Bounds on Power Savings Using Runtime Dynamic Voltage Scaling: An Exact Algorithm and a Linear-time Heuristic Approximation

Fen Xie
Dept. of Electrical Engineering
Princeton University
Princeton, NJ
fxie@princeton.edu

Margaret Martonosi
Dept. of Electrical Engineering
Princeton University
Princeton, NJ
mrm@princeton.edu

Sharad Malik
Dept. of Electrical Engineering
Princeton University
Princeton, NJ
sharad@princeton.edu

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.

Categories and Subject Descriptors
I.6.4 [Computing Methodologies]: Simulation and Modelling—*Model Validation and Analysis*

General Terms: Design, Algorithms

Keywords: Low Power, Dynamic Voltage Scaling, Bounds on Energy Savings, Linear Time

1. INTRODUCTION

With aggressive CMOS technology scaling, 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 given by the equation $Power \propto \alpha C_L V_{DD}^2 f$, where α 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 [6] and Transmeta Crusoe [14].

Various policies have been proposed to use DVFS techniques to reduce energy consumption. These policies can be classified as compile-time policies [17] and runtime policies [10, 8, 11, 15, 13, 16, 4] based on when the decisions to switch voltage/frequency are made. Runtime DVFS policies have received more research attention because of the ability to reduce energy consumption 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 often make unrealistic assumptions that limit their value. For example, they assume no cost for switching voltage/frequency [7], they require continuous voltage scaling [18], or they assume linear scaling with CPU speed, ignoring non-scalable factors like off-chip memory accesses or I/O [12, 7].

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 switching overheads due to voltage/frequency scaling. The model also considers non-scalable program behavior 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, our method can generate a bounded near-optimal value for a 300 billion instruction 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 discusses 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 at scaling points 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. Consider this sequence of scaling units running on a DVFS-enabled microprocessor with N discrete voltage/frequency levels. Our goal

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profi t or commercial advantage and that copies bear this notice and the full citation on the fi rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specifi c permission and/or a fee.
ISLPED'05, August 8–10, 2005, San Diego, California, USA
Copyright 2005 ACM 1-59593-137-6/05/0008 ...$5.00.

Unit	(t,e) at V_1/F_1	(t,e) at V_2/F_2
1	(2,1)	(1,4)
2	(2,1)	(1,5)
3	(2,1)	(1,4)
4	(2,1)	(1,4)

Table 1: A simple example with four scaling units and two scaling levels. (t,e) indicate the execution time and energy at each scaling unit.

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 of 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 [2] to calculate the overheads:

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

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

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. To model leakage energy, the CPU consumes a portion of its power even when it is idle due to off-chip accesses.

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. $t(x_{i+1} = N,..., x_M = N)$.

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:

   $$b_i == x_i$$
   $$t(b_1, b_2, ..., b_i) \leq t(x_1, x_2, ..., x_i)$$
   $$e(b_1, b_2, ..., b_i) \leq e(x_1, x_2, ..., x_i)$$

   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. Assume the initial V/f level is V_2/F_2 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

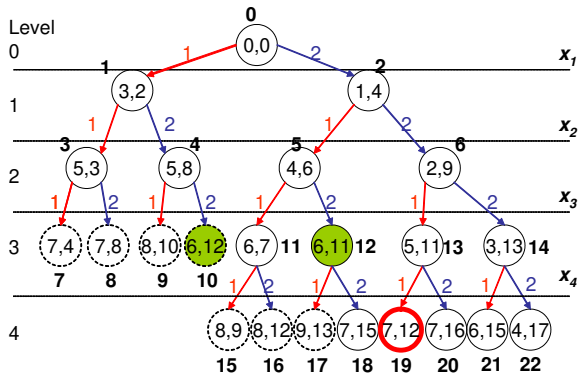

Figure 1: The state space tree constructed from the four scaling unit example.

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 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-enabled processor with five voltage/frequency levels similar to some of the voltage-frequency pairings available in Intel's XScale processors [5]: 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 μs and switching energy of 1.2μJ 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 [9] are used to generate energy/time profiles using SimpleScalar [3] with Wattch [1]. 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 execution times using 800MHz and 600MHz. Deadline3 sits at the middle of execution times using 600MHz and 400MHz. Deadline4 to deadline6 sit evenly between execution times using 400MHz and 200Mhz.

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

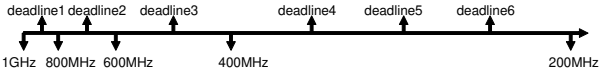

Figure 2: The positions of deadlines with respect to the execution times using single frequency.

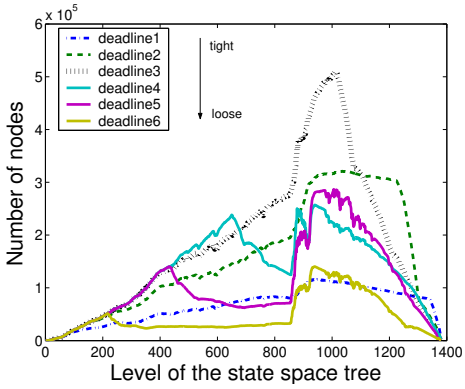

Figure 3: The number of nodes generated at each level for benchmark gsm (1380 scaling units) by the exact algorithm using six deadlines from tight to loose.

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). This 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 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 82 scaling units and requires 396055 nodes for deadline3. The epic benchmark has 646 scaling units and required nodes increase to $5.1 * 10^7$, which is 1000 times larger. For the mpeg benchmark, which has 2179 scaling units, the number rises to $2.1 * 10^9$, 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

	(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	82
epic	422.0, 82.5	212.8, 126.3	142.0, 199.1	106.6, 308.0	85.3, 465.4	646
gsm	1064.2, 183.4	532.2, 276.1	354.8, 433.0	266.1, 668.0	212.9,1007.6	1380
mpeg	1525.8, 285.0	763.2, 430.4	509.1, 676.0	382.0, 1043.8	305.7,1575.3	2179

Table 2: Basic information for benchmarks.

	number of nodes				runtime			
	adpcm	epic	gsm	mpeg	adpcm	epic	gsm	mpeg
deadline1	0.6×10^5	0.8×10^7	0.9×10^8	0.3×10^9	0.05s	22s	5m5s	12m
deadline2	3.0×10^5	2.9×10^7	2.1×10^8	1.0×10^9	0.74s	88s	12m41s	41m
deadline3	4.0×10^5	5.1×10^7	2.5×10^8	2.1×10^9	1.06s	165s	11m15s	94m
deadline4	3.0×10^5	2.4×10^7	1.8×10^8	1.7×10^9	0.76s	76s	7m5s	75m
deadline5	1.9×10^5	1.7×10^7	1.5×10^8	1.6×10^9	0.39s	54s	5m35s	69m
deadline6	0.9×10^5	0.9×10^7	0.6×10^8	0.7×10^9	0.14s	27s	2m19s	25m

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

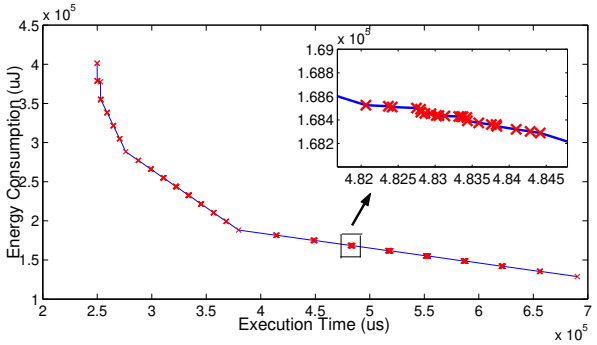

Figure 4: The energy-delay relationship for all nodes at the same level.

in the worst case for the optimal algorithm. We are looking for a mechanism to further shrink the search scope in the solution space by removing unpromising nodes.

Figure 4 shows the energy consumption and execution time of nodes at the same level for benchmark mpeg 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. If 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 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 $nbins$ nodes at each level. For procedure BRANCH-PRUNE, the first FOR loop needs $nbins * N * C_1$ steps in the worst case. The second FOR loop needs $nbins * N * C_2$ steps in total. Thus the total cost for procedure BRANCH-PRUNE is $nbins * N * (C_1 + C_2)$. Procedure SELECT needs $nbins * N * C_3 + nbins * N * C_4$. Therefore, the total cost for building nodes for one level is $nbins * N * (C_1 + C_2 + C_3 + C_4)$ 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 $M * nbins * N * C_{level}$, which is linear in M when $nbins$ and N are fixed.

Figure 5 shows the number of generated nodes for four benchmarks using six different deadlines for 100 bins. The node number is normalized to the number of nodes generated by the exact 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 100 bins, i.e. at most 100 nodes at each depth, the results from the heuristic algorithm are very close to the optimal results for adpcm, epic, gsm and mpeg. The energy difference is less than 1% for all benchmarks.

It is not surprising that the heuristic algorithm produces near-optimal results. As shown in Figure 4, there exist many near-

Algorithm 1 the Heuristic Algorithm

procedure BRANCH-PRUNE($SET(x_1,..,x_{i-1}),,E(i),T(i)$)
  for each $(x_1,..,x_{i-1}) \in SET$ **do**
    for $j \leftarrow 1$ to N **do** ▷ Worse-case cost for one iteration C_1
      $SRT[i] \leftarrow SRT[i-1] - T[i,N]$
      $t(x_1,..,x_i) \leftarrow t(x_1,..,x_{i-1}) + T(i,j)$
      $e(x_1,..,x_i) \leftarrow e(x_1,..,x_{i-1}) + E(i,j)$
      if $PRUNE_{DEADLINE}(x_1,..,x_i) == False$ **then** ▷ Pruning Condition 1
        Insert $(x_1,..,x_i)$ in $SET(x_1,..,x_i)$ based on energy
      end if
    end for
  end for
  for each $(x_1,..,x_i) \in SET(x_1,..,x_i)$ **do** ▷ Worse-case cost for each iteration C_2
    if $PRUNE_{OPTIMALITY}(x_1,..,x_i) == TRUE$ **then** ▷ Pruning Condition 2
      Remove $(x_1,..,x_i)$ from $SET(x_1,..,x_i)$
    end if
  end for
end procedure

procedure SELECT($SET, nbins$)
  distribute nodes $(x_1,..,x_i) \in SET$ into $nbins$ bins ▷ Cost $Length(SET) * C_3$
  for $i \leftarrow 1$ to $nbins$ **do** ▷ Worst-case cost for one iteration C_4
    $tran \leftarrow$ the nodes with the lowest transition counts in the bin
    $min \leftarrow$ the node with the lowest energy-delay product in $tran$
    Remove nodes other than min, the first node and the last node in the bin
  end for
end procedure

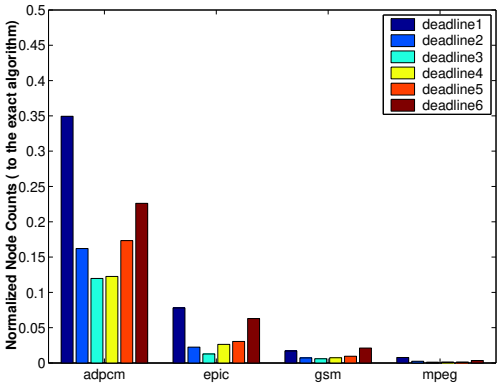

Figure 5: The number of total nodes generated by the heuristic for MediaBench benchmarks using 100 bins. The numbers are normalized to the nodes generated by the exact algorithm.

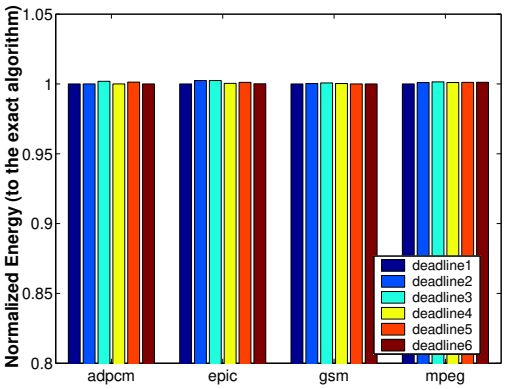

Figure 6: The energy consumption by the heuristic using 100 bins for MediaBench benchmarks. Energy is normalized to energy of the exponentially-complex exact algorithm.

optimal solutions. This is because many scaling units are similar in terms of the scalability of the execution time and energy with respect to frequency. Suppose that at one step, the node leading to the optimal solution is removed and another node with smaller energy-delay product is kept. If we define error to be the energy difference between node leading to the optimal solution and the chosen node, then error is introduced here. However, error is not cumulative with the growth of levels. For example, if the execution time of the chosen node is less than the optimal node (the energy consumption of the chosen node is higher than the optimal node), then a short period of slack is introduced. This period of slack can be reclaimed by slowing down some scaling units later on, which reduces the energy difference and thus the error. For large programs, this error adjustment occurs more often. The worst case is the chosen node leads to infeasible solutions that exceeds the deadline. In that case, the solution generated from the node in the neighboring bin with higher energy (shorter execution time) will be chosen. However, this error is still bounded by the energy difference between bins.

We show the runtime of MediaBench [9] benchmarks using heuristic algorithm with 100 bins in Table 4. The heuristic algorithm takes significantly less time than the exact algorithm. For benchmark mpeg, the runtime is only about 2 seconds instead of hours using the exponentially-complex exact algorithm. The speedup is up to 1000X while the energy difference between the heuristic and the exact algorithm is less than 0.5%. Also the runtime differences between different deadlines are not as dramatic as the exact case.

5. COMPARISON OF BOUNDS

In this section, we compare the lower bounds of energy consumptions using the optimistic analytical model from [17] 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 [17]. Six different deadlines are used from tight (deadline1) to loose (deadline6). The energy results are shown in 7. The energy is normalized to the energy using the best single frequency (the lowest frequency that can meet the deadline).

deadline	adpcm	epic	gsm	mpeg
1	0.01s	0.48s	1.30s	2.06s
2	0.04s	0.56s	1.35s	2.27s
3	0.04s	0.58s	1.36s	2.20s
4	0.04	0.53s	1.25s	2.27s
5	0.03s	0.44s	1.21s	1.95s
6	0.02s	0.43s	1.11s	1.97s

Table 4: Runtime for MediaBench benchmarks using the heuristic with 100 bins.

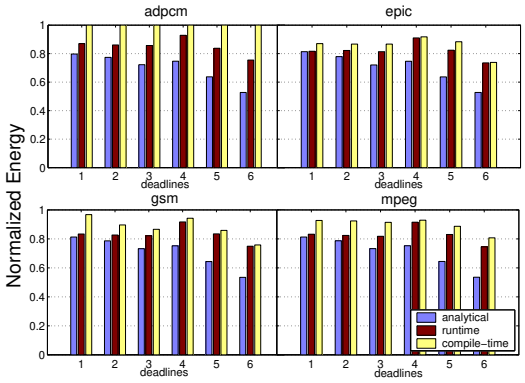

Figure 7: The minimum energy consumption predicted by an ideal analytical model, the lower bounds of energy consumption from runtime DVFS and the actual energy consumption achieved by a compile-time DVFS policy for six deadlines. Energy consumption is normalized to the best single frequency.

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 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 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 these upper bounds.

7. REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000.

[2] 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.

[3] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., July 1996.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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