# Efficient Behavior-driven Runtime Dynamic Voltage Scaling Policies

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

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

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

## ABSTRACT

Power consumption has long been a limiting factor in microprocessor design. In seeking energy efficiency solutions, dynamic voltage/frequency scaling (DVFS), a technique to vary voltage/frequency on the fly, has emerged as a powerful and practical power/energy reduction technique that exploits computation slack due to relaxed deadlines and memory accesses. DVFS has been implemented in some modern processors such as Intel XScale and Transmeta Crusoe. Hence the bulk of research efforts have been devoted to developing policies to detect slack and pick appropriate V/f assignments such that the energy is minimized while meeting performance requirements. Since slack is a product of memory accesses and relaxed deadlines, the number of instances and the duration of available slack are highly dependent on the runtime program behavior. Runtime DVFS policies must take into consideration program characteristics in order to achieve significant energy savings. In this paper, we characterize program behavior and classify programs in terms of the memory access behavior. We propose a runtime DVFS policy that takes into consideration the characteristics of program behavior for each category. Then we examine the efficiency of the proposed DVFS policies by comparing with previously derived upper bounds of energy savings. Results show that the proposed runtime DVFS policies approach the upper bounds of energy savings in most cases.

**Categories and Subject Descriptors:** D.4.7 [Operating System]: Organization and Design–*Real-time Systems and Embedded Systems*

**General Terms:** Design, Experimentation

**Keywords:** Runtime Dynamic Voltage Scaling, Low Power

## 1. INTRODUCTION

High system power/energy consumption has long been 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 [11], AMD mobile K6 Plus [1] and Transmeta Crusoe [21].

Various policies have been proposed to use DVFS to reduce energy consumption. These policies can be classified as compile-time policies [24, 10] and runtime policies [14, 13, 17, 22, 18, 23, 6] based on when the decisions to switch voltage/frequency are made. Runtime DVFS policies have drawn more research attention because of the ability to reduce energy assumption in response to variations in workload.

Any runtime DVFS policy consists of two important elements:

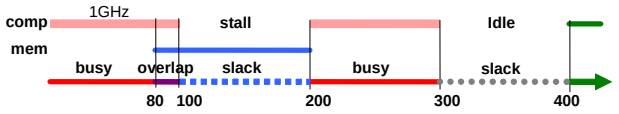
Scaling points: Scaling points are positions where voltage/frequency scaling can occur. They are signaled by a set of events such as timer interrupts, cache misses, task arrivals and completions. The piece of code enclosed by two scaling points is referred to as a scaling unit. All dynamic instructions within a scaling unit are assigned to run at the same voltage/frequency.

Scaling criteria: Scaling criteria determine the voltage/frequency level for the next scaling unit at a scaling point. It is usually composed of an off-line analysis and an online algorithm. The off-line analysis gathers information about the programs that will be used by the online algorithm to make quick DVFS decisions.

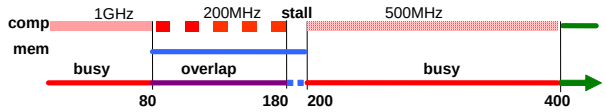
The goal of a runtime DVS policy is to exploit slack in the workload with an allocated time slot such that the energy consumption is minimized while completing within the allocated time slot. Depending on the types of the scaling points, DVFS policies can be classified as interval-based policies (timer interrupts) [14, 22, 18, 6], microarchitecture-based policies (cache misses and performance counters) [8, 16] and task-based policies (task arrivals and completions) [15, 13, 26, 19, 17, 2]. For interval-based DVS policies [22, 18], the workload of the next interval is assumed to remain the same as the previous interval and the v/f for the next interval is calculated based on the previous interval's workload information. For task-based policies, worst-case execution times of tasks are used to get the v/f setting for the next task. In [16], the supply voltage/frequency of the processor scales down on cache misses. In [6, 23], algorithms use performance counters (performance monitoring unit) to decide the new voltage/frequency.

In this paper, we will investigate the sources of slack. Slack is product of memory accesses and relaxed deadlines. While slack due to relaxed deadlines has been fully exploited, the slack due to memory accesses has drawn lots of research attention [10, 6, 23]. Hsu and kremer [10] proposed an algorithm to identify memory-bound regions and assign V/f settings at compile time. Choi [6] *et al.* used the ratio of off-chip access to on-chip computation times to direct DVFS. Weiser *et al.* [23] suggested to use memory requests per cycle and instructions per cycle to determine the appropriate V/f levels.


*CODES+ISSS'05,* Sept. 19–21, 2005, Jersey City, New Jersey, USA.




(a) Breakdown of execution time without DVFS. Slack is generated due to memory access and relaxed deadline




(b) Breakdown of execution time using DVFS. The duration of slack due to memory access is reduced while slack due to relaxed deadline is eliminated.

Figure 1: The breakdown of execution time before and after using DVFS.

We classify programs into memory-bound programs, computation-dominant programs and compound programs based on the characteristics of slack. We discuss the appropriate V/f assignments for each category. Then we propose a behavior-aware DVFS policy to approximate the maximum energy savings.

The rest of paper is organized as follows: Section 2 describes the sources of slack and how energy consumption can be reduced by using DVFS. Section 3 categorizes the programs based on the characteristics of slack and discuss the corresponding appropriate V/f assignments. Section 4 proposes a DVFS policy to approximate the upper bounds of energy savings. Section 5 presents the experimental results. Section 6 summarizes the contributions of our work.

## 2. MOTIVATION

When executing a program, the status of the CPU falls into one of the three categories: busy, stall and idle. Normally the CPU is busy executing instructions. However, due to the gap between CPU and off-chip device speeds, the CPU might stall waiting for off-chip services such as fetching data from memory. Note that memory in this paper refers to the off-chip memory. If the performance requirement, expressed as the deadline, is more relaxed than the actual execution time of the program and there is no runnable program available at the completion time, the CPU enters idle status. Figure 1(a) illustrates an execution trace on an out-of-order processor with a memory latency of 120ns. The CPU is busy during the first 80ns. At 80ns, a memory access is requested. The CPU spends another 20ns processing the independent operations that can overlap with the memory operation and then stalls waiting for the completion of the memory operation. At 200ns, the memory operation completes and the CPU continues with normal computation operations. The program finishes at 300ns. Because the next task arrives at 400ns, the CPU idles for 100ns. Correspondingly, the execution time can be broken down to CPU busy, memory stall and idle time. CPU busy time is further broken into only CPU busy and overlap time where the CPU is busy while the memory requests are being served.

We refer to the period of time when CPU stalls or idles as slack. Two instances of slack are produced in this example. One instance of slack is due to memory access and the other slack is due to the relaxed deadline. The existence of slack indicates that peak performance is actually not needed. DVFS is able to reduce energy consumption in this case by slowing down the CPU clock speed during the slack periods. For example, in Figure 1(a), we can slow down the clock speed to 200MHz at 80ns. Then the overlapped operations take longer to complete. Since the memory access still takes 120ns to complete, the stall time is reduced. Energy savings here are two-fold. The overlapped operations are executed at a lower frequency (200MHz in the example) and thus consume less energy than the no-DVFS case. Also, the energy consumption during stall time is reduced because of the shortened time and low voltage/frequency. After 200ns, we set the frequency to 500MHz. The program finishes at 400ns and thus the idle time is completely eliminated. Again, energy savings are twofold. Energy consumption during idle time is eliminated because the idle time is eliminated. At the same time, instructions are executed at 500MHz instead of 1GHz. The energy consumption of those computations is thus reduced. The execution trace using DVFS is shown in Figure 1(b). The program finishes sharply at the deadline 400ns, which means this set of V/f assignments also satisfy the performance requirement.

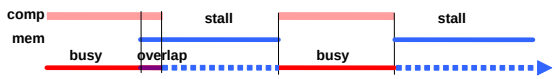
In the above example, we see the ability of DVFS to achieve energy savings. However, the amount of energy savings depends on the runtime DVFS policy's ability to identify slack and to assign appropriate V/f such that energy is reduced while meeting the deadlines.

Considering that slack due to relaxed deadlines is a single period of time and has been fully exploited by task-based policies, we will focus on the slack due to memory accesses. The number of slack instances and the duration of the slack period due to memory accesses are highly dependent on the program behavior, or specifically memory usage. We will start by categorizing program behavior based on memory resource usage. Then we will examine the optimal or near-optimal V/f assignments for each category based on the characteristics of slack.

## 3. V/F ASSIGNMENTS BASED ON CHARACTERISTICS OF SLACK

Consider a run of a program using a given data input. The execution trace of instructions is sliced into many scaling units by scaling points. A scaling unit can be either computation-dominant or memory-bound depending on the memory resource usage. If the majority of operations performed within the scaling unit are computation operations, the scaling unit is computation-dominant. If most of the execution time of the scaling unit is spent waiting for memory operations, the scaling unit is memory-bound. The percentage of computation-dominant scaling units and memory-bounds scaling units as well as the the way those two different classes of scaling units mix will affect the V/f assignments. We will discuss three different compositions: mainly computation-dominant scaling units; mainly memory-bounds scaling units; and a mix of computation-dominant scaling units and memory-bound scaling units. Programs with these compositions are referred to as computation-dominant programs, memory-bound programs and compound programs, respectively.

With the complete knowledge of the program, we can find the V/f assignments that minimize the energy consumption or give near-optimal energy consumption. We proposed an exact algorithm to provide the lower bounds of energy consumption in our previous work [25]. Considering a sequence of scaling units labelled from 1 to $M$ running on a DVFS-enabled microprocessor with $N$ discrete voltage/frequency levels, we profile the execution trace on a per frequency base for each scaling unit. The algorithm takes the profiles and searches the solution space in a breadth-first order until an optimal solution ($x_1$,$x_2$,...,$x_M$) has been found and confirmed. Basically, it branches over V/f levels to generate all feasible partial V/f assignments along with the input sequence. First, it 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. Branching from a certain partial solution will be stopped if the deadline cannot be met using that partial solution or there is another partial solution that uses less time and consumes




(a) Program where memory operations alternate with computation operations.


(b) Program where memory operations are clustered.

Figure 2: Programs with the same number of memory accesses but different memory access patterns.

less energy. Since scaling units can be made very fine, this method determines the upper bound on energy savings over any possible DVFS policy that can be applied for this program trace.

Next, we will discuss the optimal or near-optimal V/f assignments for three different categories of programs observed using this bound analysis.

## 3.1 V/f assignments for Computation-dominant Programs

Computation-dominant programs have the following characteristics:

1. The amount of slack produced by memory accesses is negligible.
2. The execution time can be represented by $N_{total}/f$, which scales linearly with CPU speed.

Because there is no significant amount of slack generated during memory accesses, slack due to relaxed deadlines is the only slack to be considered. Furthermore, because the execution time of all operations scales linearly with CPU speed, the number of execution cycles remains constant as the supply voltage/frequency changes. In this case, there exist optimal V/f assignments as shown in Ishihara's work [12]. In general, at most two voltages are needed to minimize the energy consumption. For example, if the deadline sits between the total execution time running the program at $f_1$ and total execution time running the program at $f_2$, then only $f_1$ and $f_2$ are needed and at most one switching is needed to minimize the energy regardless of the total available voltage levels. To achieve the minimum energy, the program will run the first $n_1$ cycles using $f_1$ and the remaining cycles using $f_2$ if the switching overheads do not offset the energy savings using DVFS. $n_1$ is determined by:

$$n_1/f_1 + (N_{total} - n_1)/f_2 + S_t(f_1, f_2) = deadline$$

where $N_{total}$ is the number of total execution cycles and $S_t$ is the switching time cost from $f_1$ to $f_2$.

## 3.2 V/f Assignments for Memory-bound Programs

Memory-bound programs consist mostly of memory-bound scaling units that spend significant amount of time waiting for the completion of memory operations. Due to the widening gap between CPU speed and memory speed, significant slack is available. The execution time can be expressed as $N_{cpu}/f + t_{slack}$ where $N_{cpu}$ is the number of CPU busy execution cycles that scale with CPU speed and $t_{slack}$ is the time spent on memory stall cycles that does not scale with CPU speed.

However, since most scaling units are memory-bound, slack instances are scattered across all scaling units, which means all scaling units are equally likely to be scaled down. Therefore, simple V/f assignments similar to computation-dominant programs are reasonably efficient. At most two frequencies that sandwich the deadline will be used. The program will run first $i_1$ CPU busy cycles at $f_1$. Then it switches to $f_2$ and runs the remaining instructions at $f_2$ if the energy savings using $f_2$ will not be offset by the switching overheads. The value of $i_1$ is determined by equation

$$i_1/f_1 + (N_{cpu} - i_1)/f_2 = (deadline - T_{mem})$$

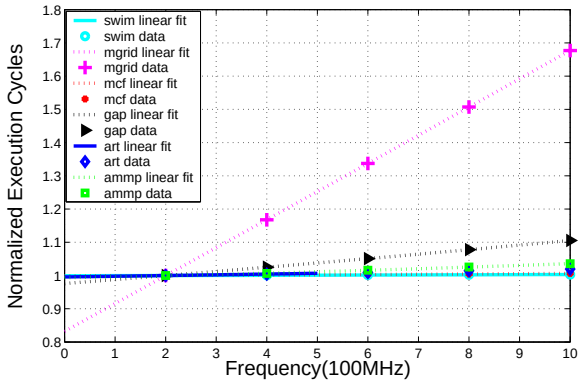



Figure 3: The number of execution cycles for benchmarks from SPEC2000 at different frequencies. The number is normalized to the number of execution cycles at 200MHz.

There are some differences between the V/f assignments here and the V/f assignments for computation-dominant case. We use the number of CPU busy cycles instead of the number of total execution cycles in the equation. The number of total execution cycles generally increases with frequency because memory speed does not scale with CPU speed.

This simplified V/f assignments might not be optimal but it gives near-optimal results.

## 3.3 V/f assignments for Compound Programs

V/f assignments for programs with mixed computation-dominant scaling units and memory-bound scaling units are complex. Ideally slowing down the CPU speed during those memory-bound scaling units will reduce energy consumption with little impact on performance. However, there are time and energy overheads associated with switching. For example, the switching time cost for Intel XScale processors is around $10\mu s$, which amounts to 10000 cycles at 1GHz. Considering the switching costs, switching cannot occur too frequently or the switching costs may offset the energy savings from DVFS.

The composition of computation-dominant scaling units and memory-bound scaling units has a great impact on V/f assignments. Figure 2 shows four scaling units with two different compositions. The first example's memory-bound scaling units alternate with computation-dominant scaling units while the second example's memory-bound scaling units are clustered. Though the two examples have the same number of each kind of scaling units, for example (b), DVFS will be more beneficial. This determines the scaling priority in the V/f assignments.

High Priority: clustered memory-bound scaling units

Medium Priority: mixed memory-bound scaling units and computation-dominant scaling units

Low Priority: clustered computation-dominant scaling units.

## 3.4 Summary

The V/f assignments depend on the characteristics of program behavior. For computation-dominant programs where the vast majority are computation-dominant scaling units and memory-bound programs that consist of a majority of memory-dominant scaling units, two frequencies that sandwich the deadline are needed in the V/f assignment. Program runs the first scaling units at one frequency, then switches to the other frequency and runs the remaining of the program at that frequency. For other programs with considerable number of memory-dominant scaling units and computation-dominant scaling units, we favor scaling down clustered memory-dominant scaling units.

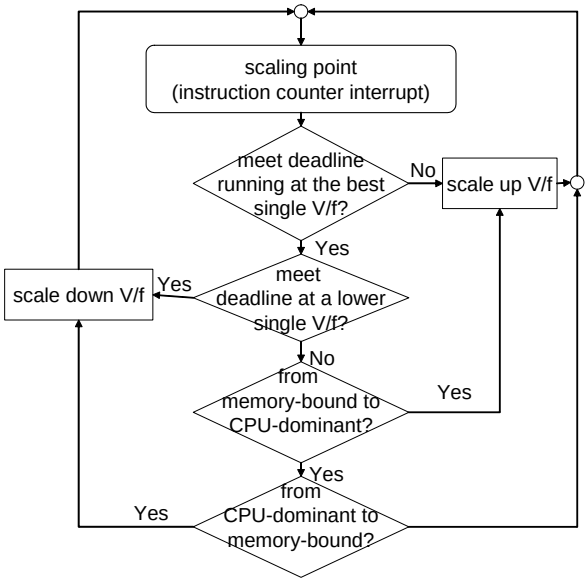



Figure 4: The flowchart of the proposed runtime DVFS policy.

# 4. PROPOSED RUNTIME DVFS POLICY

A runtime DVFS policy is composed of scaling points and scaling criteria. We will assume the scaling points are fixed and focus on the scaling criteria in this section.

Based on the previous discussion, the DVFS policy should be able to identify computation-dominant and memory-bound scaling units at runtime. Many modern microprocessors provide performance counters that can be used to generate the information needed by DVFS [6, 23]. Therefore, we assume information such as the memory stall ratio for a scaling unit is available by monitoring the number of memory accesses occurring within a scaling unit. Memory-bound scaling units and computation-dominant scaling units can be identified in terms of the memory stall ratio. Note that IPC (instruction per cycle) or CPI (cycle per instruction) cannot be used to distinguish memory-bound scaling units and computation-dominant units due to the existence of long latency instructions.

The DVFS policy also requires certain statistical information to calculate the program's execution time. The number of total execution cycles, which is assumed to be constant, is commonly used. While this assumption is valid for computation-dominant programs, it does not work for programs with a number of memory accesses. Figure 3 shows the number of execution cycles at different frequencies for six programs from SPEC2000 [20]. The numbers are normalized to the total execution cycles at 200MHz. Benchmark mgrid has intensive memory accesses and thus the number of execution cycles increases dramatically as the frequency increases. Benchmark gap has mediocre memory accesses, so the slope is not sharp. Other benchmarks are computation-dominant and the number of execution cycles does not vary much. We notice that the linear fitting curve fits the data very well. Instead of using one number of execution cycles, we will use the linear function to calculate the execution cycles at different frequencies accurately.

The efficiency of the DVFS policy lies in its ability to decide when to switch. A runtime DVFS policy only has the knowledge of the past. The switching decisions are made based on the prediction of the future. For simplicity, we assume the behavior of the next scaling unit is the combination of passed $j$ scaling units where the current scaling unit has more weight. The simple case is to assume the next scaling unit is the same as the current scaling unit.

With information ready, the simplified scaling criteria is defined as shown in Figure 4. Before running the program, the operating system will use the available information to compute the best single frequency. The best single frequency is the lowest frequency at which the program will meet the deadline. At each scaling point, the policy will check if the deadline will be met by running the remaining program at the best single frequency. If the deadline cannot be met, then the V/f is scaled up. Next, the policy will check if the deadline will be met by running the remaining program at the frequency that is the immediate lower neighbor of the best single frequency. If the deadline can still be satisfied, the V/f level will switch to the lower V/f level. Otherwise, the operating system will read performance counters to predict the program behavior of the next scaling unit. If the next scaling unit is predicted to be memory-bound while the previous scaling units are computation dominant, the V/f will be scaled down. For each V/f switch, the associated time and energy overheads will be counted.

To summarize, this runtime DVFS policy switches V/f when program behavior changes and when deadline cannot be met.

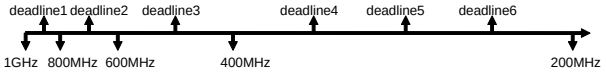



Figure 5: The positions of deadlines with respect to the execution times using single frequency.

| Parameter | Value |
|---|---|
| RUU size | 64 instructions |
| LSQ size | 32 instructions |
| Fetch Queue size | 8 instructions |
| Fetch width | 8 instructions/cycle |
| Decode width | 8 instructions/cycle |
| Issue width | 8 instructions/cycle |
| Commit width | 8 instructions/cycle |
| Functional Units | 4 Integer ALUs |
| | 1 integer multiply/divide |
| | 1 FP add, 1 FP multiply |
| | 1 FP divide/sqrt |
| Branch Predictor | Combined, bimodal 2K table |
| | 2-level 1K table, 8bit history |
| | 1K chooser |
| BTB | 512-entry, 4 way |
| L1 data-cache | 64K, 4-way(LRU) |
| | 32B blocks, 1 cycle latency |
| L1 instruction-cache | 32K, 4-way(LRU) |
| | 32B blocks, 1 cycle latency |
| L2 | Unified, 512K, 4-way(LRU) |
| | 64B blocks, 8-cycle latency |
| TLBs | 32-entry, 4096-byte page |

Table 1: Configuration parameters for CPU simulation.

# 5. EXPERIMENTAL RESULTS

## 5.1 Experiment settings

We use SimpleScalar [5] with Wattch [3] to simulate an out-of-order DVFS-enabled processor as shown in Table 1. The processor supports five voltage levels: 0.7V/200MHz, 0.99V/400MHz, 1.3V/600MHz, 1.65V/800MHz and 2.05V/1GHz. The switching costs are calculated using the following equations taken from [4] that are considered to be an accurate modeling of these switching costs.:

$$S_E = (1-u) * c * |v_i^2 - v_j^2| \quad (1)$$

$$S_T = \frac{2*c}{I_{MAX}} |v_i - v_j|$$

where $c$ is the capacitance of the voltage regulator, $u$ is the energy efficiency of the power regulator. $I_{MAX}$ is the maximum allowed current. In the experiments, we use $c = 10\mu f$, $u = 90\%$ and $I_{MAX} = 1A$, which will yield $12\mu$s switching time and $1.2\mu$J switching energy cost for a transition from 600Mhz/1.3V to 200Mhz/0.7V. Values are picked to approximate XScale's [11] switching costs.

Benchmarks are chosen from SPEC2000 [20]. These benchmarks represent a wide range of memory behavior from computation-dominant to memory-bound and are classified into three groups as

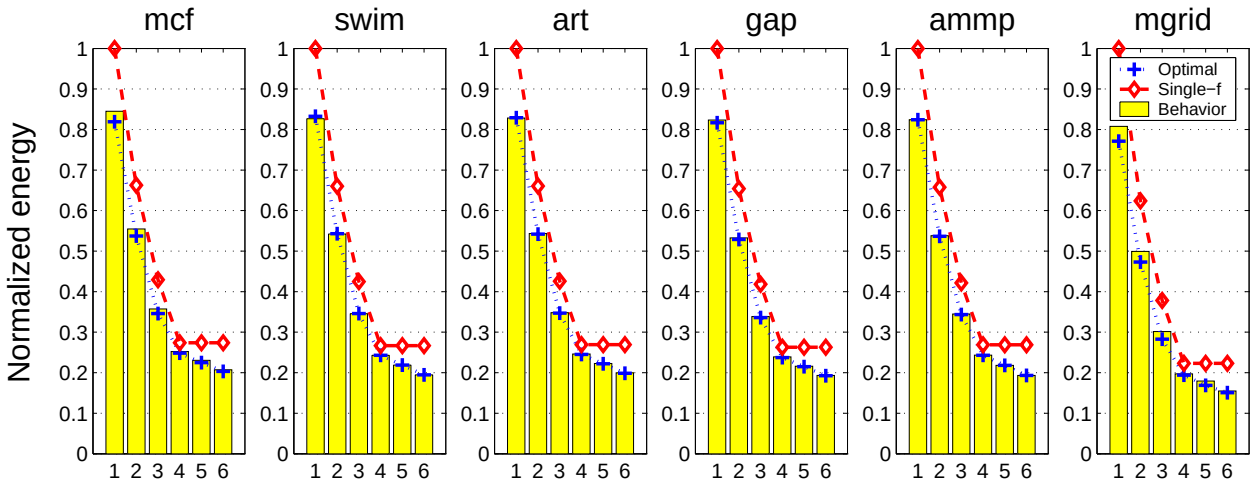



Figure 6: The normalized energy consumption of benchmarks from SPEC2000 using DVFS for six different deadlines. Energy consumption is normalized to energy using the highest frequency (1GHz). The bar represents the normalized energy using our proposed DVFS. The dotted line represents the normalized optimal energy consumption. The dashed line represents the normalized energy consumption using the best single frequency.

| Class | Benchmarks |
|---|---|
| Computation-dominant | mcf, swim, art |
| compound | gap |
| Memory-bound | ammp, mgrid |

Table 2: Characteristics of benchmarks from SPEC2000.

shown in Table 2. For benchmark mcf, swim and art, the percentage of execution time spent in memory stalls is less than 4% at 1GHz. The percentage of execution time is 11% for benchmark gap, 23% for ammp and 40% for mcf at 1GHz. For all SPEC2000 benchmarks, we run the first two billion instructions only.

For every benchmark, we consider six different deadlines from tight to loose. The positions of deadlines with respect to the execution times using single frequency are shown in Figure 5. Deadline1 is the average of the execution times using 1GHz and 800GHz. Deadline2 is the average of the execution times using 800MHz and 600MHz. Deadline 4 to 6 sit evenly between the execution times using 400MHz and 200MHz.

## 5.2 Energy Results

In the experiment, scaling points occur every $10^6$ dynamic instructions. This granularity is picked based on our previous work [25] which shows that there is no need to use granularity smaller than $10^6$ instructions.

Figure 6 shows the energy consumption using the proposed DVFS policy for all benchmarks. Energy consumption is normalized to the energy consumption using the highest frequency (without DVFS). The bar shows the normalized energy using our proposed DVFS. The dotted line shows the normalized optimal energy consumption obtained from our bound analysis algorithm [25]. The dashed line shows the minimum energy that can be achieved if only one frequency is allowed for one program. The proposed DVFS policy is effective at reducing energy. First, it outperforms the single-frequency DVFS policy by up to 18%, which demonstrates the necessity of intra-program DVFS. Second, it comes close to the optimal energy consumption in most cases. For benchmarks mcf and mgrid, the proposed DVFS consumes more energy. However, the difference is less than 4%. At deadline1, benchmark swim consumes a little less energy using the proposed behavior-driven DVFS policy than the optimal energy consumption at the price of exceeding deadline. From mcf to mgrid, the trend of energy consumption is decreasing, which is consistent with our intuition that the energy can be reduced further by exploiting energy waste due to slack produced in memory accesses. One exception is that the energy consumption of benchmark gap is slightly lower than the benchmark ammp that has higher percentage of memory stall time. This is because the memory-bound scaling units are clustered in benchmark gap while in benchmark ammp, the majority of scaling units are memory-bound with equal memory stall percentage.

Figure 7 shows the execution time using the behavior-driven DVFS policy. The execution time is normalized to the deadline. For most benchmarks, the execution time is very close to the required deadline. In some cases, the execution time exceeds the deadline slightly. However, the percentage is less than 2%. This is because the proposed DVFS policy is based on the prediction of future memory behavior to determine the V/f for the next scaling unit.

# 6. CONCLUSION

In this paper, we focus on the energy waste due to slack generated at memory accesses and proposed a behavior-driven runtime DVFS policy to reduce energy. The policy, though simple, has demonstrated the ability to reduce energy consumption for memory-bound programs, computation-dominant programs and compound programs. The efficiency is shown by comparing the results with the minimum energy consumption obtained from our bound analysis algorithm. There are two observations from our results. First, slack due to memory accesses should be taken into to further reduce energy. This is very important for many database programs where almost two-thirds of execution time is spent in off-chip accesses [9, 7]. Second, a runtime behavior-aware DVFS policy is able to reduce the energy efficiently. Thus DVFS policies proposed in [6, 23] will achieve similar energy savings.

# 7. REFERENCES

[1] Advanced Micro Devices Corporation. AMD-K6 processor mobile tech docs, 2002. http://www.amd.com.

[2] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10518, Washington, DC, USA, 2004. IEEE Computer Society.

[3] 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.

[4] 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.

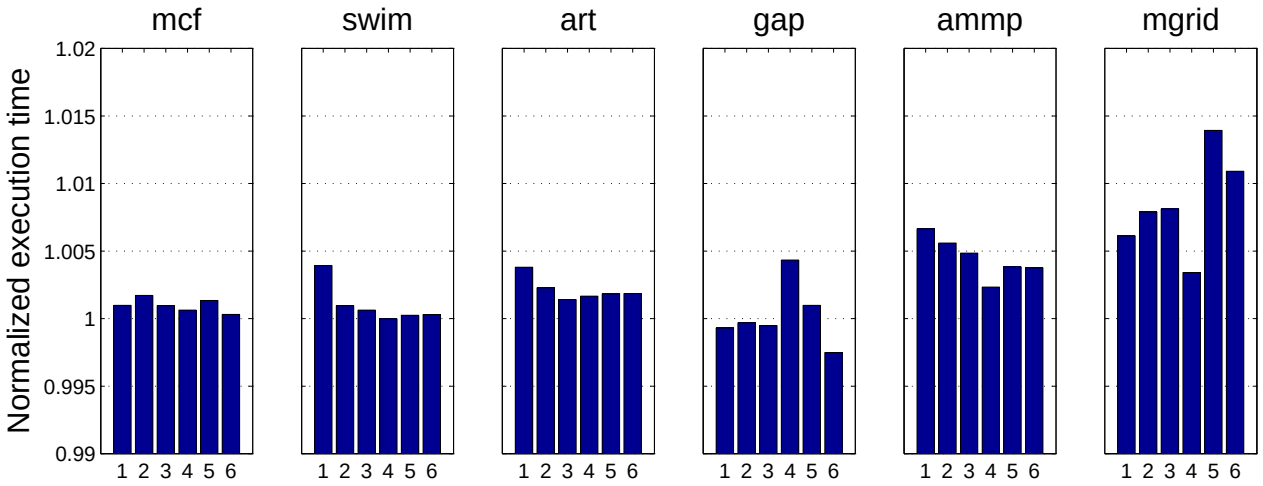



Figure 7: Normalized execution time using the proposed DVFS policy for six different deadlines from loose to tight. Execution time is normalized to the corresponding deadline.


[5] 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.

[6] 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. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 18–28, Jan 2005.

[7] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In *Proceedings of the 31st annual international symposium on Computer architecture (ISCA04)*, page 76. IEEE Computer Society, 2004.

[8] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Workshop on Complexity-Effective Design*, June 2000.

[9] R. Hankins, T. Diep, M. Annavaram, B. Hirano, H. Eric, H. Nueckel, and J. Shen. Scaling and characterizing database workloads: Bridging the gap between research and practice. In *Proceedings of the 36th International Symposium on Microarchitecture*, page 76, Washington, DC, USA, December 2003. IEEE Computer Society.

[10] C. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proceedings of ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI'03)*, June 2003.

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

[12] 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.

[13] 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.

[14] 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.

[15] J. Luo and N. K. Jha. Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems. In *Int. Conf. VLSI design*, Jan. 2003.

[16] D. Marculescu. On the use of microarchitecture-driven dynamic voltage scaling. In *Workshop on Complexity-Effective Design*, June 2000.

[17] 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.

[18] 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.

[19] V. Swaminathan, C. Schweizer, K. Chakrabarty, and A. Patel. Experiences in implementing an energy-driven task scheduler in rt-linux. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02).*, page 229, 2002.

[20] The Standard Performance Evaluation Corporation. WWW Site. http://www.specbench.org, 2000.

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

[22] 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.

[23] 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.

[24] 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.

[25] F. Xie, M. Martonosi, and S. Malik. Bounds on power savings using runtime dynamic voltage/frequency scaling: An exact algorithm and a linear-time heuristic approximation. In *International Symposium on Low Power Electronics and Design (IS LPED-05)*, August 2005.

[26] Y. Zhang, X. Hu, and D. Chen. Energy minimization of real-time tasks on variable voltage processors with transition energy overhead. In *Proceedings of the ASP-DAC 2003 Design Automation Conference*, pages 65–70, 2003.