# Bounds on Power Savings Using Runtime Dynamic Voltage/Frequency Scaling: An Exact Algorithm and A Linear-time Heuristic Approximation

## ABSTRACT

Dynamic voltage/frequency scaling (DVFS) has been shown to be an efficient power/energy reduction technique. Various runtime DVFS policies have been proposed to utilize runtime DVFS opportunities. However, it is hard to know if runtime DVFS opportunities have been fully exploited by a DVFS policy without knowing the upper bounds of possible energy savings. We propose an exact but exponential algorithm to determine the upper bound of energy savings. The algorithm takes into consideration the switching costs, discrete voltage/frequency voltage levels and different program states. We then show a fast linear time heuristic can provide a very close approximate to this bound.

## 1. INTRODUCTION

With CMOS technology scaling down, high power/energy consumption has become a limiting factor in our ability to develop designs not only for battery-operated mobile systems but also for server and desktop systems due to exorbitant cooling, packaging and power costs.

In CMOS systems, dynamic power dissipation varies linearly with frequency and quadratically with supply voltage as shown by the equation $Power \propto \alpha C_L V_{DD}^2 f$, where $\alpha$ is the switching activity factor, $C_L$ is the load capacitance, $V_{DD}$is the supply voltage and $f$ is the clock frequency. Considering that most applications do not need to continuously maintain peak performance, dynamic voltage/frequency scaling (DVFS) trades off performance for energy savings by scaling down the voltage/frequency when peak performance is not required. As an efficient energy reduction technique, DVFS has been implemented in several contemporary microprocessors such as Intel Xscale [4] and Transmeta Crusoe [12].

Various policies have been proposed to use DVFS technique to reduce energy consumption. These policies can be classified as compile-time policies [15] and runtime policies [8, 6, 9, 13, 11, 14, 2] based on when the decisions to switch voltage/frequency are made. Runtime DVFS policies have drawn more research attentions because of the ability to reduce energy assumption in response to variations in workload. Hence, the study of theoretical bounds for energy savings by runtime DVFS is important in the sense of guiding the development of an efficient runtime DVFS policy or assessing a particular policy.

In this paper, we are interested in providing the upper bounds on energy savings (or lower bounds on energy consumption) given a DVFS-enabled processor and a particular workload. Several models have been studied in the past to provide the upper bounds of energy savings by runtime DVFS. Unfortunately, those models either are based on infeasible assumptions such as no voltage/frequency switching costs [5], continuous voltage scaling [16], or assuming linear scalability with CPU speed ignoring non-scalable factors such as off-chip memory accesses and I/O service [10, 5]. Our work overcomes these limitations and takes into consideration switching costs, non-scalable program behaviors and discrete voltage levels to provide accurate upper bounds of energy savings.

The primary contributions of this paper are:

- We propose a realistic model to study the upper bounds on energy savings of runtime DVFS. The model includes a realistic DVFS-enabled processor with discrete voltage levels and the switching overheads due to voltage/frequency scaling. The model also takes into consideration the non-scalable program behaviors including off-chip memory accesses and I/O service.
- We propose an optimal algorithm to provide the exact upper bound, which works efficiently for problems with up to thousands of computation segments that can be independently scaled (referred to as scaling units).
- We provide a linear-time heuristic algorithm to very closely approximate the upper bound, which makes our model work for large problems. For example, we can get an approximate bound for a $3 * 10^5$ scaling unit problem in seven minutes.

The rest of paper is organized as follows: Section 2 describes the model including model assumptions and notation. Section 3 introduces an exact algorithm and discuss the complexity of the algorithm. Section 4 presents the linear-time heuristic algorithm and compares the results with the exact algorithm. Section 5 compares energy bounds predicted by an optimistic analytical model and energy bounds by runtime DVFS with the energy results using an optimal compile-time DVFS policy. Finally, Section 6 summarizes the contributions of our work.

## 2. PROBLEM STATEMENT

We define scaling points to be a series of events such as timer interrupts and cache misses where voltage/frequency scaling can occur. Consider a run of a program on a given data input. The trace of instructions is sliced into $M$ units labelled 1,2,...$M$ by scaling points. These $M$ units are referred to as scaling units, each of which can be scheduled to a specific V/f level. Considering this sequence of scaling units running on a DVFS-enabled microprocessor with $N$ discrete voltage/frequency levels. Our goal is to find a set of V/f assignments ($x_1$, $x_2$, ... ,$x_M$) such that the energy consumption using this set of V/f assignments is minimized while meeting the performance requirements expressed in the form of a deadline. If the scaling units can be made arbitrarily fine, then this method determines the upper bound on energy savings over any possible DVFS policy that can be applied for this program trace. It can be shown that this problem is a NP-hard combinatorial optimization problem (an instance of multiple-dimensional knapsack problem) and can be solved by searching feasible solutions in the solution space. In this paper, we present the exact upper bounds of possible energy savings by an optimal assignment and then introduce a linear-time heuristic algorithm to approximate the exact upper bounds. We first introduce the assumptions and notation used in the paper.

## 2.1 Assumptions and Notations

### *2.1.1 System assumptions*

There are $N$ voltage/frequency scaling levels, namely $V_1/F_1$, $V_2/F_2$, ... and $V_N/F_N$ where $V_N/F_N$ denotes the highest voltage/frequency level. The energy and time overheads for one voltage/frequency scaling from $V_i/F_i$ to $V_j/F_j$ are denoted by $S_E$ and $S_T$. In this paper, we use equations taken from [1] to calculate the overheads:

$$S_E = (1-u) * c * |V_i^2 - V_j^2| \tag{1}$$

$$S_T = \frac{2 * c}{I_{MAX}} |V_i - V_j| \tag{2}$$

| Unit | (t,e) at V1/F1 | (t,e) at V2/F2 |
|---|---|---|
| 1 | (2,1) | (1,4) |
| 2 | (2,1) | (1,5) |
| 3 | (2,1) | (1,4) |
| 4 | (2,1) | (1,4) |

**Table 1: An simple example with four scaling units and two scaling levels.**

where $c$ is the capacitance of the voltage regulator, $u$ is the energy efficiency of the power regulator and $I_{MAX}$ is the maximum allowed current.

### 2.1.2 *Notation*

Let $T_{ij}$ and $E_{ij}$ denote the execution time and energy consumption of the $i^{th}$ scaling unit $u_i$ running at the $j^{th}$ V/f level. $T_{upper}$ refers to the total execution time of all scaling units running at the highest frequency $V_N/F_N$.

$x_i$ is the V/f level assigned to the $i^{th}$ scaling unit. We use the tuple ($x_1$,$x_2$,...,$x_i$) to represent a set of V/f assignments for the first $i$ scaling units and define $SET$($x_1$,$x_2$,...,$x_i$) to be the sets of V/f assignments for the the first $i$ scaling units. Tuples are referred to as partial solutions when $i$ is less than $M$. $t$($x_1$,$x_2$,...,$x_i$) refers to the execution time using the partial solution for the first $i$ scaling unit, which includes time overheads if switching occurs (i.e. if $x_j \neq x_{j+1}$). Similarly, $e$($x_1$,$x_2$,...,$x_i$) refers to the total energy consumption using the partial solution for the first $i$ scaling unit including energy overheads for switchings.

Tuples of length $M$ ($x_1$,$x_2$,...,$x_M$) represent the complete scheduling solutions. Feasible solutions refer to a subset of solutions that satisfy the deadline requirements.

## 3. OPTIMAL ALGORITHM

The standard way to solve the optimal DVFS problem is to search the solution space until an optimal solution has been found and confirmed. Considering the succession of scaling units, we use breadth-first search that generates partial solutions along with the input sequence. The algorithm enumerates all possible V/f levels for the first scaling unit and generates partial solution set ($x_1$) after considering the first scaling unit. Then for each ($x_1$), it enumerates all possible V/f levels for the second scaling unit and generates all partial solutions ($x_1$,$x_2$) after considering the second scaling unit. This process repeats until complete solutions have been generated ($x_1$,$x_2$,...,$x_M$) after considering the last scaling unit.

We can visualize the process as building a state space tree such as the one shown in Figure 1. Each node in the tree represents a problem state and the path from the root node to a level $i$ node represents a solution state that defines a partial solution ($x_1$,$x_2$,...,$x_i$). Starting from a root node, the algorithm branches on possible V/f levels for the first scaling unit and generates level 1 nodes, each of which represents a partial solution ($x_1$). Then for each node at level 1, it branches on all possible V/f levels for the second scaling unit and generates level 2 nodes representing partial solutions ($x_1$,$x_2$). It continues branching from higher level nodes until reaching level $M$ nodes. The naive algorithm would result in $N^{i+1}$ nodes at level $i$ for the general case of $N$ choices per level.

Due to the deadline and the optimality requirements, branching from unpromising nodes that generate infeasible solutions or non-optimal solutions should be avoided. There are two circumstances, referred to as pruning conditions, in which branching from a certain node will be discontinued.

Suppose the partial solution defined by node $k$ is ($x_1$,$x_2$,...,$x_i$). We define $t$($x_1$,$x_2$,...,$x_i$) and $e$($x_1$,$x_2$,...,$x_i$) as the execution time and energy consumption of the node. The shortest remaining time (SRT) of node $k$ is defined as the execution time running the remaining scaling units at the highest frequency, i.e. $t$($x_{i+1} = N$,..., $x_M = N$).

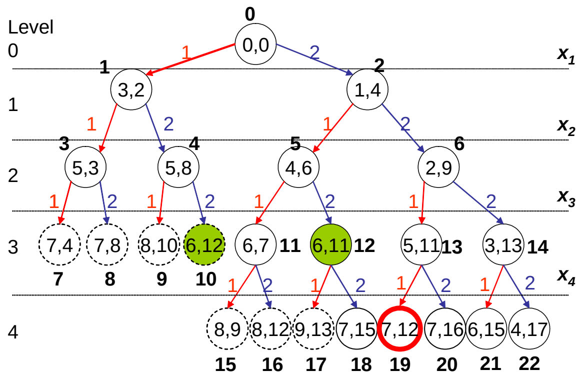

**Figure 1: The state space tree constructed from the four scaling unit example.**

If node $k$ satisfies one of the following conditions, the branching from node $k$ will be discontinued:

1. The sum of the execution time and the shortest remaining time of node $k$ is greater than the deadline, i.e.

   $$t(x_1, x_2, ..., x_i) + t(x_{i+1} = N, ..., x_M = N) > deadline$$

   The inequality means the partial solution will not meet the deadline by running the remaining scaling units at the highest frequency. Thus any solutions generated from this node are infeasible solutions. Nodes satisfying this condition are called infeasible nodes because they generate infeasible solutions.

2. There exists a level $i$ node $l$ ($b_1$,$b_2$,...,$b_i$) such that:

   $$\begin{aligned} b_i &== x_i \\ t(b_1, b_2, ..., b_i) &\leq t(x_1, x_2, ..., x_i) \\ e(b_1, b_2, ..., b_i) &\leq e(x_1, x_2, ..., x_i) \end{aligned}$$

   In this case node $l$ uses both less energy and less time than node $k$ and can always be used instead of node $k$ to generate a better solution without additional switching of V/f levels.

   Since our goal is to find a feasible solution with the lowest energy consumption, if a node is confirmed to generate non-optimal solutions, branching from this node should be stopped to reduce unnecessary node generation. The existence of node $l$ declares that node $k$ will not generate the optimal solution since the best feasible solution generated from node $l$ will always consume less energy consumption than the best feasible solution generated from node $k$. Node $l$ is referred to as the dominating node.

Nodes satisfying either condition 1 or 2 are referred to as dead nodes. We use the function PRUNE to check the status of nodes. If PRUNE ($x_1$,$x_2$,...,$x_i$) returns true, the node is a dead node and there is no need to branch from that node. Otherwise, the node is live and further branching is possible.

Let us look at a simple example. Suppose there are four scaling units running on a processor with two voltage/frequency levels as shown in Table 1. We assume the initial V/f level is V2/F2 and deadline is 7. We also assume the switching time overhead is 1 and energy overhead is 1. The generated state space tree is shown in Figure 1. We start from a root node and branch on the first scaling unit $x_1$. Since there are 2 V/f levels, $x_1$ can be either 1 or 2. Thus root node generates two level 1 nodes: node 1 and node 2. Both nodes are live nodes. We pick node 1 to branch on $x_2$ and generate node 3 and node 4. Then we pick node 2 to branch on $x_2$ and

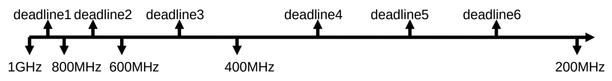

**Figure 2: The positions of deadlines with respect to the execution times using single frequency.**

generate another two level 2 nodes: node 5 and node 6. All four nodes are live nodes, so we continue branching on $x_3$ and generate nodes 7-14. Node 7 satisfies pruning condition 1 since the deadline is 7 and the execution time of node 7 plus the SRT is $7 + 1 = 8$ exceeding the deadline. Thus node 7 is a dead node. For the same reason, node 8 and node 9 are dead nodes. Node 10 is dead because node 10 consumes more energy than node 12 while having the same execution time and assignment for $x_3$. We branch on $x_4$ from node 11 to node 14, which are live nodes, and generate node 15 to node 22 where solid-lined nodes (node 18 to node 22) represent feasible solutions and dot-lined nodes (node 15, node 16 and node 17) are infeasible solutions. Node 19 represents the optimal solution (2,2,1,1) with the minimum energy consumption 12.

The pseudo-code of the optimal algorithm is given as the procedure BRANCH-PRUNE in Algorithm 1.

## 3.1 Practical Complexity

In this section, we will discuss the practical complexity of the optimal algorithm. We first discuss the impact of deadlines on the complexity. Then we will show the optimal algorithm needs exponential runtime with respect to the number of scaling units.

We consider a DVFS-enable processor with five voltage/frequency levels similar to some of the voltage-frequency pairings available in Intel's XScale processors [3]: 0.7V/200MHz, 0.99V/400MHz, 1.3V/600MHz, 1.65V/800MHz and 2.05V/1GHz. We use $c = 10\mu f, I_{MAX} = 1A, u = 90\%$ in Equation (1-2) to calculate switching overheads, which generates switching time of 12 $\mu$s and switching energy of 1.2$\mu$J) for a transition from 600MHz to 200MHz. Note that those settings are parameters and can be changed easily. We consider the size of a scaling unit to be $10^6$ instructions. Four benchmarks from mediabench [7] are used to generate energy/time profiles using SimpleScalar with Wattch. The number of scaling units for each benchmark is listed in Table 2 that includes the execution time and energy consumption using five V/f levels for each benchmark. Six deadlines for each benchmark are picked from tight to loose as shown in Figure 2. Deadline1 sits at the middle of the execution time using 1GHz and the execution time using 800MHz. Deadline2 sits at the middle of 800MHz and 600MHz. Deadline3 sits at the middle of 600MHz and 400MHz. Deadline4 to deadline6 sit evenly between 400MHz and 200Mhz.

We first examine the impact of different deadlines on the algorithm complexity. Figure 3 shows the the number of nodes generated at each level for epic using six difference deadlines for epic. Figures for other benchmarks are similar in shape to Figure 3.

Note that the number of nodes generally increases at first as the level increases and then decreases when approaching the end. This is because at the beginning, few nodes satisfy the first deadline pruning condition. Nodes compete with each other for optimality and only the second pruning condition is responsible for removing non-optimal nodes. As the level grows, the execution time of nodes approach deadlines. Then more nodes are killed by the first deadline pruning condition.

We notice that the middle deadlines generate more nodes than tight deadlines (deadline 1 and deadline 2) and loose deadlines (deadline 5 and 6), which reflects the fact that the solution space shrinks when the deadline approaches the upper bounds and lower bounds of execution time. However, the reasons for the reduced number of nodes are different. For tight deadlines, significant number of nodes are taken out by the first pruning condition because of infeasibility. For loose deadlines, most nodes are killed because of

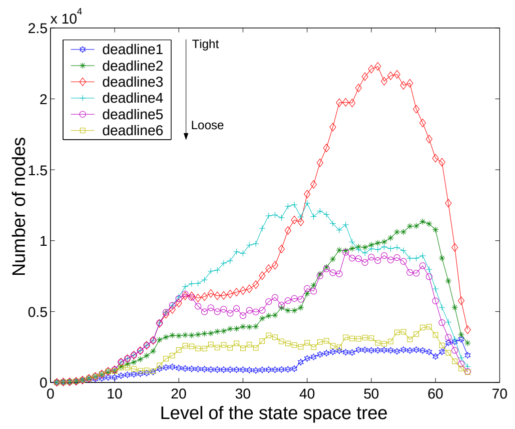

**Figure 3: The number of nodes generated at each level by the exact algorithm using six deadlines from tight to loose.**

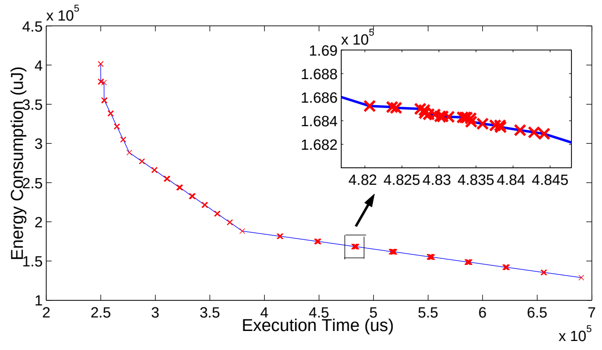

**Figure 4: The energy-delay relationship for all nodes at the same level.**

the confirmed non-optimality by the second pruning condition.

Next, we will examine the relationship between the number of total nodes generated in the state space tree and the number of scaling units. Table 3 shows the total number of nodes for benchmarks using six deadlines.

The adpcm benchmark has 8 scaling units and requires 630 nodes for deadline3. The epic benchmark has 64 scaling units and required nodes increase to $6.3 \times 10^5$, which is 1000 times larger. For the mpeg benchmark, which has 250 scaling units, the number rises to $3.3 * 10^7$, which is $5 * 10^4$ times larger than adpcm. The results demonstrate that the number of nodes grows exponentially as the number of scaling units gets larger. Table 3 also shows the runtime using the exact algorithm. The algorithm works efficiently for small scaling unit sets. epic and gsm take seconds while mpeg takes around 1 minute. However, the runtime increases quickly as the number of scaling units increases. In fact, it takes hours to find the optimal solution when the number of the scaling units rises to thousands. Thus the exact algorithm is impractical for analyzing large problems with tens of thousands scaling units. This motivates our linear-time heuristic algorithm to approximate the bounds provided by the optimal algorithm.

## 4. LINEAR-TIME HEURISTIC ALGORITHM

The pruning function in the optimal algorithm does a good job of removing unpromising nodes and thus significantly shrinking the search paths. However, a vast majority of solutions (partial solutions) must be enumerated before optimality can be confirmed in the worst case for the optimal algorithm. We are looking for a mechanism to further shrink the search scope in the solution space

| | (t,e)@200Mhz | (t,e)@400Mhz | (t,e)@600Mhz | (t,e)@800Mhz | (t,e)@1000Mhz | M scaling units |
|---|---|---|---|---|---|---|
| adpcm | 55.9, 10.3 | 28.0, 15.5 | 18.6, 24.2 | 14.0, 37.3 | 11.2, 56.3 | 9 |
| epic | 422.0, 82.5 | 212.8, 126.3 | 142.0, 199.1 | 106.6, 308.0 | 85.3, 465.4 | 64 |
| gsm | 1064.2, 183.4 | 532.2, 276.1 | 354.8, 433.0 | 266.1, 668.0 | 212.9,1007.6 | 138 |
| mpeg | 1525.8, 285.0 | 763.2, 430.4 | 509.1, 676.0 | 382.0, 1043.8 | 305.7,1575.3 | 250 |

**Table 2: Basic information for benchmarks.**

| | number of nodes | | | | runtime (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | adpcm | epic | gsm | mpeg | adpcm | epic | gsm | mpeg |
| deadline1 | 122 | $0.8 \times 10^5$ | $0.6 \times 10^6$ | $0.6 \times 10^7$ | 0 | 0.05 | 0.84 | 11 |
| deadline2 | 462 | $3.3 \times 10^5$ | $1.4 \times 10^6$ | $2.0 \times 10^7$ | 0 | 0.48 | 2.52 | 44 |
| deadline3 | 630 | $6.3 \times 10^5$ | $1.5 \times 10^6$ | $3.3 \times 10^7$ | 0.1 | 1.12 | 2.67 | 74 |
| deadline4 | 484 | $4.4 \times 10^5$ | $0.7 \times 10^6$ | $2.8 \times 10^7$ | 0 | 0.75 | 0.96 | 62 |
| deadline5 | 319 | $3.2 \times 10^5$ | $0.5 \times 10^6$ | $2.1 \times 10^7$ | 0 | 0.45 | 0.67 | 47 |
| deadline6 | 81 | $1.4 \times 10^5$ | $0.2 \times 10^6$ | $0.8 \times 10^7$ | 0 | 0.12 | 0.2 | 16 |

**Table 3: The number of nodes and runtimes for benchmarks using six deadlines**

by removing unpromising nodes.

Figure 4 shows the energy consumption and execution time of nodes at the same level in the state space tree using the same V/f assignment where each dot represents a node. First, we notice that the trend of the dots is monotonically decreasing. This is due to the second pruning condition. Second, we notice that instead of scattering randomly, nodes are clustered. This is due to the discrete voltage/frequency levels. Those clustered nodes have close energy consumption and execution time as shown in the embedded figure window. Thus the best solutions generated from these clustered nodes also have close energy consumptions. Suppose one of them leads to the optimal solution, then the solutions generated from other nodes produce near-optimal results. If we choose one of them and remove the others, we might remove the optimal node but we can still get a near-optimal solution. If none of the nodes leads to the optimal solution, then there is no harm to keep one and remove others. This way, we can reduce the number of nodes greatly.

We create bins by dividing the energy axis (y axis) evenly into $nbins$ number of ranges. Suppose the the energy consumption of the leftmost node is $E_{max}$ and the rightmost node is $E_{min}$. Then the energy difference between nodes within the same bin is less than $(E_{max} - E_{min})/nbins$. We keep one node in each bin and remove the others. Hence the number of nodes at each level is controlled to be at most $nbins$.

Now we need to decide which node to keep. Considering hardware complications from voltage/frequency scaling such as pipeline flushing, fewer switches are usually preferred. Thus we pick the node with the lowest switching count. If there are multiple nodes with the lowest switching count, we choose the one with the lowest energy-delay product since using energy (or delay) as the metric alone will favor solutions with low frequency (or high frequency).

As shown in Algorithm 1, The heuristic algorithm is built on the exact algorithm. After generating all nodes by the BRANCH-PRUNE procedure, instead of proceeding to the next level, the heuristic traverses the nodes and keeps one node for each bin. This screening procedure is described in Procedure SELECT that selects one node with the lowest energy-delay product from the nodes with the lowest switching count and removes other nodes in each bin.

## 4.1 Algorithm Complexity

Suppose the major costs of statements are $C_1$,$C_2$,$C_3$ and $C_4$ as shown in Algorithm 1. There are at most $b$ nodes at each level. For procedure BRANCH-PRUNE, the first FOR loop needs $b * N * C_1$ steps in the worst case. The second FOR loop needs $b * N * C_2$ steps in total. Thus the total cost for procedure BRANCH-PRUNE is $b * N * (C_1 + C_2)$. Procedure SELECT needs $b * N * C_3 + b * N * C_4$. Therefore, the total cost for building nodes for one level is $b * N * (C_1 + C_2 + C_3 + C_4)$ in worst case. Let $C_{level} = C_1 + C_2 + C_3 + C_4$. Since there are $M$ scaling units in total, the total runtime cost is bounded by $M * b * N * C_{level}$, which is linear in $M$ when $b$ and $N$ are fixed.

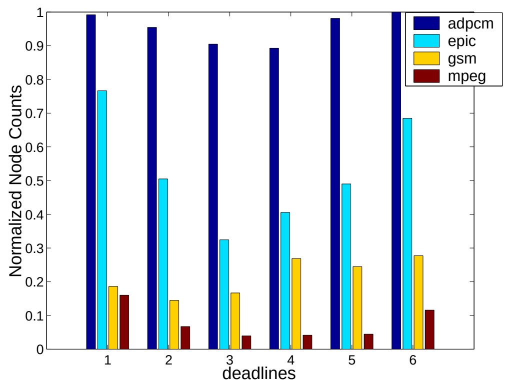

**Figure 5: The number of total nodes generated by the heuristic. The numbers are normalized to the nodes generated by the optimal algorithm.**

Figure 5 shows the number of generated nodes for four benchmarks using 6 different deadlines for 1000 bins. The node number is normalized to the number of nodes generated by optimal algorithm. For small programs such as the adpcm benchmark, the node reduction is not effective. As the number of scaling units increases such as for the mpeg benchmark, the number of nodes is reduced significantly.

## 4.2 Discussion

In this section, we will compare the energy results using the heuristic algorithm with the results using the exact algorithm described in Section 3.

The energy results generated by the exact algorithm and the heuristic are shown in Figure 6. Energy consumption is normalized to the energy consumption using the exact algorithm. As shown in the figure, the heuristic generates higher ”minimum energy”. However, the energy difference is very small especially when the number of bins is big. When using 1000 bins, i.e. at most 1000 nodes at each depth, the results from the heuristic algorithm are very close to the optimal results for adpcm, gsm and mpeg. Only when using

**Algorithm 1** the Heuristic Algorithm

```
91: procedure BRANCH-PRUNE($SET(x_1,..,x_{i-1}),, E(i), T(i)$)
92:     for each $(x_1,..,x_{i-1}) \in SET$ do
93:         for $j \leftarrow 1$ to $N$ do                              ▷ Worse-case cost for one iteration $C_1$
94:             $SRT[i] \leftarrow SRT[i-1] - T[i,N]$
95:             $t(x_1,..,x_i) = t(x_1,..,x_{i-1}) + T(i,j)$
96:             $e(x_1,..,x_i) = e(x_1,..,x_{i-1}) + E(i,j)$
97:             if $PRUNE_{DEADLINE}(x_1,..,x_i) == False$ then          ▷ Pruning Condition 1
98:                 Insert $(x_1,..,x_i)$ in $SET(x_1,..,x_i)$ based on energy
99:             end if
910:        end for
911:    end for
912:    for each $(x_1,..,x_i) \in SET(x_1,..,x_i)$ do                  ▷ Worse-case cost for each iteration $C_2$
913:        if $PRUNE_{OPTIMALITY}(x_1,..,x_i) == TRUE$ then             ▷ Pruning Condition 2
914:            Remove $(x_1,..,x_i)$ from $SET(x_1,..,x_i)$
915:        end if
916:    end for
917: end procedure
918:
919: procedure SELECT($SET, b$)
920:    distribute nodes $(x_1,..,x_i) \in SET$ into $nbins$ bins        ▷ Cost $Length(SET) * C_3$
921:    for $i \leftarrow 1$ to $nbins$ do                                ▷ Worst-case cost for one iteration $C_4$
922:        $tran \leftarrow$ the nodes with the lowest transition counts in the bin
923:        $min \leftarrow$ the node with the lowest energy-delay product in $tran$
924:        Remove nodes other than $min$, the first node and the last node in the bin
925:    end for
926: end procedure
```

| Deadlines | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| exact | 10.9 | 43.7 | 62 | 46 | 16 | 8 |
| 1000 bins | 1.1 | 2.1 | 2.2 | 1.8 | 1.7 | 1.4 |
| 100 bins | 0.18 | 0.2 | 0.19 | 0.16 | 0.15 | 0.14 |
| 10 bins | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.01 |

**Table 4: Runtime (in seconds) for mpeg using the exact algorithm and the heuristic with different number of bins.**

10 bins, the energy increases by a significant amount for certain deadlines.

The heuristic algorithm takes significantly less time than the optimal algorithm. We show the runtime of mpeg using exact and heuristic algorithm in Table 4. The speedup is up to 300X while the energy difference is less than 0.1% using 100 bins. Also the runtime differences between different deadlines are not as dramatic as the exact case.

Now we can use the heuristic to approximate the lower bounds of energy consumptions on large problems. For example, we can use the heuristic to decide the granularity of scaling points. We consider four granularities: $10^4$ instructions, $10^5$ instructions, $10^6$ instructions and $10^7$ instructions. For the finest granularity $10^4$ instructions, gsm has $1.4 * 10^4$ scaling units and mpeg has $2.5 * 10^4$ scaling units. The exact algorithm cannot be used here due to the long runtime (days) and extremely large space required. However, the heuristic can get the approximate bounds in 7 mins for mpeg using 1000 bins. Figure 7 shows the approximate lower bounds of energy consumptions for benchmarks using four granularities. As shown in the figure, the energy consumptions do not vary much from fine granularity to coarse granularity. Thus we cannot achieve more energy savings by using fine granularity, which has been proved in [2] by experiments.

## 5. COMPARISON OF BOUNDS

In this section, we compare the lower bounds of energy con-

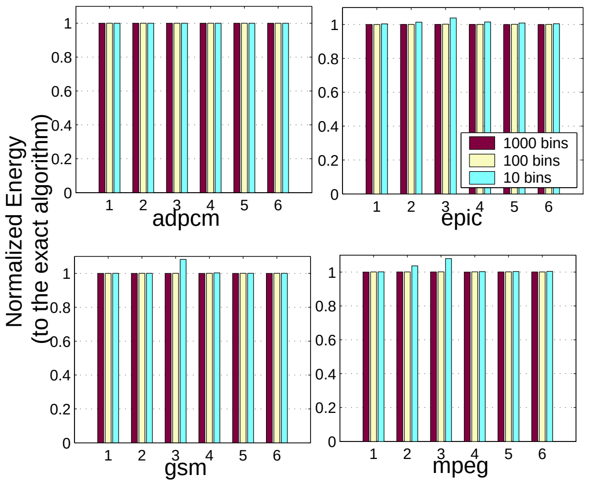

**Figure 6: The energy consumption by the heuristic using different number of bins. Energy consumption is normalized to energy using the the exact algorithm.**

sumptions using the optimistic analytical model from [15] that considers the ideal case where the V/f may be switched at no cost after every instruction, the possible minimum energy consumption for runtime DVFS using the exact algorithm presented in the paper with the actual energy consumption using an optimal compile-time DVFS policy [15]. Six different deadlines are used from tight (deadline1) to loose (deadline6). The energy results are shown in 8. The energy is normalized to the energy using the best single frequency (the lowest frequency that can meet the deadline).

As expected, the analytical model predicts more energy savings than runtime DVFS and compile-time DVFS can possibly achieve. This is because the analytical model assumes no switching costs. Note also that the DVFS method comes quite close to the optimistic

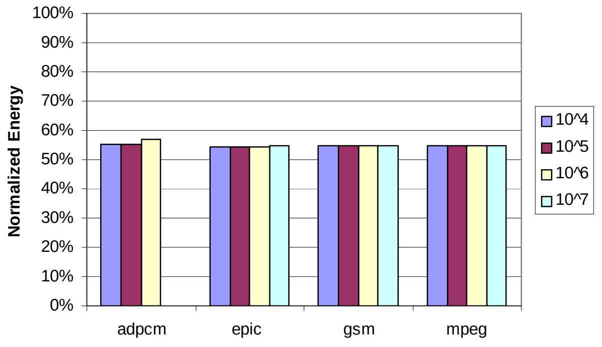

**Figure 7: The approximate lower bounds on energy consumption using different granularities. Energy consumption is normalized to energy using the the highest frequency.**

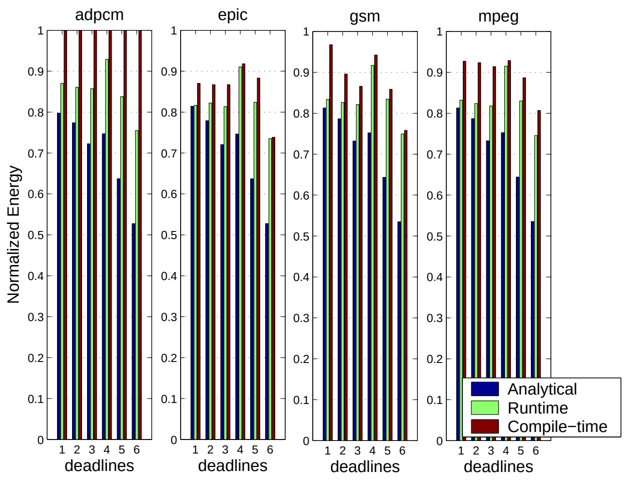

**Figure 8: The minimum energy consumption predicted by an ideal analytical model, the lower bounds of energy consumption from runtime DVFS and the actual energy consumption achieved by a compile-time DVFS policy for six deadlines. Energy consumption is normalized to the best single frequency.**

analytical model, indicating the usefulness of that model despite its simplicity.

The possible minimum energy for runtime DVFS provided by the exact algorithm is lower than the actual energy consumption using an optimal compile-time policy. For adpcm, there is no energy saving using compile-time DVFS while the savings might be up to 18% using runtime DVFS. Except for certain deadlines, epic, gsm and mpeg also show runtime DVFS possibly can achieve more energy savings than compile-time DVFS. This motivates the need for runtime DVFS even in cases of complete program knowledge. The reason is that runtime DVFS can assign different V/f levels to the same piece of static code at different runs while compile-time DVFS is usually confined to static code structure where the piece of static code is assigned to run using same frequency at different runs.

## 6. CONCLUSIONS

We have demonstrated the ability of the algorithm to provide exact upper bounds of energy savings for small to medium problems given a DVFS-enabled processor. We also proposed a linear-time heuristic to approximate the upper bounds for large problem where exact bounds are computationally expensive to get. This model can be used widely to analyze the energy savings from runtime DVFS.

We believe that the model is a powerful tool to guide the development of runtime DVFS policies. We have successfully investigated the impact of scaling granularity, program behavior variation and memory system on the energy savings from runtime DVFS using this model. The development of this model leads to our current work: developing a fast runtime DVFS policy to achieve the energy savings corresponding to upper bounds.

## 7. REFERENCES

[1] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED-00)*, June 2000.

[2] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. pages 18–28, Jan 2005.

[3] L. Clark. Circuit Design of XScale (tm) Microprocessors, 2001. In 2001 Symposium on VLSI Circuits, Short Course on Physical Design for Low-Power and High-Performance Microprocessor Circuits.

[4] Intel Corp. Intel XScale (tm) Core Developer's Manual, 2003. http://developer.intel.com/design/intelxscale/.

[5] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design (ISLPED-98)*, pages 197–202, August 1998.

[6] R. Jejurikar and R. Gupta. Energy aware task scheduling with task synchronization for embedded real time systems. In *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems*, pages 164–169, 2002.

[7] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems. In *Proceedings of the 30th International Symp. on Microarchitecture*, Dec. 1997.

[8] J. Lorch and A. Smith. Improving dynamic voltage algorithms with PACE. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2001)*, June 2001.

[9] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symp. on Operating Systems Principles*, 2001.

[10] G. Qu. What is the limit of energy saving by dynamic voltage scaling? In *Proceedings of the International Conference on Computer Aided Design*, 2001.

[11] A. Sinha and A. Chandrakasan. Dynamic voltage scheduling using adpative filtering of workload traces. In *Proceedings of the 14th International Conference on VLSI Design*, Jan 2001.

[12] Transmeta Corporation. Crusoe processor documentation, 2002. http://www.transmeta.com.

[13] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *the 1st Symposuim on Operating Systems Design and Implementation (OSDI-94)*, pages 13–23, 1994.

[14] A. Weissel and F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, 2002.

[15] F. Xie, M. Martonosi, and S. Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proceedings of ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI'03)*, June 2003.

[16] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, page 374. IEEE Computer Society, 1995.