Dynamic Voltage and Frequency Scaling Under a Precise Energy Model Considering Variable and Fixed Components of the System Power Dissipation*

Kihwan Choi, Wonbok Lee, Ramakrishna Soma, and Massoud Pedram
Department of EE-Systems, University of Southern California, Los Angeles, CA90089
{kihwanch, wonbokle, rsoma, pedram}@usc.edu

ABSTRACT
This paper presents a dynamic voltage and frequency scaling (DVFS) technique that minimizes the total system energy consumption for performing a task while satisfying a given execution time constraint. We first show that in order to guarantee minimum energy for task execution by using DVFS it is essential to divide the system power into *active* and *standby* power components. Next, we present a new DVFS technique, which considers not only the active power, but also the standby component of the system power. This is in sharp contrast with previous DVFS techniques, which only consider the active power component. We have implemented the proposed DVFS technique on the BitsyX platform - an Intel PXA255-based platform manufactured by ADS Inc., and report detailed power measurements on this platform. These measurements show that, compared to conventional DVFS techniques, an additional system energy saving of up to 18% can be achieved while satisfying the user-specified timing constraints.

1. Introduction
Demand for low power consumption in battery-powered computer systems has risen sharply. This is due to the fact that extending the service lifetime of these systems by reducing their power dissipation requirements is a key requirement. Low power design is a critical design consideration even in high-end computer systems where expensive cooling and packaging costs and lower reliability often associated with high levels of on-chip power dissipation are the important concerns.

Dynamic voltage and frequency scaling (DVFS) technique has proven to be a highly effective method of achieving low power consumption for the CPU while meeting the performance requirements due to the quadratic relation between the energy consumption and operating voltage of a CMOS circuit. A number of modern microprocessors such as Intel's XScale [1] and Transmeta's Crusoe [2] are equipped with the DVFS functionality.

There have been extensive studies of low power system designs using DVFS techniques [3]-[8]. All of these works have focused only on the reduction of the CPU power. However, in reality, the battery lifetime of a computer system is also affected by the fixed power consumed in other components of the system, which have their own operating frequency and voltage levels. The heterogeneity in performance and power dissipation of these components make it difficult to apply DVFS techniques.

To guarantee minimum energy for task execution while satisfying a given time constraint, it is important to divide the system power into two parts: *fixed* and *variable* power. Fixed power represents the component of power that remains unchanged during the task execution. Examples include DC-DC converter power and PLL power as well as leakage power dissipation. Variable power captures the component of system power consumption that changes with time. Examples include the CPU and memory power dissipations as well as I/O controller power. The variable power component is, in turn, decomposed into two subcomponents: *idle* and *active* power. As the name implies, *active* (*idle*) power is the portion of variable power that is consumed when the system is executing some (no) useful task. We also define *standby* power as the summation of fixed plus idle power components of the system.

There have been many studies on DVFS for both real-time and non real-time operations. These works can roughly be divided into inter-task [4][5] and intra-task [6][7] depending on the scaling granularity. Alternatively, they may be divided into application-specific [6][7] and general-purpose [3][5][8] depending on whether or not the application program must be modified. However, all these approaches focus only on the CPU energy saving and are based on two key assumptions: 1) an inverse relationship between the task execution time and the operating frequency and 2) a cubic relationship between the system power and the operating frequency. More recently, a number of DVFS approaches have attempted to exploit the asynchrony of memory access to the CPU clock during a task execution. For example, in [9] and [10], compiler-assisted DVFS techniques were proposed, in which CPU frequency is lowered in the memory-bound region of a program with little performance degradation. Other DVFS approaches have made use of embedded hardware, i.e., a performance monitoring unit (PMU). For example, in [11], IPC (instruction per cycle) rate of a program execution is used to guide the voltage scaling. Reference [12] proposes to choose the optimal CPU clock frequency under a fixed performance degradation constraint based on dynamic program behavior such as the number of executed instructions and memory access counts during the whole execution time by using the PMU. In [13] a DVFS technique which enables more precise energy-performance trade-off by using the PMU is presented in which the optimal CPU clock frequency and the corresponding minimum voltage level are chosen based on the ratio of the on-chip computation time to the off-chip

access time. A similar DVFS approach exploiting the ratio of the on-chip and off-chip access times has been proposed for the MPEG decoding application [14].

DVFS can reduce only the active component of system power dissipation. If this component is large compared to the standby component of system power, then lowering the CPU clock frequency and the supply voltage will result in lower system energy consumption due to the quadratic relation between the CPU power consumption and voltage. On the other hand, if the active component of system power is small compared to the standby component, then slowing down the CPU speed may in fact increase the system energy consumption due to an increase in the task execution time and the dominance of the standby power dissipation component.

In this paper, we present a new DVFS technique, which considers not only *active* power, but also *standing* power components of the system. The *standing* components of the system power are measured by monitoring the system power when it is idle. The *active* component of the system power is estimated at run time by a technique known as workload decomposition whereby the workload of a task is decomposed into on-chip and off-chip, based on statistics reported by a performance monitoring unit (PMU), which most modern processors such as XScale80200 [1] or PXA255 [18] come equipped with. The proposed technique has been implemented on an embedded system platform built around the PXA255 processor. By detailed current measurements, we performed a task with up to 12% less system energy compared to the case with normal DVFS techniques, which consider only variable power.

The remainder of this paper is organized as follows. In Section 2, models for both execution time and power dissipation estimation are described. Details of the target platform and the proposed DVFS policy are presented in Sections 3 and 4, respectively. Experimental results and conclusions are given in Sections 5 and 6, respectively.

2. Workload Decomposition

2.1 Estimating the Execution Time of a Task

Workload of a task is defined as the sum of the CPI's of all instructions in the instruction stream of the task. The task workload depends on various dynamic parameters such as the *on-chip stall* cycle count due to data/control dependency or the branch misprediction, and the *off-chip stall* cycle count due to I/D cache miss or I/D TLB miss. During an off-chip access, the CPU stalls until the requested memory transaction is completed. Thus, CPU clock cycles during an off-chip access are wasted. To explain the workload decomposition technique, we must provide some definitions first.

Definition 1: *On-chip workload*, W^{on}, is the number of CPU clock cycles required to perform the set of on-chip instructions, which are executed inside the CPU only.

The execution time required to finish W^{on}, T^{on}, varies depending on the CPU frequency, f^{cpu}, and is calculated as $T^{on} = W^{on}/f^{cpu}$.

Definition 2: *Off-chip workload*, W^{off}, is the number of external clock cycles needed to perform the set of off-chip accesses. Note that the CPU stalls until the external memory transactions are completed.

The execution time required to finish W^{off}, T^{off}, depends on the external memory clock frequency, f^{ext}, and is calculated as $T^{off} = W^{off}/f^{ext}$.

Based on definitions 1 and 2, W^{on} and W^{off} are written as:

$$W^{on} = N \cdot CPI_{on}^{avg}, \quad W^{off} = M \cdot CPI_{off}^{avg} \tag{1}$$

where $CPI^{avg}{}_{on}$ denotes the number of CPU clock cycles per on-chip instruction, M is the number of off-chip accesses, and $CPI^{avg}{}_{off}$ denotes the number of external clock cycles per an off-chip access. From these two definitions, the execution time, T, for a task is calculated as:

$$T = T^{on} + T^{off} = \frac{N \cdot CPI_{on}^{avg}}{f^{cpu}} + \frac{M \cdot CPI_{off}^{avg}}{f^{ext}} \tag{2}$$

When the CPU frequency changes, the change in T is solely due to T^{on}:

$$\frac{\Delta T}{\Delta f^{cpu}} = \frac{\Delta T^{on}}{\Delta f^{cpu}}, \quad \frac{\Delta T^{off}}{\Delta f^{cpu}} \approx 0 \tag{3}$$

2.2 Modeling the System Power Consumption

We consider a computing system consisting of a CPU with a variable operating frequency, f_n, where $f_{min} \le f_n \le f_{max}$. The system also includes N system modules. Let P_{cpu,f_n} and $P_{mod,i}$ denote the power dissipation of the CPU at f_n and the i^{th} module. Then, the required system energy to complete a task in time T with f_n is given by:

$$E_{sys,f_n} = \int_{t_0}^{t_0+T} P_{sys,f_n}(t) \cdot dt \tag{4}$$

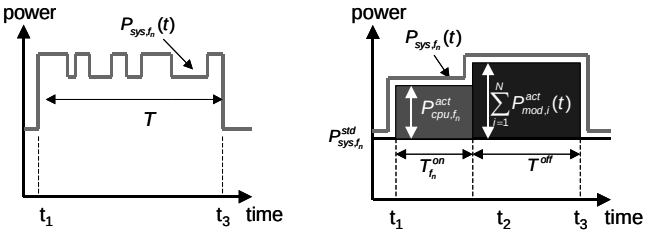
where $P_{sys,f_n}(t)$ is the time-varying system power at f_n and is calculated as:

$$\begin{aligned} P_{sys,f_n}(t) &= P_{sys}^{fix} + P_{sys,f_n}^{idle} + P_{sys,f_n}^{act}(t) = P_{sys,f_n}^{std} + P_{sys,f_n}^{act}(t) \\ &= P_{cpu,f_n}^{std} + \sum_{i=1}^{N} P_{mod,i}^{std} + P_{cpu,f_n}^{act}(t) + \sum_{i=1}^{N} P_{mod,i}^{act}(t) \end{aligned} \tag{5}$$

P_{cpu,f_n}^{act} is the active portion of P_{cpu,f_n}. P_{cpu,f_n}^{std} is the standing portion of P_{cpu,f_n}, which is in turn the summation of the idle portion (P_{cpu,f_n}^{idle}) plus the fixed portion (P_{cpu,f_n}^{fix}). $P_{mod,i}^{act}$ is the active portion of $P_{mod,i}$ when the i^{th} module is being accessed whereas $P_{mod,i}^{std}$

denotes the standing portion of $P_{mod,i}$ when the i^{th} module is not accessed, which is equal to the idle component of the i^{th} module, $P^{idle}_{mod,i}$.

In eq (5), $\left(P^{std}_{cpu,f_n} + \sum_{i=1}^{N} P^{std}_{mod,i}\right)$ denotes the standing system power consumption, P^{std}_{sys,f_n}, whereas $\left(P^{act}_{cpu,f_n}(t) + \sum_{i=1}^{N} P^{act}_{mod,i}(t)\right)$ denotes the active system power consumption, $P^{act}_{sys,f_n}(t)$.



(a) without decomposition **(b) with decomposition**

Figure 1: System power consumption during task execution

Generally speaking, it is difficult to accurately calculate $P^{act}_{sys,f_n}(t)$ because the power dissipation caused by executing various instructions can be quite different. Only the CPU is needed to execute the on-chip workload whereas, for the off-chip workload, the memory is also required. During the execution of a program, instructions causing on-chip and off-chip works are arbitrarily interleaved. Consequently, $P^{act}_{sys,f_n}(t)$ can severely fluctuate as shown in Figure 1 (a). However, once the workload of a task is decomposed into two contiguous components: on-chip and off-chip, then $P_{sys,f_n}(t)$ can easily be calculated as:

$$P_{sys,f_n}(t) = \begin{cases} P^{act}_{cpu,f_n}(t) + P^{std}_{sys,f_n}, & during \;\; T^{on}_{f_n} \\ \sum_{i=1}^{N} P^{act}_{mod,i}(t) + P^{std}_{sys,f_n}, & during \;\; T^{off} \end{cases} \qquad (6)$$

Figure 1 (b) shows $P_{sys,f_n}(t)$ after workload decomposition.

Hence, E_{sys,f_n} after workload decomposition is given as:

$$E_{sys,f_n} = \left[P^{act}_{cpu,f_n} + P^{std}_{sys,f_n}\right] \cdot T^{on}_{f_n} + \left[\sum_{i=1}^{N} P^{act}_{mod,i} + P^{std}_{sys,f_n}\right] \cdot T^{off} \qquad (7)$$

In eq.(7), P^{act}_{cpu,f_n} and P^{std}_{sys,f_n} can easily be obtained from simple measurements on the target system by running the benchmark programs with different CPU frequencies. However, it is difficult to get $\sum_{i=1}^{N} P^{act}_{mod,i}(t)$ because complete information about how the various system components are used by the target program is not available. In practice, for the software programs used in this paper, $\sum_{i=1}^{N} P^{act}_{mod,i}(t)$ is approximated by the memory power dissipation because memory is the most frequently-used system component and the power consumed in the memory takes up more than half of the system power in our target system. So, we include the power consumptions of all other system components in P^{std}_{sys,f_n}.

2.3 System Energy vs. CPU Frequency

As shown in eq. (7), E_{sys,f_n} is a function of various parameters of the system configuration (P^{act}_{cpu,f_n}, P^{std}_{sys,f_n}, and $P^{act}_{mod,i}$) and the application program ($T^{on}_{f_n}$ and T^{off}). Depending on these parameters, an optimal CPU frequency that results in task execution with minimum system energy consumption is determined as explained next.

The system energy equation (7) is rewritten as:

$$E_{sys,f_n} = P^{act}_{cpu,f_n} \cdot T^{on}_{f_n} \cdot \left[1 + \left(\frac{P^{std}_{sys,f_n}}{P^{act}_{cpu,f_n}}\right) + \left(\frac{P^{act}_{mem} + P^{std}_{sys,f_n}}{P^{act}_{cpu,f_n}}\right) \cdot \left(\frac{T^{off}}{T^{on}_{f_n}}\right)\right] \qquad (8)$$

where P^{act}_{mem} denotes the memory power and is used in place of $\sum_{i=1}^{N} P^{act}_{mod,i}(t)$ in eq. (7) as mentioned before. The case in which P^{std}_{sys,f_n} and P^{act}_{mem} are all zero is equal to the situation assumed in the previous DVFS works, where purely CPU-intensive task is executed on the system consisting of only one CPU with no standby power. In that case, lowering CPU frequency always results in system energy saving. Assuming a linear relationship between the operating voltage and frequency ($P^{act}_{cpu,f_n} \propto f_n^3$ and $T^{on}_{f_n} \propto f_n^{-1}$), then E_{sys,f_n} becomes dependent upon f_n as following form:

$$E_{sys,f_n} = a \cdot f_n^2 + b \cdot f_n^{-1} + c \qquad (9)$$

where *a*, *b*, and *c* are constant coefficients. In particular, *b* and *c* represent the amounts of standby power in the total system power dissipations. Subsequently, an optimal CPU frequency which gives the minimum system energy, f_{opt}, is

calculated as $\sqrt[3]{0.5 \cdot b/a}$ by taking the derivative of eq. (8). If b is zero, then f_{opt} is f_{min}, but f_{opt} increases as b increases.

3. Description of the Target System

3.1 BitsyX Platform

Our target system for DVFS is the BitsyX system from ADS Inc. [15]. BitsyX has a PXA255 microprocessor which can operate from 100MHz to 400MHz, with a corresponding core supply voltage of 0.85V to 1.3V. Power supply for the PXA255 core is provided externally through an on-board variable voltage generator. There are nine different frequency combinations, F_1 to F_9. Each combination is given as a 3-tuple consisting of the processor clock frequency (f^{cpu}), the internal bus clock frequency (f^{int}), and the external bus clock frequency (f^{ext}). These frequency combinations and appropriate CPU voltage levels are reported in Table 1. The internal bus connects the core and other functional blocks inside the CPU such as I/D-cache unit and the memory controller whereas the external bus in the target system is connected to SDRAM (64MB).

1 Frequency combinations in BitsyX system

No	f^{cpu} (MHz)	CPU Volt. (V)	f^{int} (MHz)	f^{ext} (MHz)
F_1	100	0.85	50	100
F_2	200	1.0	50	100
F_3	300	1.1	50	100
F_4	200	1.0	100	100
F_5	300	1.1	100	100
F_6	400	1.3	100	100
F_7	400	1.3	200	100
F_8	133	0.85	66	133
F_9	265	1.0	133	133

3.2 Execution Time Model in BitsyX

To derive a suitable execution timing model for BitsyX, five different applications were run over all frequency sets, F_1 to F_9, and the total execution time for each case was measured and shown in Figure 2.

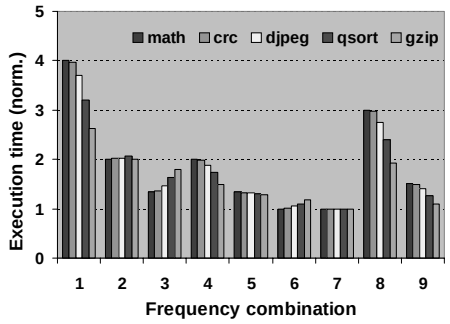


Figure 2: Execution time variation over different frequency combinations

Figure 2 provides the execution time of all applications for each frequency setting normalized to the execution time with the maximum performance setting, i.e., setting F_7. From Figure 2, we can easily see that "math", "crc", and "djpeg" are more CPU-intensive than the "gzip" and "qsort" applications since lowering the CPU frequency for these applications introduce significant execution time increase compared to "gzip" and "qsort" cases. Comparing execution times of settings F_1, F_2 and F_3 (where only the CPU frequency is different, while all other clocks are the same) also validates this observation. In fact, this comparison allows us to determine that "gzip" is more memory-bound than "qsort" by looking at the execution time variation according to CPU frequency only.

Execution time T is sum of T^{on} and T^{off} as in eq.(2). Clearly T^{off} is strongly dependent on the external clock frequency. However, an important observation from data reported in Figure 2 is that the internal bus clock frequency also affects T^{off}. The relation between the *internal* bus clock and T^{off} can be understood from a closer examination of the operations performed during the external memory access. For example, a D-cache miss requires two operations: data fetch from the external memory and data transfer to the CPU core where the cache-line and destination register are updated. The time needed for the latter operation is obviously affected by the internal bus frequency. Due to the lack of exact timing information about these two operations that are performed during a D-cache miss service, we have opted to model T^{off} as a function of both the internal clock frequency and the external memory access clock as follows:

$$T^{off} = T^{off1} + T^{off2} = \frac{\alpha \cdot W^{off}}{f^{int}} + \frac{(1-\alpha) \cdot W^{off}}{f^{ext}} \qquad (10)$$

where α is the ratio between the data *transfer* time (T^{off1}) and the data fetch time (T^{off2}) and f^{int} is internal bus clock

frequency.

Based on the experimental results on various application programs, the average error in predicting the execution time for all applications and over all frequency combinations was less than 2% with α value of 0.35.

3.3 Energy Consumption Model for BitsyX

Measured energy consumptions for each application are presented in Figure 3. Notice that the frequency combination at which the minimum energy is consumed is not F_1. On the contrary, F_1 causes the largest energy dissipation among all frequency combinations. Furthermore, the energy dissipation results of Figure 3 closely follow the execution time results of Figure 2, i.e., lower execution time causes lower system energy dissipation and vise versa. In other words, because of the rather large standby power in the target platform, it is desirable to finish the application program as soon as possible in order to minimize the system energy consumption. Another important observation is that the frequency set that yields the minimal system energy changes depending on the type of the application program. For example, setting F_6 (CPU frequency of 400MHz) produces the minimum energy for "math" which is the most CPU-intensive application program in our suite, whereas setting F_9 (CPU frequency of 265MHz) does the same for "gzip" which is the most memory-intensive program in our suite. Note that the reason that F_9 is the best setting for "gzip" is that F_9 has the fastest condition for memory access operation, i.e., both memory clock and internal bus clock are 133MHz.

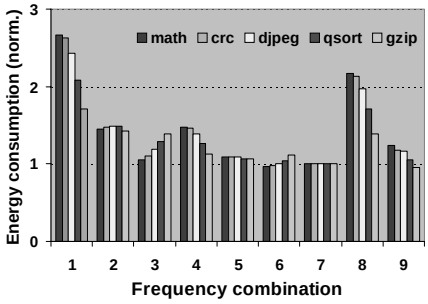


Figure 3: Energy consumption over different frequency combinations

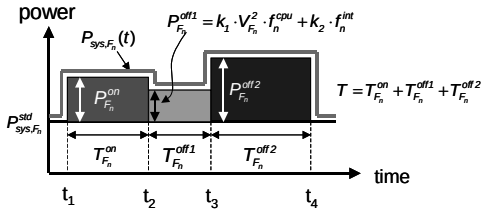


Figure 4: Total system power consumption during execution

Energy consumption model in BitsyX is shown in Figure 4. Terms used in this figure are explained next.

- F_n : *the* n^{th} *frequency setting,* $F_n(f_n^{cpu}, f_n^{int}, f_n^{ext})$
- $T_{F_n}^{on}$: *on-chip computation time at* F_n
- $T_{F_n}^{off1}$: *data update time after fetch from memory at* F_n
- $T_{F_n}^{off2}$: *data fetch time from memory at* F_n
- $P_{sys,F_n}(t)$: *time-varying system power at* F_n
- P_{sys,F_n}^{std} : *standing power in* $P_{sys,F_n}(t)$
- $P_{F_n}^{on}$: *active power in* P_{sys,F_n} *during* $T_{F_n}^{on}$
- $P_{F_n}^{off1}$: *active power in* P_{sys,F_n} *during* $T_{F_n}^{off1}$
- $P_{F_n}^{off2}$: *active power in* P_{sys,F_n} *during* $T_{F_n}^{off2}$
- V_{F_n} : *CPU operating voltage at* F_n
- k_1 : *fitting coefficient for* $P_{F_n}^{off1}$, *[nF]*
- k_2 : *fitting coefficient for* $P_{F_n}^{off1}$, *[V²·nF]*

Here, $P_{F_n}^{off1}$ is represented as $k_1 \cdot V_{F_n}^2 \cdot f_n^{cpu} + k_2 \cdot f_n^{int}$ since power consumption during $T_{F_n}^{off1}$ is a function of the CPU voltage/frequency for data update into the destination register or I/D-Cache as well as the voltage level of the internal bus clock generator for data transfer to the CPU (assumed to be 3.3V). k_1 and k_2 are coefficients which relate $P_{F_n}^{off1}$ to the CPU

frequency/voltage and the internal bus clock frequency/voltage, respectively. P^{std}_{sys,F_n} is obtained by measuring the system power dissipation in every frequency setting for the case that the system is in the standby mode. $P^{on}_{F_n}$, which is the active component of the CPU power, is the difference between P^{std}_{sys,F_n} and the measured power when a CPU-intensive task is running. $P^{off2}_{F_n}$ is the power consumption of accessing the memory. The main memory has a total size of 64MB, comprising of two 32MB SDRAMs. For each 32MB SDRAM, we used data sheet values [16] of 446mW when the SDRAM is being accessed and 132mW when it is in the idle mode. Therefore, $P^{off2}_{F_n}$ can be calculated as 2*(446mW-132mW)/0.8 = 785mW, where factor of 0.8 represents the energy conversion efficiency of the DC-DC converter (12V to 3.3V conversion.) We tried a curve fitting procedure with measured power values to get k_1 and k_2, and found them to be 0.73 and 6.2, respectively. Extracted parameters are summarized in Table 2.

1 Extracted parameters for system energy estimation

	P^{std}_{sys,F_n} (mW)	$P^{on}_{F_n}$ (mW)	$P^{off2}_{F_n}$ (mW)
F_1	1665	86.786	785
F_2	1699	218.156	785
F_3	1732	344.091	785
F_4	1728	216.97	785
F_5	1778	377.869	785
F_6	1869	672.885	785
F_7	1963	674.912	785
F_8	1757	147.858	785*1.33
F_9	1836	335.682	785*1.33

The system energy for a task at F_n, E_{sys,F_n}, is given as:

$$E_{sys,F_n} = P^{std}_{sys,F_n} \cdot T + P^{on}_{F_n} \cdot T^{on}_{F_n} + P^{off1}_{F_n} \cdot T^{off1}_{F_n} + P^{off2}_{F_n} \cdot T^{off2}_{F_n} \qquad (11)$$

The estimated energy consumption for all tested benchmarks using eq (11) and extracted parameters over all frequency combinations were compared with the actually measured ones. The average error rate was about 3%.

4. Proposed DVFS Policy

4.1 Scaling granularity

In reality, it takes time to change the CPU frequency/voltage due to factors such as the internal PLL locking time and capacitances that exist in the voltage path. For the PXA255 processor, the latency for switching the CPU voltage/frequency is 500usec [17]. In order to safely ignore this overhead, the minimum quantum of time for scaling the CPU frequency/voltage must be at least two to three orders of magnitude larger than this switching latency. At the same time, we would like to minimize the overhead of the voltage/frequency scaling as far as the OS is concerned. Correspondingly, we use the start time of an (OS) *quantum* (approximately 60msec in Linux) used by the OS to schedule processes as DVFS decision points, that is, each time the OS invokes the scheduler to schedule processes in the next quantum, we also make a decision so as to whether or not the CPU voltage/frequency is changed, and if so, we then scale the voltage/frequency of the CPU.

4.2 Calculating the Average On-chip CPI

We calculated the W^{on} and W^{off} of a program at run time, by using the processor's PMU. The PMU unit consists of a clock counter and two other counters, each of which can monitor one of 15 different events such as cache hit/miss, TLB hit/miss, and number of executed instructions. The overhead for accessing the PMU (for both read and write operations) is less than 1usec [12] and can thus be ignored. Our approach is similar to [13], where number of memory bus transactions and executed instruction count were used to accurately estimate the on-chip CPU. Since the PMU in PXA255 does not provide the number of memory bus transactions, we have used the following three events based on extensive experiments: (i) number of instructions being executed (INSTR) and (ii) number of stall cycles due to data dependency (STALL) (iii) number of D-cache miss (DMISS).

At the end of every quantum, INSTR and STALL event statistics are read from the PMU. In addition, the number of clock cycles in a quantum (CCNT) is given by the clock counter. From these values, we calculate the average CPU clock cycles per instruction (CPI^{avg}) as CCNT/INSTR. Similarly, average number of stalls per instruction (SPI^{avg}) is calculated. SPI^{avg} accounts for both the on-chip stalls (SPI^{avg}_{on}) and the off-chip stalls (SPI^{avg}_{off}).

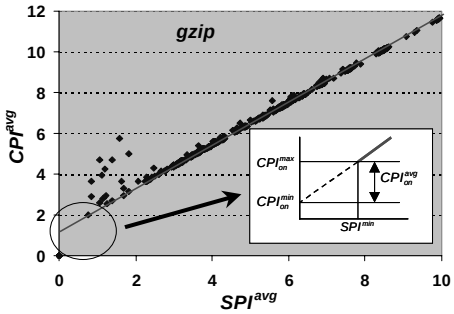


Figure 5: Contour plots of CPI^{avg} versus SPI^{avg} for different clock frequency combinations

Figure 5 shows the plot with SPI^{avg} of each quantum on the x-axis and the CPI^{avg} on the y-axis for "gzip" application. From this figure, we can easily see that CPI^{avg} is linearly related to SPI^{avg} as follows:

$$CPI^{avg} = k \cdot SPI^{avg} + c \qquad (12)$$

where k is the slope (~1). Notice that the y-intercept c is equal to the average on-chip CPI without any stall cycle, $CPI^{min}{}_{on}$.

To obtain $CPI^{avg}{}_{on}$, we need to extract $SPI^{avg}{}_{on}$ from SPI^{avg}. To do this, we consider the y-intercept of the above line, the CPI^{avg} when no data-stalls occur, as the lower bound for the on-chip CPI ($CPI^{min}{}_{on}$). The CPI^{avg} at the lowest SPI^{avg} value (SPI^{min}) is considered as the upper bound ($CPI^{max}{}_{on}$). The $CPI^{avg}{}_{on}$ is estimated from both $CPI^{min}{}_{on}$ and $CPI^{max}{}_{on}$ along with the values of DMISS of the quantum. The intuition for using DMISS to calculate $CPI^{avg}{}_{on}$ is that if the number of data cache misses is high, most of the stalls are off-chip stalls. Therefore, if the value of DMISS is high (low) then a CPI^{avg} value close to $CPI^{min}{}_{on}$ ($CPI^{max}{}_{on}$) is chosen. Let *DPI* denotes *D-cache miss count* per instruction, defined as DMISS/INSTR. We equally divided the region from $CPI^{max}{}_{on}$ to $CPI^{min}{}_{on}$, into *n* sub-regions and each region is selected with the reported *DPI* value, which results in $CPI^{avg}{}_{on} = CPI^{avg}{}_{min} + CPI^{avg}(DPI)$, where $CPI^{avg}(DPI)$ is CPI^{avg} value for the corresponding *DPI* and increases (decreases) as the *DPI* value decreases (increases).

Our DVFS approach requires three events: INSTR, STALL and DMISS. Since PXA255's PMU can only provide two event statistics at a time, the PMU must be read twice in every quantum: (INSTR, STALL) pair is read during the first half whereas (INSTR, DMISS) pair is read during the second half of every quantum.

4.3 Determining the Optimal Frequency Setting

In the proposed DVFS policy, an optimal frequency set is determined considering both timing constraints and minimum system energy consumption. As a timing constraint for non real-time applications, we use a performance loss (PF_{loss}) factor, which is defined as the increased execution time of a program due to lowered clock frequency and given as [12][13]:

$$PF_{loss} = \frac{(T_{F_n} - T_{F_{max}})}{T_{F_{max}}} \qquad (13)$$

where F_{max} is the best performance frequency combination, i.e., F_7, T_{Fn} and T_{Fmax} are the total task execution time at frequency combination of F_n and F_{max}, respectively. After obtaining the CPI^{avg} value for the current quantum *i*, $CPI^{avg}{}_{on,i}$, we calculate on-chip and off-chip execution times for this quantum, $T^{on}{}_i$ and $T^{off}{}_i$, as follows:

$$T_i^{on} = \frac{N_i \cdot CPI_{on}^{avg}}{f_{i,n}^{cpu}}, \quad T_i^{off} = T_i - T_i^{on} \qquad (14)$$

where N_i is the number of executed instructions, T_i and $f^{cpu}{}_{i,n}$ are the execution time and the CPU frequency in F_n during the quantum *i*, respectively.

$W^{on}{}_i$ and $W^{off}{}_i$ are derived from the calculated values of $T^{on}{}_i$ and $T^{off}{}_i$ based on definition 1, 2, eq.(2), and eq.(10). It is assumed that $W^{on}{}_{i+1}$ and $W^{off}{}_{i+1}$ are equal to $W^{on}{}_i$ and $W^{off}{}_i$, respectively.

An optimal frequency set for the quantum i+1, $F^{opt}{}_{i+1}$, is determined as following:

1. $\Psi = \{ F_1, \ldots, F_9 \}$, $\Gamma = \{\phi\}$, and $E_{min} = \infty$
2. *for every frequency setting* F_n *in* Ψ
3. *if* ($T_{F_n}^{i+1} \le (1 + PF_{loss}) \cdot T_{F_7}^{i}$)
4. $\Gamma = \Gamma \cup F_n$;
5. *for every frequency setting* F_n *in* Γ
6. *calculate* E_{sys,F_n} *from eq.(12)*
7. *if* ($E_{sys,Fn} \le E_{min}$)
8. $E_{min} = E_{sys,Fn}$; $F^{opt}{}_{i+1} = F_n$;

where $T_{F_n}^{i+1}$ is the expected execution time of quantum i+1 at F_n and $T_{F_7}^{i}$ is the execution time of quantum i at F_7. **)**

5. Experimental Results

We implemented the proposed policy on the BitsyX platform, which runs Linux (v2.4.17). More precisely, we wrote a software module implementing the proposed policy. This module was tied to the Linux OS scheduler in order to allow voltage scaling to occur at every context switch. To show the effectiveness of the proposed DVFS method considering system energy (SE-DVFS), we also implemented the DVFS method used in [13], which considers CPU energy only (CE-DVFS), and compared the results with each other.

To measure the power consumption of the system, we inserted a 0.125 ohm precision resistor between the external power source (~12V) and the system power line. The actual power consumption at run time was measured by using a data acquisition system which operates up to 100 KHz sampling frequency by reading voltage drop across the precision resistor [19]. Our experiments are performed on a number of applications including a common UNIX utility program, "gzip", and four representative benchmark programs available on the web [20].

Figure 6 represents the measured performance degradation with target performance loss ranging from 10% to 50% at steps of 10% for SE-DVFS. As seen in this figure, performance loss values for all programs are upper bounded by 2% and 12% for CPU-intensive and memory-intensive applications, respectively. This is true even with 50% loss target. This is because the fixed power in the target system is rather large, and therefore, to minimize the overall energy consumption, it is better to complete the tasks as soon as possible regardless of the allowed performance loss factor.

Figure 7 depicts the power consumption waveform of the BitsyX system when running "gzip" with a 30% target performance degradation factor using: (a) CE-DVFS and (b) SE-DVFS. Notice that the average power dissipation for CE-DVFS is less than that for SE-DVFS because the active power dissipation is lower for the CE-DVFS. However, the overall system energy is lower for SE-DVFS because the execution time for SE-DVFS is much shorter. In fact, for this application, SE-DVFS results in 11.4% lower energy consumption compared to CE-DVFS. Similar results for other application programs are shown in Figure 8. Notice that SE-DVFS results in an additional energy saving from 2% to 18% when compared to CE-DVFS.

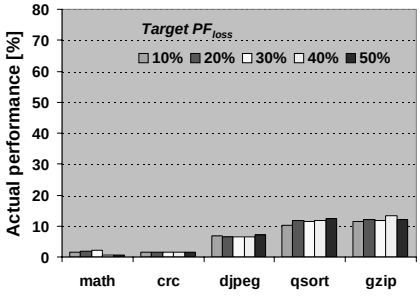
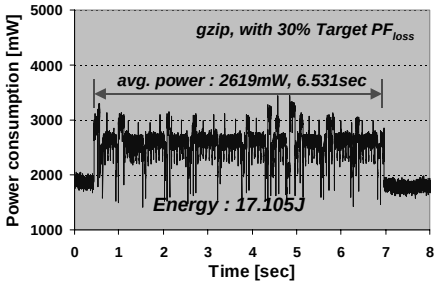
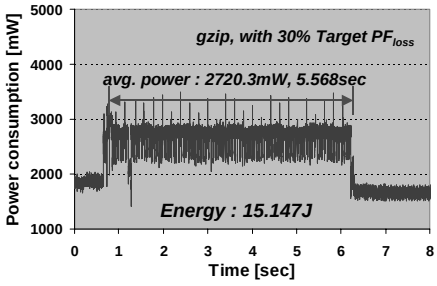


Figure 6: Actual performance: SE-DVFS



(a) CE- DVFS



(b) SE- DVFS

Figure 7: Actual power consumption of two DVFS methods

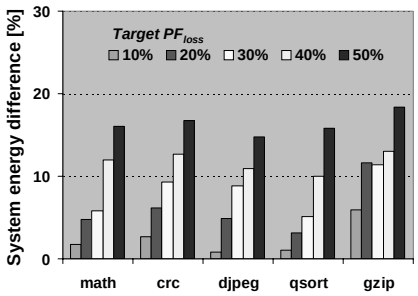


Figure 8: System energy difference: SE-DVFS vs. CE-DVFS

6. Conclusion

In this paper, a DVFS policy for the actual system energy reduction was proposed and implemented on a PXA255-based platform. In the proposed DVFS approach, a program execution time and system energy required for the program are quite accurately estimated using workload decomposition in which execution time of the program is decomposed into on-chip computation and off-chip access latencies. System power is also decomposed into variable and fixed power and very accurately estimated using decomposed execution time. The CPU voltage/frequency is scaled based on the ratio of the on-chip and off-chip latencies for each process such that both a given performance degradation factor and minimal energy consumption are satisfied. This ratio is given by a regression equation, which is dynamically updated based on runtime event monitoring data provided by an embedded performance monitoring unit. Through actual current measurements in hardware, we demonstrated that up to 18% less energy saving was achieved with the proposed DVFS compared to conventional DVFS techniques.

References

1 Developer manual: "Intel 80200 Processor Based on Intel XScale Microarchitecture," http://developer.intel.com/design/iio/manuals/273411.htm

2 "Crusoe SE Processor TM5800 Data Book v2.1," http://www.transmeta.com/everywhere/products/embedded/embedded_sefamily.html .

3 F. Yao, A. Demers, and S. Shenker, "Scheduling model for reduced CPU energy," *IEEE Annual Foundations of Computer Science*, pp.374-382, 1995.

4 Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proc. of the 36th Annual Design Automation Conference*, pp.134-139, 1999.

5 I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processor," *Proc. of the 19th IEEE Real-Time Systems Symposium*, pp.178-187, 1998.

6 D. Shin, J. Kim, and S. Lee, "Low-energy intra-task voltage scheduling using static timing analysis," *Proc. of Design Automation Conference*, pp.438-443, 2001.

7 S. Lee and T. Sakurai, "Run-time power control scheme using software feedback loop for low-power real-time applications," *Proc. of Asia-Pacific Design Automation Conference*, pp.381-386, 2000.

8 M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. of the 1st Symposium on Operating Systems Design Implementation*, pp.13-23, 1994.

9 C. Hsu and U. Kremer, "Compiler-directed dynamic voltage scaling for memory-bound applications," *Technical Report DCS-TR-498*, Department of Computer Science, Rutgers University, Aug. 2002.

10 C. Hsu and U. Kremer, "Single region vs. multiple regions: A comparison of different compiler-directed dynamic voltage scheduling approaches," *Proc. of Workshop on Power-Aware Computer Systems*, Feb. 2002.

11 S. Ghiasi, J. Casmira, and D. Grunwald, "Using IPC variation in workloads with externally specified rates to reduce power consumption," *Proc. of Workshop on Complexity Effective Design*, Jun. 2000.

12 A. Wissel and F. Bellosa, "Process Cruise Control," *CASES 2002*, Grenoble, France, Oct. 2002.

13 K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," *Proc. of Design, Automation and Test in Europe*, 2004.

14 K. Choi, R. Soma, and M. Pedram, "Off-chip latency-driven dynamic voltage and frequency scaling for an MPEG decoding," *Proc. of Design Automation Conference*, 2004.

15 http://www.applieddata.net/products_bitsyX.asp

16 http://download.micron.com/pdf/datasheets/dram/sdram/256MSDRAM_G.pdf

17 Developer's manual: "Intel XScale Microarchitecture for the PXA255 Processor"http://www.intel.com/design/pca/applicationsprocessors/manuals/278693.htm

18 User's manual: "Intel XScale Microarchitecture for the PXA255 Processor"http://www.intel.com/design/pca/applicationsprocessors/manuals/278796.htm

19 http://www.instrument.com/pci/udas.asp