP_M and Heu, are presented in each entry. Columns “1.0”, “0.9”, and “0.8”, respectively, represent the confidence probability is 1.0, 0.9, and 0.8, respectively. Column “%” shows the percentage of reduction on the total energy consumption, compared the results of algorithm Heu. The average percentage reduction is shown in the last row “Ave. Redu(%)” of Table 3, which is computed by averaging energy reductions at all different timing constraints. The entry with “x” means no solution available.

We found in many situations, algorithm VAP_M has significant energy saving than algorithm Heu. For example, in Table 3, under the timing constraint 220, for probability 0.8, the entry under “Heu” is 900, which is the minimum total energy consumption using algorithm Heu. The entry under “Ours” is 268, which means by using VAP_M algorithm, we can achieve minimum total energy consumption 268 with confidence probability 0.8 under timing constraint 220, and the energy reduction is 70.2%.

The experimental results of energy-saving improvement of VAP_M over Heu for different DSP filters are shown in Table 4. “Node #” stands for the number of nodes of a PFG. The experimental results show that our algorithm can greatly reduce the total energy consumption while having a guaranteed confidence probability. On average, algorithm VAP_M gives an energy-saving of 56.1% with confidence probability 0.8 satisfying timing constraints, and energy-savings of 59.3% and 61.7% with confidence probabilities 0.9 and 1.0 satisfying timing constraints, respectively. The experiments using VAP_M on these benchmarks are finished within several minutes using low-end PC.

6 Conclusion

This paper uses the probabilistic approach to explore design space for real-time embedded systems. By taking advantage of the uncertainties in execution time of tasks, our approach relaxes the rigid hardware requirements for software implementation and eventually avoids over-designing the system. For the voltage assignment with probability (VAP) problem, by using Dynamic Voltage Scaling (DVS), we proposed two optimal algorithms, VAP_S and VAP_M, to give the optimal solutions for uniprocessor and multiprocessor embedded systems. Experimental results demonstrated the effectiveness of our approach.

Table 3. The expected total expected $E$ with computed confidence $P$ under various $T$ for RLS-Laguerre filter.

<table>
<thead>
<tr>
<th>Node #</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heu</td>
<td>Ours</td>
<td>Heu</td>
<td>Ours</td>
</tr>
<tr>
<td>130</td>
<td>900</td>
<td>892</td>
<td>0.9</td>
</tr>
<tr>
<td>136</td>
<td>900</td>
<td>892</td>
<td>0.9</td>
</tr>
<tr>
<td>140</td>
<td>900</td>
<td>756</td>
<td>0.9</td>
</tr>
<tr>
<td>180</td>
<td>900</td>
<td>408</td>
<td>0.9</td>
</tr>
<tr>
<td>220</td>
<td>900</td>
<td>268</td>
<td>0.9</td>
</tr>
<tr>
<td>260</td>
<td>225</td>
<td>218</td>
<td>3.2</td>
</tr>
<tr>
<td>272</td>
<td>225</td>
<td>202</td>
<td>10.2</td>
</tr>
<tr>
<td>280</td>
<td>225</td>
<td>186</td>
<td>17.2</td>
</tr>
<tr>
<td>320</td>
<td>225</td>
<td>152</td>
<td>32.4</td>
</tr>
<tr>
<td>360</td>
<td>225</td>
<td>94</td>
<td>58.2</td>
</tr>
<tr>
<td>420</td>
<td>225</td>
<td>58</td>
<td>74.2</td>
</tr>
</tbody>
</table>

Ave. Redu(%) 54.3 57.6 59.2

Table 4. Experimental results of energy-saving improvement of VAP_M over Heu for different DSP filters.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Node #</th>
<th>type</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volterra</td>
<td>27</td>
<td>Tree</td>
<td>58.6%</td>
<td>61.7%</td>
<td>64.8%</td>
</tr>
<tr>
<td>4-stage Lattice</td>
<td>26</td>
<td>Tree</td>
<td>55.9%</td>
<td>59.7%</td>
<td>62.4%</td>
</tr>
<tr>
<td>8-stage Lattice</td>
<td>42</td>
<td>Tree</td>
<td>59.5%</td>
<td>62.8%</td>
<td>65.2%</td>
</tr>
<tr>
<td>Diff. Equ. Solver</td>
<td>11</td>
<td>DAG</td>
<td>52.6%</td>
<td>55.3%</td>
<td>57.4%</td>
</tr>
<tr>
<td>RLS-Laguerre</td>
<td>19</td>
<td>DAG</td>
<td>54.3%</td>
<td>57.6%</td>
<td>59.2%</td>
</tr>
<tr>
<td>Elliptic</td>
<td>34</td>
<td>DAG</td>
<td>55.8%</td>
<td>58.7%</td>
<td>61.3%</td>
</tr>
</tbody>
</table>

Ave. Saving 56.1% 59.3% 61.9%