

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

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 *fixed*, *idle* and *active* power components. Next, we present a new DVFS technique, which considers not only active power, but also idle and fixed power components of the system. This is in sharp contrast to previous DVFS techniques, which only consider the active power component. The fixed plus idle 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). We have implemented the proposed DVFS technique on the BitsyX platform; an Intel PXA255-based platform manufactured by ADS Inc., and performed detailed energy measurements. These measurements show that, for a number of widely used software applications, a total system energy savings of up to 18%, compared to a conventional DVFS technique that considers only variable power, is 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 customer 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 (DVFS) technique has proven to be a highly effective method of achieving low power consumption for the CPU while meeting the performance requirements. The key idea behind the DVFS technique is to dynamically scale the supply voltage level of the CPU so as to provide “just-enough” circuit speed to process the system workload while meeting total computation time and/or throughput constraints, and thereby, reduce the energy dissipation (which is quadratically dependent on the supply voltage level). A number of modern microprocessors such as Intel’s XScale [1] and Transmeta’s Cruso [2] are equipped with the DVFS functionality.

There have been extensive studies of low power system designs using DVFS techniques [3]-[11]. All of these works have focused only on the reduction of the CPU power. However, in reality, the battery lifetime of a VLSI circuit is also affected by the fixed power consumed in other components of the system. The validity of these studies for low energy system design is based on the following two assumptions: (1) execution time of a task is exactly proportional to the inverse of operating frequency, f , as $T \propto 1/f$. (2) the system consists of a single processor,

meaning the processor power, P , is equal to the system power and is proportional to the operating frequency in cubic way, i.e., $P \propto f^3$, with the first order relation between frequency and voltage.

Based on these two assumptions for performance and power, reducing processing speed always results in the system energy savings. Unfortunately, these two assumptions may not be valid since most computing systems, currently available in the market, are made up of several subsystems such as memory systems and other peripheral components for user demands. These subsystems have their own operating clock frequency and voltage levels. The heterogeneity in performance and power of the system components make it difficult to apply DVFS techniques for such systems. A number of studies [12] - [18] reported that due to asynchrony between the memory access cycle and the processor speed, CPU speed could be slowed down with little impact of the total execution time when the execution time of a task is dominated by the memory access time, which makes the assumption (1) invalid for memory-intensive applications. Furthermore, when power consumed in the subsystems is comparable to the CPU power, to slow down the processing speed might cause more system energy; even CPU energy is saved, because of the large fixed energy consumption, due to increased execution time.

This paper presents a dynamic voltage and frequency scaling (DVFS) technique that minimizes the total system energy consumption to perform a task while satisfying a given execution time constraint. To guarantee minimum energy for task execution using DVFS, 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 dissipations. Variable power captures the component of the 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 *standing* power as the summation of *fixed* plus *idle* power components of the system.

DVFS can reduce only the *active* component of system power dissipation. If this component is large compared to the *standing* components of system power, then lowering the CPU clock frequency and then reducing the supply voltage of the CPU (while meeting a task execution time deadline) will result in lower system energy consumption due to the linear relation between the CPU cycle time and voltage and 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 *standing* components, 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 *standing* power dissipation components.

In this paper, we present a new DVFS technique, which considers not only *active* power, but also *standing* power components of the system. This is in sharp contrast to previous DVFS techniques, which only consider the *active* power component. 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 [25] come equipped with.

The proposed technique has been implemented on an embedded system platform built around the PXA255 processor. Detailed energy saving results have been obtained by doing current measurements in actual hardware. On this platform, we performed a task with up to 12% less system energy compared to the case with normal DVFS techniques, which consider only variable power. For both CPU and memory-bound programs, target performance degradation was finely controlled. More precisely, an optimal CPU frequency for a task is determined such that the minimal system energy is consumed for the task.

The main contributions of our work are: (1) It presents the first implementation of a DVFS policy for the total system energy reduction, accounting for active, idle and fixed system power components. (2) It presents a highly accurate execution time model for a task at run time by using an embedded PMU. (3) It presents an accurate power consumption model of the target system based on workload decomposition within less than a 3% error. (4) Evaluation of the proposed method is performed through actual hardware measurements for a number of different applications.

The remainder of this paper is organized as follows. Related work is described in Section 2. In Section 3, models for both execution time and power dissipation using workload decomposition are described. Details of target platform and the proposed DVFS policy are presented in Section 4 and 5, respectively. Experimental results and conclusions are given in Sections 6 and 7, respectively.

2. Related Work

There have been many studies on DVFS for either real-time or non real-time operations. They can be divided into many categories, either inter-task [4][5][6][7] and intra-task [8][9] depending on the scaling granularity or application-specific [8][9] and application non-specific [3][5][6][7][10] depending on the modification of the application itself or not.

However, all these approaches solely focused on the CPU energy saving, based on the two assumptions mentioned in the previous section; inverse relationship between execution time versus operating frequency and cubic relationship between the system power and operating frequency.

There are different DVFS approaches that make use of the asynchrony of memory access to the CPU clock during a task execution. In [12] and [13], compiler-assisted DVFS approaches were proposed, in which frequency is lowered in memory-bound region of a program with little performance degradation. DVFS approaches that rely on micro-architecture or embedded hardware without any assistance from a compiler or a simulator have also been reported. In [14] a microarchitecture-driven DVFS technique was proposed in which cache miss drives the voltage scaling. In [15] IPC (instruction per cycle) rate of a program execution was used to guide the voltage scaling. Reference [16] presented a policy to choose the optimal CPU clock frequency under a fixed performance degradation constraint (of say 10%) based on dynamic program behavior such as the number of executed instructions and memory access counts during the whole execution time by using a performance monitoring unit (PMU). In [17] a DVFS technique which enables more precise energy-performance trade-off using PMU was 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 [18].

These approaches using a PMU require no help from either off-line simulation or compiler and used dynamic event counts from the PMU only. These approaches also considered CPU energy reduction only without noticing the existence of fixed power portion in the system power.

There are some works considering fixed power occurred by subcomponents in using DVFS techniques. References [19] and [20] have suggested that power consumption in the memory effects should be taken into account. Reference [21] reported that there is a lower bound on the CPU frequency such that any further slowing down degrades the amount of computation that can be performed per battery discharge. The authors also allude to the problem that the high cost of memory may dominate the total energy consumption of a system such that even effective DVFS for the CPU energy saving might be less effective in terms of the system energy.

3. Workload Decomposition

3.1 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. It 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. Some of these events result in a small overhead (e.g., cache hit) while others give rise to a large penalty due to external memory access (e.g., cache miss). During an off-chip access, which is asynchronous with respect to the CPU clock, the CPU stalls until the requested memory transaction is completed. Thus, CPU clock cycles during off-chip access are wasted without doing any useful work. Furthermore, the off-chip access time is solely determined by external access clock cycle, not by the CPU clock cycle. To illustrate the key point of the workload decomposition for the system energy reduction, we define two different types of workload: on-chip and off-chip workload.

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 (see discussion about out-of-order execution processors later in this section.).

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:

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

where CPI_{on}^{avg} denotes the number of CPU clock cycles per on-chip instruction, M is the number of off-chip accesses, and CPI_{off}^{avg} denotes the number of external clock cycles per an off-chip access. See Section 5.2 for how to calculate these two CPI's. From these two definitions, the execution time, T , for a task is calculated as:

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

Notice that this breakdown of the total execution time is not exact when the target processor supports out-of-order execution whereby

instructions after the instruction that caused an off-chip access may be executed during the off-chip access. In such a case, T^{on} and T^{off} can overlap. However, in practice, the error introduced in this way is quite small considering that the memory access time is about two orders of magnitude greater than the instruction execution time. Therefore, out-of-order execution does not cause a large error in eq. (2).

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

$$\frac{\Delta T}{\Delta f_{cpu}} = \frac{\Delta T^{on}}{\Delta f_{cpu}}, \quad \frac{\Delta T^{off}}{\Delta f_{cpu}} \approx 0 \quad (3)$$

3.2 Modeling the System Power Consumption

We consider a computing system consisting of a CPU with a variable operating frequency, f_n , where $f_{min} \leq f_n \leq f_{max}$. Let P_{cpu,f_n} denote the CPU power dissipation at f_n . The system also includes N system modules. Let $P_{mod,i}$ denote the power dissipation of the i^{th} module. Then, we can write the following:

$$\begin{aligned} P_{cpu,f_n} &= P_{cpu,f_n}^{std} + P_{cpu,f_n}^{act} = P_{cpu}^{fix} + P_{cpu,f_n}^{idle} + P_{cpu,f_n}^{act} \\ P_{mod,i} &= P_{mod,i}^{std} + P_{mod,i}^{act} = P_{mod,i}^{idle} + P_{mod,i}^{act} \end{aligned} \quad (4)$$

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}^{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, $P_{mod,i}^{idle}$. Here, it is assumed that the idle component and the fixed component of power dissipation in a system module are the same as one another because, generally speaking, the operating clock and voltage for the modules are not dynamically varied, but they remain fixed. The required system energy to complete a task in time T with a CPU clock frequency of f_n is given by:

$$E_{sys,f_n} = \int_{t_0}^{t_0+T} P_{sys,f_n}(t) \cdot dt \quad (5)$$

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

$$\begin{aligned} P_{sys,f_n}(t) &= P_{sys}^{fix} + P_{sys,f_n}^{idle} + P_{sys,f_n}^{act}(t) = P_{sys,f_n}^{std} + P_{sys,f_n}^{act}(t) \\ &= P_{cpu,f_n}^{std} + \sum_{i=1}^N P_{mod,i}^{std} + P_{cpu,f_n}^{act}(t) + \sum_{i=1}^N P_{mod,i}^{act}(t) \end{aligned} \quad (6)$$

Here, $\left(P_{cpu,f_n}^{std} + \sum_{i=1}^N P_{mod,i}^{std} \right)$ denotes the standing system power consumption, P_{sys,f_n}^{std} , whereas $\left(P_{cpu,f_n}^{act}(t) + \sum_{i=1}^N P_{mod,i}^{act}(t) \right)$ denotes the active system power consumption, $P_{sys,f_n}^{act}(t)$. Generally speaking, it is difficult to accurately calculate $P_{sys,f_n}^{act}(t)$ because the power requirement for each instruction is different. For example, considering instructions in the dynamic trace of an application program running on the system, the CPU is used to execute the on-chip workload, whereas the memory is required to execute the off-chip workload. When on-chip and off-chip workload are executed randomly during the program execution, $P_{sys,f_n}^{act}(t)$ should be severely fluctuating as shown in Figure 1 (a). However, once workload of a task is decomposed into on-chip and off-chip, $P_{sys,f_n}(t)$ can be modeled as:

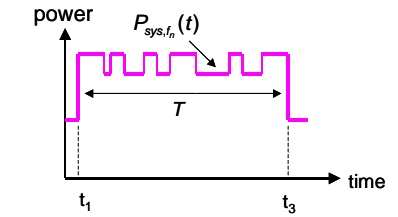
$$P_{sys,f_n}(t) = \begin{cases} P_{cpu,f_n}^{act}(t) + P_{sys,f_n}^{std}, & \text{during } T^{on} \\ \sum_{i=1}^N P_{mod,i}^{act}(t) + P_{sys,f_n}^{std}, & \text{during } T^{off} \end{cases} \quad (7)$$

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

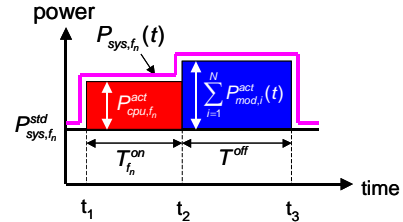
Hence, E_{sys,f_n} after workload decomposition is given as:

$$E_{sys,f_n} = \left[P_{cpu,f_n}^{act} + P_{sys,f_n}^{std} \right] \cdot T^{on} + \left[\sum_{i=1}^N P_{mod,i}^{act} + P_{sys,f_n}^{std} \right] \cdot T^{off} \quad (8)$$

In eq.(8), P_{cpu,f_n}^{act} and P_{sys,f_n}^{std} can easily be obtained from simple measurements on the target system by performing benchmark programs with different CPU frequencies, but it is difficult to get $\sum_{i=1}^N P_{mod,i}^{act}(t)$ because complete information about system components usage by the target program is not available at run time. In practice, $\sum_{i=1}^N P_{mod,i}^{act}(t)$ is approximated by memory power for general applications used in this paper 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_{sys,f_n}^{std} .



(a) without workload decomposition



(b) with workload decomposition

Figure 1: System power consumption during task execution

3.3 System Energy vs. CPU Frequency

As shown in eq.(8), E_{sys,f_n} is a function of various parameters of the system configuration (P_{cpu,f_n}^{act} , P_{sys,f_n}^{std} , and $P_{mod,i}^{act}$) and application program (T^{on} and T^{off}). Depending on these parameters, an optimal CPU frequency which results in task execution with minimum system energy consumption is determined as explained next.

The system energy equation (8) is rewritten as:

$$E_{sys,f_n} = P_{cpu,f_n}^{act} \cdot T^{on} \cdot \left[1 + \left(\frac{P_{sys,f_n}^{std}}{P_{cpu,f_n}^{act}} \right) + \left(\frac{P_{mem}^{act} + P_{sys,f_n}^{std}}{P_{cpu,f_n}^{act}} \right) \cdot \left(\frac{T^{off}}{T^{on}} \right) \right] \quad (9)$$

where P_{mem}^{act} is memory power and used instead of $\sum_{i=1}^N P_{mod,i}^{act}(t)$ in eq.(8)

as mentioned before. The case in which P_{sys,f_n}^{std} and P_{mem}^{act} 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 standing power. In that case, lowering CPU frequency always results in system energy saving. Assuming a linear relationship between the operating voltage and frequency ($P_{cpu,f_n}^{act} \propto f_n^3$ and $T^{on} \propto f_n^{-1}$), then E_{sys,f_n} becomes dependent upon f_n as following form:

$$E_{sys,f_n} = a \cdot f_n^2 + b \cdot f_n^{-1} + c \quad (10)$$

where a , b , and c are constant coefficients. In particular, b and c represent the amounts of standing 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.(10). If b is zero, then f_{opt} is f_{min} , but f_{opt} increases as b increases.

4. Description of the Target System

4.1 BitsyX Platform

Our target system for DVFS is the BitsyX system from ADS Inc. [22]. BitsyX has a PXA255 microprocessor which is a 32-bit RISC processor core, with a 32KB instruction cache and a 32KB write-back data cache, a 2KB mini-cache, a write buffer, and a memory management unit (MMU) combined in a single chip. It 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).

Table 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

4.2 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 provides the execution time of all the 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. The same observations can be made by examining settings F_4 , F_5 , and F_6 , which are again only different from each other in terms of the CPU clock frequency.

It should be noted that when frequency scaling is performed, not only f^{cpu} is changed but also f^{int} and f^{ext} are scaled in BitsyX. Therefore, the effect of f^{int} and f^{ext} on the total program execution time should also be considered. Execution time T is sum of T^{on} and T^{off} as in eq.(2) and

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:

$$T^{off} = T^{off1} + T^{off2} = \frac{\alpha \cdot W^{off}}{f^{int}} + \frac{(1-\alpha) \cdot W^{off}}{f^{ext}} \quad (11)$$

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.

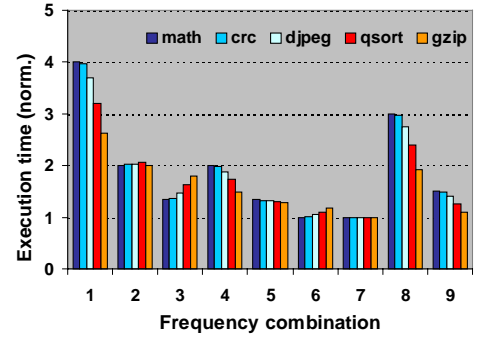


Figure 2: Execution time variation over different frequency combinations

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 as shown in Figure 3.

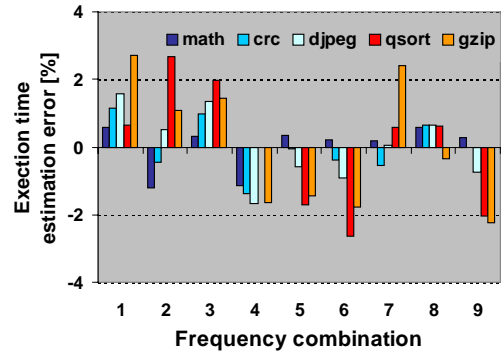


Figure 3: Execution time estimation error

4.3 Energy Consumption Model for BitsyX

Measured energy consumptions for each application are presented in Figure 4. The frequency combination at which the minimum energy is consumed is not F_1 which has the minimum CPU frequency of 100MHz for all tested applications. On the contrary, F_1 causes the largest system energy among all frequency combinations. Energy trend according to frequency sets is similar to execution time variation in Figure 2, i.e., less execution time cause less system energy. This result is due to the fact that there is the standing component in the total system power during task execution and it should be better to finish a program as soon as possible for less system energy.

One more important observation in Figure 4 is that this minimal energy frequency is varying depending on applications, either CPU-intensive or memory-intensive. For example, F_6 (CPU frequency of 400MHz) gives minimum energy for the “math” which is the most CPU-intensive, whereas F_9 (CPU frequency of 265MHz) does for the “gzip” which is the most memory-intensive. The reason why F_9 is the best for “gzip” is that F_9 has the fastest condition for memory access operation, both memory clock and internal bus clock are 133MHz.

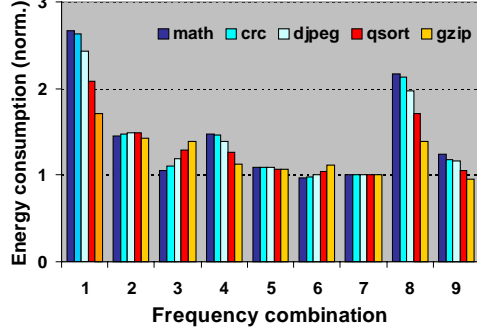


Figure 4: Energy consumption over different frequency combinations

Energy consumption model in BitsyX is shown in Figure 5. Terms used in this figure are explained next.

- F_n : the n^{th} frequency setting, $F_n(f_n^{\text{cpu}}, f_n^{\text{int}}, f_n^{\text{ext}})$
- $T_{F_n}^{\text{on}}$: on-chip computation time at F_n
- $T_{F_n}^{\text{off1}}$: data update time after fetch from memory at F_n
- $T_{F_n}^{\text{off2}}$: data fetch time from memory at F_n
- $P_{\text{sys},F_n}(t)$: time-varying system power at F_n
- $P_{\text{sys},F_n}^{\text{std}}$: standing power in $P_{\text{sys},F_n}(t)$
- $P_{F_n}^{\text{on}}$: active power in P_{sys,F_n} during $T_{F_n}^{\text{on}}$
- $P_{F_n}^{\text{off1}}$: active power in P_{sys,F_n} during $T_{F_n}^{\text{off1}}$
- $P_{F_n}^{\text{off2}}$: active power in P_{sys,F_n} during $T_{F_n}^{\text{off2}}$
- V_{F_n} : CPU operating voltage at F_n
- k_1 : fitting coefficient for $P_{F_n}^{\text{off1}}$, [nF]
- k_2 : fitting coefficient for $P_{F_n}^{\text{off2}}$, [$V^2 \cdot \text{nF}$]

Here, $P_{F_n}^{\text{off1}}$ is represented as $k_1 \cdot V_{F_n}^2 \cdot f_n^{\text{cpu}} + k_2 \cdot f_n^{\text{int}}$ since power consumption during $T_{F_n}^{\text{off1}}$ is a function of the CPU voltage/frequency for data update into the destination register or I/D-Cache and 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 relates $P_{F_n}^{\text{off1}}$ with CPU frequency/voltage and internal bus clock frequency/voltage, respectively. $P_{\text{sys},F_n}^{\text{std}}$ is obtained by measuring the system power in all frequency settings when the system is idle. $P_{F_n}^{\text{on}}$, which is the active component of the CPU power, is the difference between $P_{\text{sys},F_n}^{\text{std}}$ and the measured power when a CPU-intensive task is running. $P_{F_n}^{\text{off2}}$ 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 [23] of 446mW when the SDRAM is being accessed and 132mW when it is in the idle mode. Therefore, $P_{F_n}^{\text{off2}}$ can be calculated as $2 \cdot (446\text{mW} - 132\text{mW}) / 0.8 = 785\text{mW}$, where factor of 0.8 represents the efficiency factor for the DC-DC converter (12V to 3.3V conversion). We performed 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.

Table 2. Extracted parameters for system energy estimation

	$P_{\text{sys},F_n}^{\text{std}}$ (mW)	$P_{F_n}^{\text{on}}$ (mW)	$P_{F_n}^{\text{off2}}$ (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	$785 \cdot 1.33$
F_9	1836	335.682	$785 \cdot 1.33$

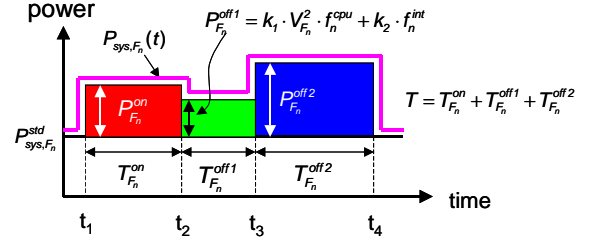


Figure 5: Total system power consumption during execution

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

$$E_{\text{sys},F_n} = P_{\text{sys},F_n}^{\text{std}} \cdot T + P_{F_n}^{\text{on}} \cdot T_{F_n}^{\text{on}} + P_{F_n}^{\text{off1}} \cdot T_{F_n}^{\text{off1}} + P_{F_n}^{\text{off2}} \cdot T_{F_n}^{\text{off2}} \quad (12)$$

Figure 6 shows the estimated energy consumption for “jpeg” using eq (12) and extracted parameters over all frequency combinations and compared these estimated energy values with the actually measured ones. The average error rate for “jpeg” is less than 4% and for other applications, the error rate is about 3%.

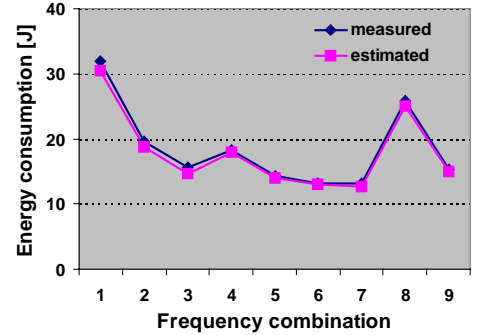


Figure 6: Accuracy of the proposed models for power consumption and execution time

5. Proposed DVFS Policy

5.1 Scaling granularity

The ideal DVFS is supposed to instantaneously change the voltage/frequency values. In reality, however, it takes time to change the CPU frequency/voltage due to some factors such as the internal PLL (phase lock loop) locking time and capacitances that exist in the voltage path. For the PXA255 processor, the latency for switching the CPU voltage/frequency is 500usec [24]. In order to safely ignore this scale 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.

5.2 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 including 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 [16] and can thus be ignored. Our approach is similar to [17], 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 support for counting 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 along with the number of clock counts in a quantum (CCNT) which is given by the clock counter are read from the PMU. 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 7 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:

$$CPI^{avg} = k \cdot SPI^{avg} + c \quad (13)$$

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

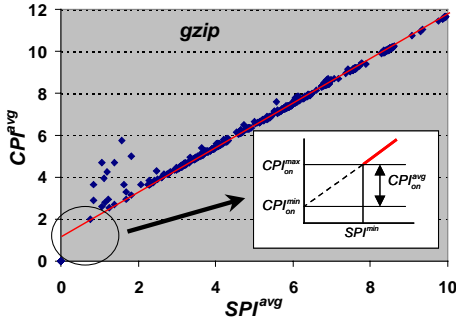


Figure 7: Contour plots of CPI^{avg} versus SPI^{avg} for different clock frequencies combinations

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 $CPI^{avg}_{on} = CPI^{min}_{on} + CPI^{avg}(DPI)$, 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.

5.3 Determining the Optimal Frequency Setting

In the proposed DVFS policy, an optimal frequency is determined considering both the timing constraints and minimum system energy consumption. As a timing constraint for non real-time applications, we used performance loss (PF_{loss}) which is defined as the increased execution time of a program due to lowered clock frequency and given as [16][17]:

$$PF_{loss} = \frac{(T_{F_n} - T_{F_{max}})}{T_{F_{max}}} \quad (14)$$

where F_{max} is the best performance frequency combination, i.e., F_7 , T_{F_n} and $T_{F_{max}}$ 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_i^{on} and T_i^{off} , as follows:

$$T_i^{on} = \frac{N_i \cdot CPI^{avg}_{on}}{f_{i,n}^{cpu}}, \quad T_i^{off} = T_i - T_i^{on} \quad (15)$$

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

W_i^{on} and W_i^{off} are derived from the calculated values of T_i^{on} and T_i^{off} based on definition 1, 2, eq.(2), and eq.(11). It is assumed that W_{i+1}^{on} and W_{i+1}^{off} are equal to W_i^{on} and W_i^{off} , respectively.

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

1. $\Psi = \{F_1, \dots, F_9\}$, $\Gamma = \{\phi\}$, and $E_{min} = \infty$
2. for every frequency setting F_n in Ψ
3. if ($T_{F_n}^{i+1} \leq (1 + PF_{loss}) \cdot T_{F_n}^i$)
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 ($E_{sys,F_n} \leq E_{min}$)
8. $E_{min} = E_{sys,F_n}$; $F^{opt}_{i+1} = F_n$;

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

6. Experimental Results

We implemented the proposed policy on the BitsyX platform, which runs Linux (v2.4.17). Precisely speaking, we wrote a software module implementing the proposed policy. This module is 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 [17], which considers CPU energy only (CE-DVFS), and compared the results each other.

To measure the power consumption of the system, we inserted a 0.125 ohm precision resistor between the external power source ($\sim 12V$) 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 [26]. 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 [27].

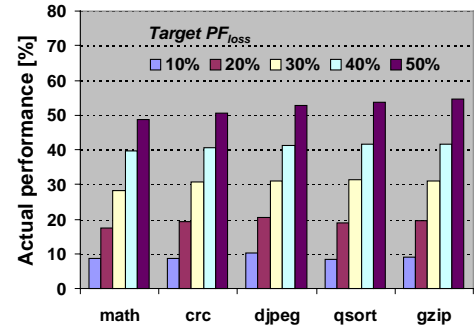
Figure 8 represents the measured performance degradation with target performance loss ranging from 10% to 50% at steps of 10% for both CE-DVFS and SE-DVFS. As seen in this figure, using CE-DVFS in case (a), we obtained actual performance loss values very close to the target values for all programs (i.e., actual average within 1.5% of the target), whereas performance loss values obtained using SE-DVFS are saturated to 2% and 12% for CPU-intensive and memory-intensive applications, respectively even with 50% target. This is due to significant fixed power in the target system and it is better to finish task as soon as possible in terms of total system energy saving.

Figure 9 shows the achieved system energy saving with (a) CE-DVFS and (b) SE-DVFS running benchmark programs at various performance loss values. System energy saving is calculated by comparing measured system energy applying DVFS method with that of without any DVFS method case in which programs were run at F_7 . From this figure, it is found that system energy increased for all applications by applying CE-DVFS, whereas there are energy savings for “math”, “crc”, and “gzip” programs in case of SE-DVFS and little changes in the system energy are observed in “djpeg” and “qsort”. These results are corresponding to data in Figure 4. For example, the frequency set for minimum energy consumption of “djpeg” is F_6 and CPU frequency of F_6 , 400MHz, is the same as in F_7 . CE-DVFS does not consider the system energy, but only concerns timing constraint. So, as target performance increases, less frequency set is chosen by CE-DVFS, resulting in system energy increase. This is not the case in SE-DVFS so that minimum system energy is maintained by using SE-DVFS.

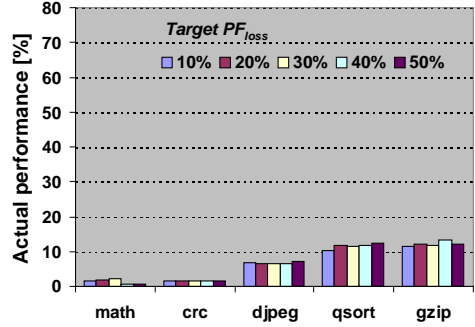
Figure 10 depicts the power consumption waveform of the BitsyX system when running “gzip” with 30% target performance degradation factor using: (a) CE-DVFS and (b) SE-DVFS. In case (a) using CE-DVFS, the average power is less than that of case (b) since active power is reduced. But, due to increased fixed energy by lowered frequency, total system energy increased in case of CE-DVFS. For this application, SE-DVFS requires 11.4% less system energy than that of CE-DVFS. For other applications with different performance target values are shown in Figure 11 and about 2% to 18% of total system energy is reduced by using SE-DVFS compared with the results of using CE-DVFS. From these measurements, we conclude that our proposed SE-DVFS technique is quite helpful to extend the whole system lifetime.

7. 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 12% less energy saving was achieved with the proposed DVFS compared with the results in the previous DVFS techniques. For both CPU and memory-bound programs, given timing constraints were also satisfied.

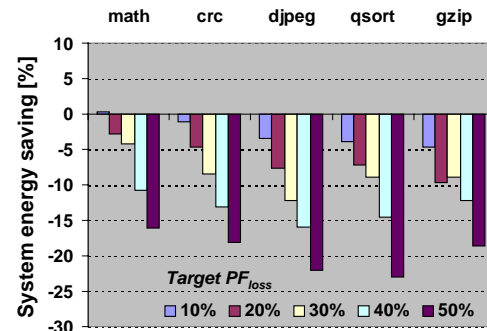


(a) conventional DVFS (CE-DVFS)

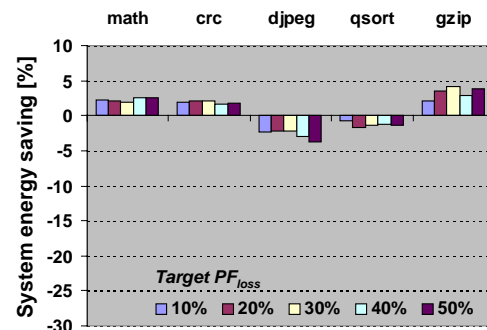


(b) proposed DVFS (SE-DVFS)

Figure 8: Actual performance: CE-DVFS and SE-DVFS

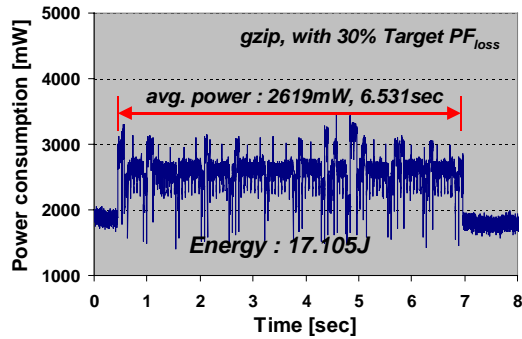


(a) CE- DVFS

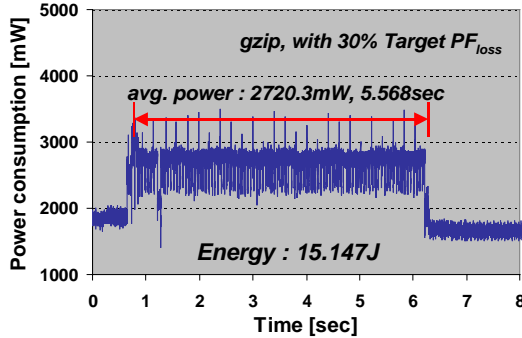


(b) SE- DVFS

Figure 9: System energy saving: SE-DVFS and CE-DVFS



(a) CE- DVFS



(b) SE- DVFS

Figure 10: Actual power consumption of two DVFS methods

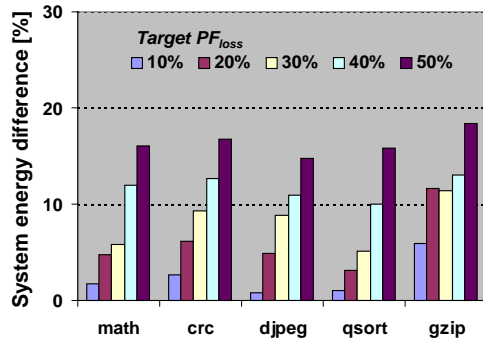


Figure 11: System energy difference: SE-DVFS vs. CE-DVFS

8. REFERENCES

- [1] Developer manual: "Intel 80200 Processor Based on Intel XScale Microarchitecture," <http://developer.intel.com/design/iao/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] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. of The International Symposium on Low Power Electronics and Design*, Monterey, pp.197-202, Aug. 1998.
- [7] G. Quan and X. Hu, "Minimum energy fixed-priority scheduling for variable voltage processors," *Proc. of Design Automation and Test in Europe*, pp.782-787, Mar. 2002.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low power CPU," *Proc. of the 1st ACM Int'l Conference on Mobile Computing and Networking*, pp.13-25, 1995.
- [12] 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.
- [13] 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.
- [14] D. Marculescu, "On the use of microarchitecture-driven dynamic voltage scaling," *Proc. of Workshop on Complexity-Effective Design*, Jun. 2000.
- [15] 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.
- [16] A. Wissel and F. Bellosa, "Process Cruise Control," *CASES 2002*, Grenoble, France, Oct. 2002.
- [17] 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.
- [18] 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.
- [19] T. Martin, "Balancing batteries, power and performance: System issues in CPU speed-setting for mobile computing," PhD thesis, Carnegie Mellon University, 1999.
- [20] T. L. Martin, D. P. Siewiorek, and J. M. Warren, "A CPU speed-setting policy that accounts for nonideal memory and battery properties," *Proc. 39th Power Sources Conference*, pp.502-505, Jun. 2000.
- [21] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," *The 7th Annual International Conference on Mobile Computing and Networking 2001*, pp.251-259, 2001.
- [22] http://www.applieddata.net/products_bitsyX.asp
- [23] http://download.micron.com/pdf/datasheets/dram/sdram/256MSDRAM_G.pdf
- [24] Developer's manual: "Intel XScale Microarchitecture for the PXA255 Processor" <http://www.intel.com/design/pca/applicationsprocessors/manuals/278693.htm>
- [25] User's manual: "Intel XScale Microarchitecture for the PXA255 Processor" <http://www.intel.com/design/pca/applicationsprocessors/manuals/278796.htm>
- [26] <http://www.instrument.com/pci/udas.asp>
- [27] <http://www.eecs.umich.edu/mibench>