Memory Access Aware On-Line Voltage Control for Performance and Energy Optimization

Xi Chen*, Chi Xu†, and Robert P. Dick*

*EECS Department
University of Michigan
Ann Arbor, MI 48109
{chexi@, dickrp@eecs.}umich.edu

†ECE Department
Univeristy of Minnesota
Minneapolis, MN 55455
xuchi@umn.edu

***Abstract*—This paper describes an off-chip memory access-aware runtime DVFS control technique that minimizes energy consumption subject to constraints on application execution times. We consider application phases and the implications of changing cache miss rates on the ideal power control state. We first propose a two-stage DVFS algorithm based on formulating the throughput-constrained energy minimization problem as a multiple-choice knapsack problem (MCKP). This algorithm uses a power model that adapts to application phase changes by observing processor hardware performance counter values. The solutions it produces provide upper bounds on the energy savings achievable under a performance constraint. However, this algorithm assumes a priori (oracle or profiling-based) knowledge of application phase change behavior. To relax this assumption, we propose P-DVFS, an predictive DVFS algorithm for on-line minimization of energy consumption under a performance constraint without requiring a priori knowledge of an application's behavior. P-DVFS uses hardware performance counter based performance and power models. It predicts remaining execution time online in order to control voltage and frequency settings to optimize energy consumption and performance. The P-DVFS problem is formulated as a multiple-choice knapsack problem, which can be efficiently and optimally solved online. We evaluated P-DVFS using direct measurement of a real DVFS-equipped system. When bounding performance loss to at most 20% of that at the maximum frequency and voltage, P-DVFS leads to energy consumptions within 1.83% of the optimal solution for our problem instances on average with a maximum deviation of 4.83%. In addition to producing results approaching those of an oracle formulation, P-DVFS reduces power consumption for our problem instances by 9.93% on average, and up to 25.64%, compared with the most advanced related work.**

I. Introduction and Related Work

Energy consumption is important in both portable computer systems, due to its impact on battery lifespan, and high-performance stationary computers, due to its impact on energy and cooling costs. Prior work has considered minimizing processor energy consumption. Chang et al. proposed a dynamic programming energy minimization technique for multiple supply voltage scheduling in both pipelined and non-pipelined datapaths [1]. Zhang et al. developed a two-phase technique that integrates task assignment, task scheduling, and voltage selection for energy minimization [2]. Varatkar et al. proposed a communication-aware task scheduling and voltage selection algorithm to minimize the overall system energy consumption in a multiprocessor environment [3]. However, the goal of these techniques is to minimize energy without affecting performance; trade-offs between performance and energy consumption were not considered.

Other researchers have considered power management mechanisms that trade off performance and power consumption. One of the most promising of these is dynamic voltage and frequency scaling (DVFS). Two characteristics are important to DVFS control policies. First, a well-designed DVFS control policy must model and react to the dynamically changing trade-offs between application performance and power consumption. A reduction in processor voltage and frequency has very different energy and performance impacts on applications that are heavily accessing off-chip memory, and those that are consistently hitting in cache and therefore have performance constrained only by the current frequency of the processor. A well-designed DVFS policy must continuously monitor and adapt to the behavior of applications. Second, if a DVFS control policy is to guarantee that a particular application consistently runs with adequate performance, e.g., honoring an instruction throughput constraint, it should maximize energy consumption savings by predicting the distribution of future instructions among different memory access behaviors categories. This allows the control policy to increase processor voltage and frequency when the performance benefit per lost energy unit is the highest and reduce frequency and voltage when the energy benefit per lost performance unit is the highest.

A number of researchers have worked on DVFS-related control to optimize power and energy consumption. Isci et al. proposed a runtime phase monitoring and prediction technique to reduce power consumption using DVFS [4]. However, this technique does not bound performance degradation. Wu et al. proposed dynamic compiler driven DVFS for controlling microprocessor energy and performance [5]. However, their work requires changes to the underlying compilation infrastructure. In addition, their technique does not attempt to honor performance constraints. Liu et al. proposed a technique to optimize peak temperature subject to a real-time performance constraint using DVFS [6]. However, their assumption that the execution time of a task is inversely proportional to CPU frequency is correct only for systems in which all layers of the memory hierarchy operate at the same frequency, as we will demonstrate in Section II-A. The technique proposed by Choi et al. is the closest to ours [7]. The goal of their technique is to minimize energy consumption under a constraint on the total program execution time. Detailed comparisons with their work can be found in Section IV-B. Their DVFS policy considers the impact of application phases and off-chip memory accesses. However, it considers only immediate application behavior instead of adaptively controlling power state using predictions based on long-term behavior history.

Our work differs from prior work in the following main ways.

1) We propose a two-stage DVFS algorithm that allows us to formulate the throughput-constrained energy minimization problem as an MCKP problem, solve it optimally, and use the solution to guide online frequency and voltage control. This algorithm builds on an application phase-dependent power model, taking advantage of processor hardware performance counters. The solutions obtained using the two-stage algorithm determine the optimal energy savings under a performance degradation ratio, for our formulation and problem instances. However, it assumes access to oracle or profiling-

This work was supported in part by NSF under awards CCF-0702761, CNS-0347941, and CNS-0720691 and in part by SRC under award 2007-HJ-1593.

based information about application behavior. In the rest of the paper, we will use "optimal solution" in the context of our problem formulation when this does not introduce ambiguity.

2) We also propose P-DVFS, a predictive online DVFS algorithm that requires no a priori knowledge of application behavior. P-DVFS uses hardware performance counter based power and performance models to adapt to the behavior of running applications. It predicts remaining execution time online in order to control voltage and frequency to minimize energy consumption under application-level performance constraints. Like the two-stage oracle DVFS algorithm, P-DVFS is also formulated as a multiple-choice knapsack problem. This formulation permits rapid, optimal, on-line solution of real problem instances.

3) In contrast with all related work, except that of Choi et al. [7], we consider the dependence of the power consumption performance tradeoffs available via DVFS upon application memory access behavior, i.e., phase. By adapting to application phase, our technique supports more aggressive power management settings when they have the least negative performance impact. To this end, we describe a method of modeling the performance and power consumption of the processor using built-in hardware performance counters.

4) In contrast with all past work, our problem formulation supports application-level throughput requirement, not instantaneous instruction throughput requirement. This is supported by on-line monitoring of application behavior as well as prediction of application run times.

We evaluated P-DVFS via direct measurement during operation on a real system. When limiting performance loss to at most 20% of that possible at the maximum frequency and voltage, P-DVFS leads to energy savings within 1.83% of optimal on average with a maximum deviation of 4.83%, for our problem instances. It improves energy consumption by 9.80% on average, and up to 29.86%, compared to the most advanced related DVFS control technique. P-DVFS also reduces power consumption by up to 25.64% (9.93% on average) compared with the most advanced related work.

II. Motivation and Problem Formulation

In this section, we first describe how the trade-offs between performance and energy consumption change depending on application off-chip memory access behavior. We then present the problem formulation for energy minimization given a user-specified constraint on application execution time. Finally, we present a dynamic power state control policy that adjusts CPU frequency based on off-chip memory access patterns.

II.A. Performance and Energy Trade-Offs

The execution time of a task can be decomposed into on-chip and off-chip latencies. The latencies of on-chip components scale linearly with CPU frequency, because they share the same clock with the processor. In contrast, off-chip latencies, caused by accesses to off-chip resources such as main memory and disk, are independent of CPU frequency, because the off-chip resources have one or more separate clock.

The power consumption of a task can be divided into dynamic power and static power. Dynamic power consumption is caused by switching transistors charging and discharging capacitive loads. It generally scales superlinearly with CPU clock frequency [8]. Static power consumption is primarily due to gate and subthreshold leakage currents of transistors. It does not directly depend on CPU frequency but depends on the voltage. In general, reducing frequency and voltage reduces both dynamic and static power consumption.

Many modern processors support dynamic voltage and frequency scaling (DVFS) capability. The typical voltage change overhead for our evaluation platform is 50 µs. Given an application with some phases in which instruction throughput is limited largely by processor core performance and other phases in which instruction throughput is limited largely by (processor frequency independent) off-chip memory access latency, we can maximize energy consumption improvement and minimize performance overhead by using a low CPU frequency during memory-bound application phases and a high CPU frequency during core-bound application phases. What temporal granularity should this control use? The DVFS switching overhead of 50 µs (see Section IV) implies that adjustments should happen no more frequently than once every hundred microseconds, thus limiting overhead.

II.B. Problem Formulation

The performance-constrained energy minimization problem can be formulated as follows: Given that α is the user-specified performance degradation ratio relative to the maximum performance of a given task and T_{fmax} is the execution time of the task running at the highest frequency, find the optimal CPU frequency as a function of time t, such that the total energy consumption of the task is minimized and the actual execution time of the task subject to DVFS, is no larger than $(1+\alpha)T_{fmax}$. Note that this performance constraint is soft, i.e., it is highly desirable to meet it. However, violating the constraint does not mean failure: a cost function may be associated with the degree of constraint violation.

As indicated in Section II-A, the energy saving potential directly relates to the proportion of total execution time resulting from waiting for off-chip data access, which are primarily L2 cache misses in our experiments. We assume that each L2 cache miss takes the same amount of time. Hence, the number of L2 cache misses per instruction (MPI), is a good indicator of the potential for saving energy. Intuitively, it is beneficial to assign higher frequencies for intervals with low MPIs (to improve performance) and lower frequencies for intervals with high MPIs (to save energy).

In real operating systems, power control policies are usually implemented using adjustments at discrete time intervals. Discretized MPI values are used. We define a *control point* as a time at which control decisions are made and a *scaling point* as a time at which the CPU frequency is modified. The *control period* is the duration between two consecutive control points and the *scaling period* is the duration between two consecutive scaling points. Note that these periods need not be the same. In fact, it is reasonable to use a much larger control period than scaling period to minimize performance overhead incurred by the controller and use time multiplexing to emulate continuous DVFS within a control period.

Given an MPI distribution within a control period, S is the set of all MPI slots and F is the set of all available frequency levels. Our goal is to find the correct frequency level f_i for each slot $i \in S$ such that the total energy consumption E_{total} is minimized and the actual execution time T_{act} satisfies $T_{act} \leq (1+\alpha)T_{fmax}$. Therefore, assuming the distribution is independent of frequency, for each $i \in S$ with frequency f_i, given that $\text{SPI}_i(f_i)$ is the number of seconds per instruction at frequency f_i, $P_i(f_i)$ is the power consumption, and poi_i is the percentage of instruction associated with slot i, the objective function and the constraint can be expressed in terms of total number of instructions I_{total} and total energy consumption E_{total}, i.e.,

$$E_{total} = I_{total} \cdot \sum_{i \in S} P_i(f_i) \cdot poi_i \cdot \text{SPI}_i(f_i) \text{ and} \quad (1)$$

$$T_{act} \leq (1+\alpha)T_{fmax}. \quad (2)$$

The goal is to minimize E_{total} subject to Equation 2. Since the DVFS switching overhead ranges from 50 µs to 200 µs, the performance (or energy) overhead due to a frequency change is less than 0.7%, given a scaling period of 30 ms. Therefore, we ignore its impact in our

problem formulation. Note that $P_i(f_i)$ in Equation 1 depends on both the CPU frequency and application behavior, e.g., the number of last-level cache misses per second (see Section III-B).

III. System Modeling

In this section, we first explain our task performance and power models. We then formulate the energy minimization problem as a multiple-choice knapsack problem (MCKP) and solve it optimally, assuming knowledge of the average SPI at the maximum frequency (SPI_{fmax}) and the exact application MPI distribution. We then relax our assumptions and propose an execution time predictor that is accurate at the highest frequency. This allows us to formulate the online DVFS problem again as an MCKP, which can be solved efficiently on-line. Finally, we explain the software system architecture used to control DVFS in order to accurately adjust the trade-off between performance and energy consumption.

III.A. Performance Modeling

Equation 2 depends on a formula that accurately expresses the relationship between SPI, MPI, and CPU frequency. Intuitively, the amount of time consumed per instruction can also be decomposed into on-chip and off-chip latencies. On-chip latency is inversely proportional to frequency, while off-chip latency, captured by MPI, is independent of frequency. Prior work has reached the same conclusion [4]. SPI can be expressed as

$$\begin{aligned} \text{SPI}(\text{MPI}, f) &= c_1 \cdot \text{MPI} + c_2/f, \text{or equivalently,} && (3) \\ \text{CPI}(\text{MPI}) &= c_1 \cdot f \cdot \text{MPI} + c_2, && (4) \end{aligned}$$

where CPI is the number of cycles per instruction, f is the CPU frequency, and c_1 and c_2 are constants to be determined via fitting.

Most modern processors have built-in hardware performance counters that record information about architectural events, e.g., number of instructions retired and cache misses [9]. By gathering these two event counts, we can compute SPI and MPI during application execution. Therefore, given the last N data points reported by hardware performance counters, we can determine c_1 and c_2 can be determined using linear regression. The relevant formulæ follow.

$$c_1 = \frac{N \cdot (\sum_{i=1}^{N} x_i \cdot y_i) - (\sum_{i=1}^{N} x_i) \cdot (\sum_{i=1}^{N} y_i)}{N \cdot (\sum_{i=1}^{N} x_i^2) - (\sum_{i=1}^{N} x_i)^2} \text{ and} \quad (5)$$

$$c_2 = \left(\sum_{i=1}^{N} y_i - c_1 \cdot \sum_{i=1}^{N} x_i \right) / N, \quad (6)$$

where x_i denotes the product of MPI and CPU frequency for the ith data point and y_i represents the CPI for the ith data point. Note that N should be carefully chosen to capture changes in memory access pattern quickly and support accurate regression-based modeling. In our experiments, varying N between 10 and 50 has insignificant impact on energy consumption (a variation of 0.5% in total energy was observed). However, if N is smaller than 10, e.g., 4, we see an 4% energy consumption increase due to inaccuracies in the linear regression model. In our experiments, we set N to 20.

III.B. Power Modeling

Equation 1 indicates the necessity of having an accurate formula to describe the relationship between power consumption and MPI. Since an L2 cache misses are time consuming, the power consumption is higher for larger MPI values and smaller for lower MPI values. However, the power consumption also depends on other architectural events such as number of floating point instructions executed and number of L1 data cache accesses. We experimented with different combinations of hardware performance counter events and observed that following five were sufficient to permit accurate estimation of power consumption:

1) number of L1 data cache references per second (L1DPS),
2) number of L2 cache references per second (L2PS),
3) number of L2 cache misses per second (L2MPS),
4) number of floating point instructions executed per second (FPPS), and
5) number of branch instructions retired per second (BRPS).

As a first-order approximation, we assume each access to system components such as L1 caches and L2 cache consumes a fixed amount of energy. Therefore, the total power consumption depends linearly on these five events. In addition, the dynamic power consumption depends nonlinearly on CPU frequency [10]. Given that f is the CPU frequency, the power consumption can be estimated as follows:

$$\begin{aligned} P = {} & b_0 + b_1 \cdot \text{L1DPS} + b_2 \cdot \text{L2PS} + b_3 \cdot \text{L2MPS} + \\ & b_4 \cdot \text{FPPS} + b_5 \cdot \text{BRPS} + b_6 \cdot f^{1.5}, \end{aligned} \quad (7)$$

where $b_i, i = 0, \cdots, 6$ are task-specific constants that can be determined during pre-characterization. The frequency exponent of 1.5 was determined empirically. It is worth mentioning that b_0 accounts for system idle and leakage power. For example, the formula for the "mcf" benchmark (see Section IV) follows:

$$\begin{aligned} P = {} & 4.778 + 2.2864 \times 10^{-9} \cdot \text{L1DPS} + \\ & 6.517 \times 10^{-8} \cdot \text{L2PS} - 3.596 \times 10^{-7} \cdot \text{L2MPS} + \\ & 0.6342 \cdot \text{FPPS} - 3.136 \times 10^{-9} \cdot \text{BRPS} + 4.308 \cdot f^{1.5}. \end{aligned} \quad (8)$$

For all the benchmarks we evaluated, the application-dependent power models have an average error of 6.67% and a maximum error of 12.2% across all four CPU frequencies. Note that if the processor has built-in power sensors [11], the pre-characterization phase can be eliminated and the constants can be determined during execution using a regression-based approach such as that described in Section III-A.

III.C. Cost Minimization

This section describes formulation of the DVFS power management state control problem as a multiple-choice knapsack problem (MCKP). Given multiple sets, each containing multiple items, each of which is associated with a profit and a weight, MCKP requires the selection of one item from each set. The selection is optimal when the total profit is maximized and the total weight of the selected items is below a constraint. The DVFS problem instance can be converted into an MCKP instance by treating each potential frequency level as an item. The weight of the item is the expected throughput at the associated frequency level. The profit of the item is the associated reduction in expected energy consumption compared to the energy at the highest frequency. Note that, depending on whether we have a priori knowledge SPI_{fmax} and the MPI distribution throughout program execution, the DVFS problem instance can be formulated as different MCKP instances, as explained in Section III-C2 and Section III-C3.

III.C.1) Cost Function: Equations 3 and 7 can be substituted into Equation 1. For each slot $i \in S$ within a control period where S is the set of all MPI slots, SPI_i and P_i depend only on the frequency level assigned to MPI slot i. However, both are nonlinear due to the nonlinearity of SPI and power consumption in CPU frequency. The resulting nonlinear optimization problem cannot be efficiently solved online.

We use a binary variable x_{ij} to indicate whether the frequency f_j is assigned to MPI slot i.

$$x_{ij} = \begin{cases} 1, & f_j \text{ is assigned to MPI slot } i \text{ and} \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Note that $\sum_{f_j \in F} x_{ij} = 1, \forall$ slot $i \in S$. Therefore, for each slot $i \in S$, SPI_i can be expressed as follows.

$$\begin{aligned} \text{SPI}_i &= \sum_{f_j \in F} x_{ij} \cdot (c_1 \cdot \text{MPI}_i + c_2/f_j) \\ &= c_1 \cdot \text{MPI}_i + \sum_{f_j \in F} c_2/f_j \cdot x_{ij}. \end{aligned} \quad (10)$$

Since constants c_1, c_2, and F are known at the control point, Equation 10 can be simplified as follows.

$$\begin{aligned} \text{Letting } s_0 &= c_1 \cdot \text{MPI}_i \text{ and} \\ s_j &= c_2/f_j, \forall f_j \in F, \\ \text{SPI}_i &= s_0 + \sum_{j=1}^{|F|} s_j x_{ij}. \end{aligned} \quad (11)$$

where $|F|$ is the number of elements in F. Similarly, the value of the five events in Equation 7 are also known at the control point. It is worth mentioning that the five event counts are also frequency dependent. We therefore normalize event count to instruction count instead of time. For example, for L1 data accesses, we record the number of L1 data cache accesses per instruction (L1DPI), which is independent of frequency. Hence, for MPI slot i with frequency f_j, we have

$$\text{L1DPS}_i(f_j) = \text{L1DPI}_i/\text{SPI}_i(f_j) \triangleq m_{ij,1}. \quad (12)$$

Similarly, we use $m_{ij,2}$, $m_{ij,3}$, $m_{ij,4}$, and $m_{ij,5}$ to represent $\text{L2PS}_i(f_j)$, $\text{L2MPS}_i(f_j)$, $\text{FPPS}_i(f_j)$, and $\text{BRPS}_i(f_j)$. Defining $w_0 = b_0$ and $w_{ij} = \sum_{k=1}^{5} b_i \cdot m_{ij,k} + b_6 \cdot f_j^{1.5}, \forall f_j \in F$, allows the power consumption for MPI slot i to be expressed as follows:

$$P_i = w_0 + \sum_{j=1}^{|F|} w_{ij} x_{ij}. \quad (13)$$

Combining Equations 11 and 13, Equation 1 can be rewritten as follows:

$$E_{total} = I_{total} \sum_{i \in S} poi_i \cdot (w_0 + \sum_{j=1}^{|F|} w_{ij} x_{ij})(s_0 + \sum_{k=1}^{|F|} s_k x_{ik}). \quad (14)$$

Note that poi_i is known at the control point. In addition,

$$x_{ij} \cdot x_{ik} = \begin{cases} x_{ij}, \text{if and only if } j = k \text{ and} \\ 0, \text{otherwise.} \end{cases} \quad (15)$$

Therefore, Equation 14 can be simplified as follows.

$$\begin{aligned} \text{Letting } e_0 &= I_{total} \cdot w_0 s_0 \text{ and} \\ e_{ij} &= poi_i(w_0 s_j + w_{ij} s_0 + w_{ij} s_j), \\ E_{total} &= e_0 + \sum_{i \in S} \sum_{f_j \in F} e_{ij} x_{ij}. \end{aligned} \quad (16)$$

III.C.2) Performance Constraint – the Oracle Solution: We first assume that we have a priori knowledge of SPI_{fmax} and the MPI distribution throughout the program execution and demonstrate we can solve this problem optimally. Our solution has two stages: profiling and evaluation. During profiling, we record the necessary information, e.g., SPI_{fmax} as well as the percentage of instructions and the hardware performance counter values, for each MPI slot. This allows an optimal solution to the problem. During evaluation, we use the optimal solution obtained in the profiling stage to adjust the frequency dynamically to minimize energy consumption while honoring the performance constraint. The formulation we have just described computes the optimal solutions an oracle would yield. It therefore allows us to determine an upper bound on the energy savings given a particular performance constraint. We will later propose an on-line DVFS technique requiring no application pre-characterization. We will evaluate the quality of this prediction-based technique, called P-DVFS, by comparing its results with those of the optimal oracle formulation.

Assuming the number of instructions associated with MPI slot i is denoted as I_i, Equation 2 can be rewritten as

$$\sum_{i \in S} \sum_{f_j \in F} I_i \cdot \text{SPI}_i(f_j) \cdot x_{ij} \leq (1 + \alpha) T_{fmax}. \quad (17)$$

Dividing both sides by I_{total} yields

$$\sum_{i \in S} \sum_{f_j \in F} poi_i \cdot \text{SPI}_i(f_j) \cdot x_{ij} \leq (1 + \alpha) \text{SPI}_{fmax}. \quad (18)$$

Although we can use Equation 3 to express SPI as a function of MPI and frequency, in reality we record $\text{SPI}_i(f_j)$ during profiling to eliminate the impact of linear regression error on the quality of the optimal solution. More specifically, at each scaling point during profiling, the frequency is reduced to the closest lower level. When the frequency cannot be reduced further, we increase the frequency to the highest level. This process is repeated until the program under profiling finishes. We then compute the average $\text{SPI}_i(f_j)$ associated with each MPI slot i and each frequency f_j. Hence, we can treat $\text{SPI}_i(f_j)$ as a constant k_{ij}. Equation 18 thus becomes

$$\sum_{i \in S} \sum_{f_j \in F} poi_i \cdot k_{ij} \cdot x_{ij} \leq (1 + \alpha) \text{SPI}_{fmax}. \quad (19)$$

I_{total} and e_0 are constants. Thus, the problem can be formulated as follows:

$$\text{Minimize } \sum_{i \in S} \sum_{f_j \in F} e_{ij} x_{ij} \quad (20)$$

$$\begin{aligned} \text{Subject to } & \sum_{i \in S} \sum_{f_j \in F} poi_i \cdot k_{ij} \cdot x_{ij} \leq (1 + \alpha) \text{SPI}_{fmax} \text{ and} \\ & x_{ij} \in \{0, 1\}, \ \sum_{f_j \in F} x_{ij} = 1, \forall i \in S \end{aligned} \quad (21)$$

Note that x_{ij} are binary integer variables and e_{ij}, poi_i, and $k_{i,j}$ are positive constants. Therefore, by scaling the constants with a large positive number, we can make the coefficients e_{ij}, poi_i, and $k_{i,j}$ and the right hand side of the constraint in Equation 21 positive integers. Thus, the formulation can be treated as an multiple-choice knapsack problem (MCKP) [12]. We solve this problem optimally using "lp_solve" [13]. We record the frequencies assigned to each MPI value in an $|S| \times |F|$ lookup table. During evaluation, we use the current MPI value to look up and adjust the frequency at each scaling point.

III.C.3) Performance Constraint – P-DVFS: For this formulation, we assume that the MPI distribution is unknown. However, our MPI distribution prediction technique relies on the similarity between present and future MPI distributions. It is known that most programs have repeated phases with periods ranging from 200 ms–2 s [14]. Therefore, this assumption holds given a reasonable observation duration. In our experiments, we use performance counter values during the most recent control period when deriving the optimal frequency settings for the next control period. We will also discuss our using solutions when the total number of instructions are known or unknown. In the rest of the paper, we will use P-DVFS (*predictive DVFS*) to indicate the online predictive DVFS technique.

At each control point, the number of instructions retired is known. It is therefore natural to use the remaining number of instructions I_r and remaining energy consumption E_r instead of I_{total} and E_{total} in our problem formulation. We first note that Equation 16 is still applicable, except that E_{total} and I_{total} should be replaced with E_r and I_r. Given that T_{elap} is the elapsed time and T_r is the remaining

execution time, Equation 2 can be written as

$$T_r = I_r \cdot \sum_{i \in S} poi_i \cdot \text{SPI}_i(f_i) \leq (1+\alpha) T_{fmax} - T_{elap}. \quad (22)$$

Equation 3 allows us to rewrite the left side of Equation 22 as

$$I_r \cdot \sum_{i \in S} poi_i \cdot \text{SPI}_i(f_i) = I_r \cdot \sum_{i \in S} \sum_{f_j \in F} d_{ij} x_{ij}, \quad (23)$$

where $d_{ij} = poi_i / \left(c_1 \cdot \text{MPI}_i + {}^{c_2}/_{f_j}\right), \forall f_j \in F$. Therefore, Equation 22 can be simplified as follows:

$$\sum_{i \in S} \sum_{f_j \in F} d_{ij} x_{ij} \leq \frac{(1+\alpha) T_{fmax} - T_{elap}}{I_r}. \quad (24)$$

Execution Time Prediction: Equation 24 requires an accurate prediction of T_{fmax} at each control point. By comparing T_{elap} with $(1+\alpha)T_{fmax}$, we can roughly estimate how aggressively we should adjust the CPU frequency during the remaining execution time. If $T_{elap} << (1+\alpha)T_{fmax}$, we can reduce the CPU frequency to a much lower level than that if $T_{elap} >> (1+\alpha)T_{fmax}$. However, it is challenging to predict T_{fmax} accurately online because (1) the control algorithm changes the CPU frequency very rapidly, thus resulting in rapid and significant performance fluctuations and (2) the prediction algorithm should impose little overhead.

In order to derive a fast and accurate prediction method, we fist decompose T_{fmax} into two parts: the amount of time it takes to execute the instructions retired when running at the highest frequency $T_{elap,max}$ and the remaining time to finish execution when running at the highest frequency $T_{remain,max}$. We can derive $T_{elap,max}$ using Equation 26. f_k is the frequency used for scaling period k, T_{k,f_k} is the amount of time elapsed at frequency f_k, f_{max} is the highest frequency, and MPI_k is the average MPI value, i.e., the amount of time required to execute the same number of instructions in period k when the highest frequency is employed.

$$T_{k,max} = T_{k,f_k} \cdot \frac{\text{SPI}(\text{MPI}_k, f_{max})}{\text{SPI}(\text{MPI}_k, f_k)}. \quad (25)$$

Therefore, $T_{elap,max}$ can be expressed as

$$T_{elap,max} = \sum_k T_{k,max} = \sum_k \left(T_{k,f_k} \cdot \frac{\text{SPI}(\text{MPI}_k, f_{max})}{\text{SPI}(\text{MPI}_k, f_k)} \right). \quad (26)$$

In order to determine $T_{remain,max}$, we first assume the instruction count of the current task is known, e.g., by examining the input data file size or history information. This assumption holds for most data processing applications such as image encoding and decoding, data compression, and placement and routing, whose run times are generally functions of input file size. Given that I_{total} is the total instruction count, I_{elap} is the number of instructions retired, I_r is the remaining number of instructions to be executed, and $\text{SPI}(f)$ is the amount of time per instruction at frequency f, we can express $T_{remain,max}$ as follows.

$$I_r = I_{total} - I_{elap} \text{ and} \quad (27)$$

$$T_{remain,max} = I_r \cdot \text{SPI}(f_{max}) \quad (28)$$

Combining Equations 26 and 28, T_{fmax} can be expressed as

$$T_{fmax} = T_{elap,max} + T_{remain,max}. \quad (29)$$

We also consider the scenario in which the total instruction count is unknown before the task is executed. We use I_r to denote the remaining number of instructions, in billions. We start with an I_r of 1. At every scaling point, we subtract, from the current I_r, the number of instructions retired since the last reset of I_r. If the result is smaller than 1, we reset I_r to the number of instructions retired since the task started. If the resulting I_r exceeds an upper bound I_{up},

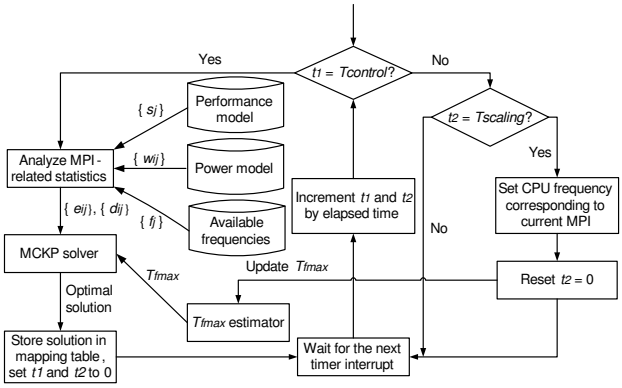

Figure 1. System architecture for P-DVFS.

we set I_r to I_{up}. I_r is then substituted into Equation 28 to estimate the remaining execution time. Note that I_{up} should be large enough to permit aggressive frequency control and yet small enough to preserve accuracy. We use an I_{up} of 30 (billion) in our experiments. We experimentally determined that the energy consumption is relatively insensitive to changes in I_{up}: a variation of only 0.8% in total energy consumption is observed when varying I_{up} from 5 to 500. In our experiments, given a performance degradation ratio of 0.2, the energy consumptions only deviate by 2% from those if I_{total} is known beforehand, i.e., from pre-characterization, file size based estimates, or assuming an oracle with knowledge of future application behavior.

Given that T_{fmax} and I_r can be estimated online, the energy minimization problem can then be formulated as an MCKP.

$$\text{Minimize } \textstyle\sum_{i \in S} \sum_{f_j \in F} e_{ij} x_{ij} \quad (30)$$

$$\text{subject to } \textstyle\sum_{i \in S} \sum_{f_j \in F} d_{ij} x_{ij} \leq \frac{(1+\alpha) T_{fmax} - T_{elap}}{I_r} \text{ and} \quad (31)$$

$$x_{ij} \in \{0,1\}, \ \textstyle\sum_{f_j \in F} x_{ij} = 1, \forall i \in S. \quad (32)$$

We can treat the right hand side of the constraint in Equation 31 as positive. Otherwise, the constraint is trivially satisfied. Unlike the oracle scenario, the P-DVFS technique requires solving MCKP online. Although MCKP is $\mathcal{NP}$-hard, there exist algorithms that can solve it in pseudo-polynomial time [15], [12]. We used "lp_solve" to obtain optimal solutions online. Our experiments had 15 MPI slots and 4 frequency levels. For each of the evaluated benchmarks, it took less than 1 ms to obtain the optimal solution, which is fast enough for online control. Note that this also indicates the energy overhead of the MCKP solver is approximately 0.1%, given the control period of 1 s in our experiments. Pisinger's MCKP solver implementation would permit an even more efficient solution in a production version of the control software [15].

III.D. P-DVFS System Architecture

We have integrated the performance model, power model, execution time predictor, and MCKP solver to accurately control the CPU frequency for a fine-grained trade-off between performance and energy. Figure 1 illustrates the system architecture for the P-DVFS technique. We use $T_{control}$ and $T_{scaling}$ to represent the control and scaling periods. As indicated in Figure 1, whenever a timer interrupt occurs, we increment the time counters t_1 and t_2. We first determine whether t_1 has reached $T_{control}$. If so, we analyze MPI-related statistics, i.e., divide the range of MPI values into distinct MPI slots and calculating the percentage of instructions (poi_i) associated with each MPI slot i, and determine the values of coefficients such as $\{s_j\}$ in Equation 11 and $\{w_{ij}\}$ in Equation 13 using the performance and power models. We also gather information about the available

processors frequencies f_j. These values are translated to $\{e_{ij}\}$ and $\{d_{ij}\}$ in Equations 30 and 31, which are then provided to the MCKP solver along with estimates of T_{fmax} and I_r in Equation 27 and 28. The optimal solutions are then stored in a mapping table and time counters t_1 and t_2 are reset to 0. When $t_1 < T_{control}$, we continue to check whether t_2 has reached $T_{scaling}$ and if so, we set the CPU frequency to the value corresponding to the current MPI in the mapping table and reset the time counter t_2. Otherwise, the T_{fmax} estimate is updated. The task then continues executing until the next timer interrupt occurs. Note that the DVFS algorithm is implemented in software and has very low performance and energy overhead (approximately 0.3%).

IV. Experimental Results

In this section, we first describe the experimental setup and implementation details of the proposed techniques. We then present the experimental results for both P-DVFS and the optimal two-stage solution. Finally, we compare the results produced by P-DVFS with those produced by the optimal oracle solution and the most advanced previous work [7].

IV.A. Experimental Setup

We implemented our techniques on a Pentium Dual Core E2220 processor running Linux 2.6.25 and operates at 1.2, 1.6, 2.0, and 2.4 GHz. Experimental results indicate the switching overhead ranges from 50 µs to 200 µs. We use PAPI 3.6.2 [16] for hardware performance counter measurement and experimentally determined that the performance overhead for accessing hardware performance counter is negligible. Due to hardware limitations, we can only sample two architectural events at a time. Therefore, we time multiplex architectural event sampling to obtain all the values needed for power calculation. The switching interval is 10 ms and five architectural event counters are monitored, yielding a scaling period, ($T_{scaling}$) of 30 ms. The control period $T_{control}$ is set to 1 s, i.e., we solve the MCKP formulation every 1 s such that we can obtain a stable MPI distribution and capture changes in memory access behavior quickly enough for accuracy. We use a sliding window of 2 s to build the MPI distribution histogram. 15 MPI slots are used to permit different memory access behaviors to be distinguished while controlling MCKP solver overheard. We experimentally determined that energy consumption is relatively insensitive to changes in the number of MPI slots: a variation of less than 0.5% in total energy was observed when varying the number of slots from 5 to 30. The same MPI slots are used throughout the execution of a benchmark.

To determine power consumption, we use a Fluke i30 current clamp on the 12 V processor power supply lines, the output of which is sampled at 10 kHz using a National Instruments USB6210 data acquisition card. This approach permits processor power consumption measurement without requiring printed circuit board rework or access to internal metal layers. An on-chip voltage regulator converts this voltage to the actual processor operating voltage. We assume a regulator efficiency of 90%.

IV.B. Comparison with Prior Work

Choi et al. [7] proposed a fine-grained runtime DVFS technique that minimizes energy consumption while meeting soft timing constraints. We will use "F-DVFS" to refer to their technique. In order to adapt to changes in the rate of off-chip accesses, F-DVFS dynamically constructs a performance model and uses it to calculate the expected workload for the next slot; frequency and voltage levels are adjusted accordingly. F-DVFS ignores long-term behavior such as the total application execution time. For example, at each scaling point, it considers only an immediate, local, user-specified performance constraint. However, sometimes even setting the frequency to the lowest level still results in a performance level higher than the user-specified constraint due to large number of off-chip accesses, opening the opportunity to improve energy savings when the MPI becomes lower later during execution. Neglecting total execution time makes it impossible to take advantage of such energy saving opportunities. Note that this sort of time-varying application behavior is very common for scientific computing applications, which commonly read a large amount of data into memory before processing. Moreover, F-DVFS neglects the relationship between frequency and energy consumption, assuming that reducing frequency is always beneficial to energy. However, this is not true when leakage power consumption is significant or the overall optimization goal is to minimize system energy consumption instead of processor power consumption. In contrast, P-DVFS automatically models and optimizes leakage power consumption and can be easily extended to handle the energy consumptions of other components such as main memory and disk.

TABLE I
Performance Degradations of F-DVFS and P-DVFS in terms of total execution time

Benchmark	F-DVFS (%)				P-DVFS (%)			
Goal	5%	10%	15%	20%	5%	10%	15%	20%
gzip	0.27	0.34	1.36	10.59	4.74	8.03	10.82	16.62
vpr	0.00	1.91	10.06	11.62	4.83	9.93	14.05	19.39
mcf	2.02	4.51	6.61	7.78	4.50	6.50	13.50	17.00
bzip2	0.51	0.62	0.67	17.9	3.11	6.09	10.76	15.36
twolf	0.0	1.87	16.31	17.9	4.13	7.92	12.40	17.23
art	0.0	4.47	5.20	5.85	3.09	6.85	13.16	16.83
equake	0.0	0.0	0.0	9.64	3.04	7.59	11.72	15.42
ammp	0.23	0.93	7.18	16.13	4.24	10.40	14.41	19.29
facerec	0.0	4.09	10.12	20.2	3.19	7.65	13.65	18.38
sphinx3	0.0	0.54	1.48	9.34	2.80	7.50	11.10	13.84
tachyon	0.0	5.91	6.83	16.4	3.22	8.41	13.57	18.43
Average	0.28	2.29	5.98	13.03	3.72	7.90	12.65	17.10

IV.C. Experimental Results

We evaluated P-DVFS on the 8 SPEC2000 benchmarks that compiled on our evaluation platform and 3 ALPBench benchmarks [17], [18]. We did not consider the remaining 2 benchmarks ("MPGenc" and "MPGdec") in the ALPBench benchmark suite because they are very disk I/O intensive: we are presently interested in evaluating the impact of off-chip memory access on energy savings. We considered 3 floating point programs and 8 integer programs. The execution time of each benchmark ranges from 40–425 s. For each benchmark, we specify a performance degradation ratio (the maximum increase in execution time relative to that at the maximum frequency and voltage) ranging from 5% to 20% with a step of 5%. The actual execution time and the average energy savings are reported compared to a scheme without DVFS (N-DVFS), F-DVFS, and the optimal oracle solution; we use the same window size for each to permit a fair comparison. Both techniques use 4 discrete frequency levels.

Table I shows the actual performance degradation for both F-DVFS and P-DVFS compared with the user-specified performance degradation ratios. The first column specifies the benchmarks we evaluated. The "P-DVFS" and "F-DVFS" columns indicate the performance degradation ratios resulting from using the two techniques, with the user-specified performance degradation constraint listed on the second "Goal" row. Given that the performance constraint is satisfied, a larger performance degradation usually corresponds to more energy savings; this was confirmed by our experiments. Experimental results indicate that P-DVFS approaches the user-specified performance level more closely than F-DVFS, implying greater energy savings. P-DVFS has finer-grained control over the trade-offs between performance and energy given a user-desired performance constraint. F-DVFS does

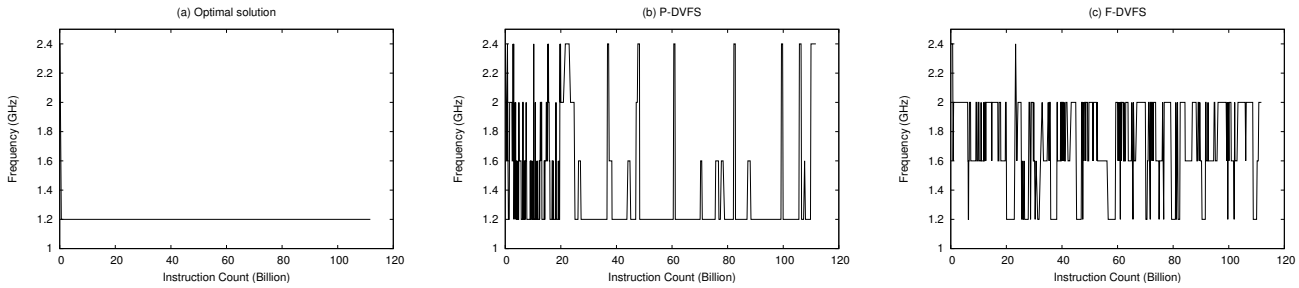

Figure 2. Processor frequency as a function of the number of instructions retired for (a) the oracle solution, (b) P-DVFS, and (c) F-DVFS for "mcf" with a performance degradation ratio of 20%.

TABLE II
DEVIATION OF ENERGY CONSUMPTIONS FROM THE OPTIMAL SOLUTION WHEN USING USING N-DVFS, F-DVFS, AND P-DVFS

Benchmark	E_{opt} (J)	N-DVFS (%)	F-DVFS (%)	P-DVFS (%)
gzip	804	7.88	6.88	0.12
vpr	1520	21.91	8.09	3.36
mcf	2401	71.10	29.86	4.83
bzip2	1345	8.18	1.93	0.30
twolf	5281	12.61	1.50	1.38
art	1810	52.49	23.20	4.42
equake	2736	14.58	7.20	1.90
ammp	7344	12.15	2.08	0.14
facerec	2621	12.59	6.37	0.04
sphinx3	1428	19.54	11.13	3.64
tachyon	2210	15.43	9.55	0.05
Average	2682	22.59	9.80	1.83

not reach the user-specified performance degradation ratio partially because the number of available frequencies is limited: whenever the calculated frequency f_{calc} does not correspond to any available frequency, F-DVFS uses the closest frequency that is larger than f_{calc} to approximate it. This may reduce the energy benefit when the number of available frequency is small. Switching between two closest available frequencies may address this problem. However, there are more fundamental reasons why F-DVFS does not work as well as P-DVFS, as we will explain later in this section. Note that both techniques may violate the soft timing constraint due to inaccuracies in the online performance model. However, for P-DVFS, the maximum violation for these benchmarks is less than 1%, which could be eliminated by using a 1% guard band for the constraint.

We compared the energy savings of N-DVFS, F-DVFS, and P-DVFS with those of the optimal oracle solution, which might be better than the actual optimal on-line solution. For performance degradation percentages of 5%, 10%, and 15%, N-DVFS generates solutions that deviate from the optimal solution by 9.31%, 12.81%, and 18.46%, with maximum deviations of 22.29%, 33.72%, and 56.55%; F-DVFS leads to energy consumptions that deviate from the optimal solution by 7.1%, 8.23%, and 9.51%, with maximum deviations of 16.84%, 15.89%, and 29.8%; and P-DVFS results in energy consumptions that deviate from the optimal solution by 1.43%, 1.16%, and 1.59%, with maximum deviations of 2.80%, 3.88%, and 4.63%. Since the results are similar for different performance degradation ratios, we only present the energy numbers for a maximum performance degradation ratio of 20% in Table II. The first column specifies the application being evaluated. The second column indicates the optimal, i.e., minimum, energy consumption for each benchmark with a performance degradation ratio of 20%. The third, the fourth, and the fifth columns represent the deviation in energy consumption from that of the optimal oracle solution when using N-DVFS, F-DVFS, and P-DVFS. As indicated in Table II, the energy consumption deviates from the optimal oracle solution by 22.59% on average when no DVFS is used, with a maximum deviation of 71.1%. F-DVFS produces solutions that deviate 9.80% from the optimal oracle solution on average, with a maximum deviation of 29.86%. Among the three candidates, P-DVFS achieves the best solution quality, i.e., an average of 1.83% deviation from the optimal oracle solution with a maximum deviation of 4.83%. Therefore, we conclude that P-DVFS can very closely approximate optimal solutions. It is also worth noting that for performance degradation ratios of 5%, 10%, 15%, and 20%, P-DVFS has average power savings of 8.3%, 11.31%, 12.3%, and 9.93% and maximum power savings of 15.94%, 12.69%, 27.36%, and 25.64% compared to F-DVFS.

For benchmarks "mcf" and "art", F-DVFS leads to solutions that are far worse than those using P-DVFS (25.03% and 18.78% difference, respectively). We now analyze their results for these benchmarks.

Analyzing *Mcf* Results: Figure 2 illustrates the dynamic processor frequency changes for the optimal oracle solution, P-DVFS, and F-DVFS during execution of the "mcf" benchmark, given a performance degradation ratio of 20%. The X-axis indicates the number of billion instructions retired and the Y-axis indicates the frequency. Figure 2(a) suggests that the optimal solution is to always set the frequency to the lowest level. While P-DVFS yields a near-optimal solution, F-DVFS behaves very differently. We note that "mcf" is a two-phase benchmark: the cache miss rate is very high during the first 20 billion instructions and alternates between a high value and a low value afterwards. In both phases, F-DVFS leads to a higher frequency on average. Recall that F-DVFS requires accurate model estimation and accurate individual coefficients so that it can correctly estimate the ratio of off-chip to on-chip memory accesses. Although the former is generally true for linear regression, the second assumption does not necessarily hold. In this case, since the MPI and CPI values do not change much in the first phase, the coefficients derived using linear regression can be inaccurate, causing F-DVFS to significantly over-estimate the average on-chip latency and thus limit itself to a relatively high frequency (2 GHz). Note that the output of the performance model, or CPI, is still accurate. In contrast, P-DVFS only requires that the output of the model match the real CPI value: the individual coefficients in the regression formula do not matter. Therefore, P-DVFS allows the CPU frequency to be decreased to a lower level, alternating between 1.6 GHz and 1.2 GHz most of the time. The frequency does not stay at the lowest level due to inaccuracies in the online performance model and the remaining execution time predictor. In the second phase, F-DVFS increases the frequency when the cache miss rate is lower and decreases the frequency when the miss rate is higher. This happens because F-DVFS considers only immediate application behavior and ignores long-term behavior. P-DVFS takes history and long-term behavior into account, allowing it to correctly determine that the frequency can be set to the lowest level even when the cache miss rate is low. Therefore, P-DVFS achieves much larger energy savings in this case, which approach those of the optimal oracle solution.

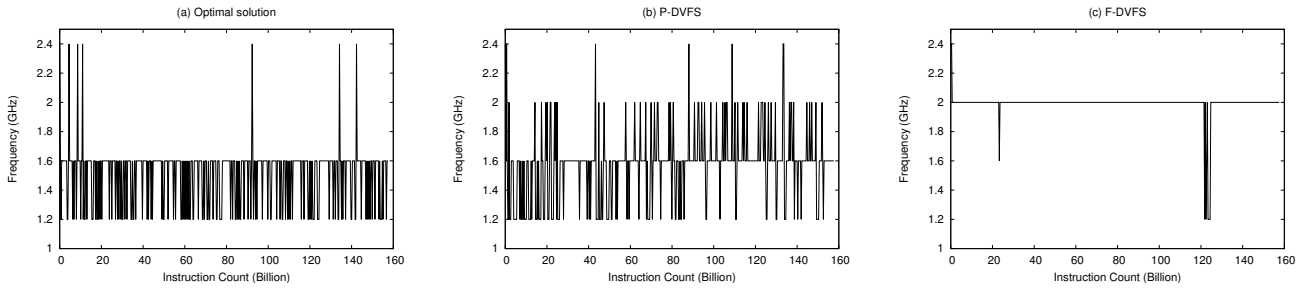

Figure 3. Processor frequency as a function of the number of instructions retired for (a) the optimal solution, (b) P-DVFS, and (c) F-DVFS during "art" execution with a performance degradation ratio of 20%.

Analyzing *Art* Results: Figure 3 illustrates the dynamic processor frequency changes for the optimal oracle solution, P-DVFS, and F-DVFS during the execution of the "art" benchmark, given a performance degradation ratio of 20%. P-DVFS closely approximates the optimal oracle solution and F-DVFS does not. This can be explained as follows. "Art" has periodic cache access behavior with a period of approximately 300 ms at the highest frequency. In each period, the MPI value starts from a low value (0.003 in our experiments) and gradually increases before it reaches the point with the highest MPI (0.005 in our experiments). Then, the MPI value starts to decrease until it returns to the previous value of 0.003. F-DVFS gathers the sampling points within the most recent second to build the performance model. The coefficients in the regression formula remain nearly constant due to the small period and large window size. Therefore, the frequency was set to a fixed number (2 GHz in our case) for all the sampling points in each period. In contrast, P-DVFS builds the MPI distribution based on the sampling points from the most recent second, translates the energy minimization problem into an MCKP instance, and solves it to get the optimal solution. This solution uses high frequency (2 GHz) for sampling points with low MPI and low frequency (1.2 GHz) for sampling points with high MPI. This permits significant reduction in energy consumption compared to F-DVFS. Since F-DVFS is not distribution-oriented, it cannot determine how SPI and power consumption change with MPI. Therefore, it cannot assign different frequencies to sampling points with different MPIs while still meeting the performance constraint.

For the rest of the benchmarks, P-DVFS slightly outperforms F-DVFS. This is because both consider the effects of off-chip memory access latencies on energy. P-DVFS achieves the greatest energy savings compared to past work for applications with phases during which the energy cost per instruction differ.

V. Conclusions

This paper has described a new power state control technique that adapts to the time-varying memory access behaviors of applications. We first proposed a two-stage DVFS algorithm based on formulating the throughput-constrained energy minimization problem as a multiple-choice knapsack problem (MCKP), assuming a priori characterization-based or oracle knowledge of application behavior. This algorithm builds on an application phase-dependent power model, which can be constructed offline using processor hardware performance counters. We then present an online DVFS technique, called P-DVFS, that predicts remaining execution time in order to control voltage and frequency to minimize energy consumption subject to a soft performance constraint. P-DVFS requires no a priori knowledge of application behavior. P-DVFS also uses a model that accurately captures the relationship between performance and off-chip memory access rate. These two models, combined with an execution time predictor, allow us to formulate the energy minimization problem as a multiple-choice knapsack problem, which can be efficiently and optimally solved online. Experimental results indicate that given a performance degradation ratio of 0.2, P-DVFS leads to energy consumptions within 1.83% of the optimal oracle solutions on average with a maximum deviation of 4.83%, whereas the most advanced related DVFS control technique (F-DVFS) results in energy consumptions within 9.80% of the optimal oracle solution on average with a maximum deviation of 29.86%. For the same performance constraint, we found that P-DVFS also reduces power consumption by 9.93% on average and up to 25.64% compared to F-DVFS. These energy and power savings are all directly measured on a real system.

References

[1] J.-M. Chang and M. Pedram, "Energy minimization using multiple supply voltages," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, no. 4, pp. 436–443, Dec. 1997.

[2] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. Design Automation Conf.*, June 2002, pp. 183–188.

[3] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2003, pp. 510–517.

[4] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. Int. Symp. Microarchitecture*, Nov. 2003, pp. 359–370.

[5] Q. Wu, et al., "A dynamic compilation framework for controlling microprocessor energy and performance," in *Proc. Int. Symp. Microarchitecture*, Nov. 2005, pp. 271–282.

[6] Y. Liu, et al., "Thermal vs energy optimization for DVFS-enabled processors in embedded systems," in *Proc. Int. Symp. Quality of Electronic Design*, Jan. 2007, pp. 204–209.

[7] 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 Trans. Computer-Aided Design of Integrated Circuits and Systems*, Dec. 2004, pp. 18–28.

[8] A. R. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, MA, 1995.

[9] "Intel 64 and IA-32 Architectures Software Developer's Manual," http://www.intel.com/products/processor/manuals/.

[10] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Prentice-Hall, 2003.

[11] C. Poirier, et al., "Power and temperature control on a 90 nm Itanium–family processor," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 2005, pp. 304–305.

[12] P. Sinha, "The multiple-choice knapsack problem," *Operations Research*, vol. 27, no. 3, pp. 503–515, 1979.

[13] "lpsolve 5.5," http://lpsolve.sourceforge.net/5.5/.

[14] C. Isci, A. Buyuktosunoglu, and M. Martonosi, "Long-term workload phases: Duration predictions and applications to DVFS," *IEEE Micro*, no. 25, pp. 39–51, Oct. 2005.

[15] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem," *European J. of Operational Research*, pp. 394–410, 1995.

[16] "PAPI 3.6.2," http://icl.cs.utk.edu/papi/.

[17] J. L. Henning, "SPEC CPU2000: Measuring CPU performance in the new millennium," *Computer*, pp. 28–35, July 2000.

[18] "The ALPBench Benchmark Suite (Version 1.0)," http://rsim.cs.illinois.edu/alp/alpbench.