Dynamic Voltage and Frequency Scaling based on Workload Decomposition*

Kihwan Choi, Ramakrishna Soma, and Massoud Pedram
Department of EE-Systems, University of Southern California, Los Angeles, CA 90089
{kihwanch, rsoma, pedram}@usc.edu

ABSTRACT
This paper presents a technique called “workload decomposition” in which the CPU workload is decomposed in two parts: on-chip and off-chip. The on-chip workload signifies the CPU clock cycles that are required to execute instructions in the CPU whereas the off-chip workload captures the number of external memory access clock cycles that are required to perform external memory transactions. When combined with a dynamic voltage and frequency scaling (DVFS) technique to minimize the energy consumption, this workload decomposition method results in higher energy savings. The workload decomposition itself is performed at run time based on statistics reported by a performance monitoring unit (PMU) without a need for application profiling or compiler support. We have implemented the proposed DVFS with workload decomposition 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 CPU energy saving of 80% can be achieved for memory-bound programs while satisfying the user-specified timing constraints.

Categories and Subject Descriptors
J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms
Algorithms, Measurement, Experimentation

Keywords
Dynamic voltage and frequency scaling, workload decomposition.

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 scaling (DVFS) technique has proven to be a highly effective method of achieving low power consumption while meeting the performance requirements. The key idea behind 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.

The workload of a task is often represented by the number of CPU clock cycles required to complete the task and is either given in advance for hard real-time operation or is predicted at run-time for soft-real time operation such as multimedia processing. In both cases, however, the question of workload composition in terms of the CPU-bound versus memory-bound instructions is often overlooked. Recall that the main memory is asynchronous with the processor and often has its own clock. Now if the execution time of a task is dominated by the memory access time, then the CPU speed can be slowed down with little impact on the total execution time of that task. However, this can result in significant CPU energy saving.

The decomposition of CPU workload of a task can be done either statically using off-line profiling and compiler support or dynamically using a performance monitoring unit (PMU), which most modern processors such as XScale80200 [1] or PXA255 [3] come equipped with. In this paper, we present a Dynamic Voltage and Frequency Scaling based on Workload Decomposition (DVFS-WD). The proposed technique has been implemented on an embedded system platform built around the PXA255 processor. Detailed energy savings results have been obtained by doing current measurements in actual hardware. On this platform, we achieved energy saving (CPU+memory) of 20-40% with 10-30% performance loss for CPU-bound applications, whereas 10-20% saving was achieved for memory-bound applications. Considering CPU energy saving only, about 80% saving can be possible for memory-bound programs. For both CPU and memory-bound programs, target performance degradation was finely controlled.

The main contributions of our paper are as follows. 1) We propose a DVFS technique for saving energy consumption in which the workload of a task is dynamically decomposed into on-chip and off-chip by using an embedded hardware unit in the processor. 2) We show how to accurately and efficiently calculate the ratio of on-chip computation time to off-chip access times by using information about the data cache misses and the CPU stall cycles due to data dependencies. 3) We present a simple timing model for calculating the off-chip access overhead in terms of internal bus and external memory access clock cycle times. 4) We implement the proposed DVFS-WD policy on a popular hardware platform and report hardware-based measurements of energy consumption savings for a number of common applications under different timing constraints.

The remainder of this paper is organized as follows. Related work is described in Section 2. In Section 3 and 4, a new DVFS policy is presented. Details of the implementation, including both hardware

* This research was supported in part by DARPA PAC/C program under contract DAAB07-02-C-P302 and by NSF under grant no. 9988441.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISLPED’04, August 9-11, 2004, Newport Beach, California, USA.
Copyright 2004 ACM 1-58113-929-2/04/0008…$5.00.

and software, are described in Section 5. Experimental results and conclusions are given in Sections 6 and 7, respectively.

2. RELATED WORKS

Previous DVFS works may be divided into two categories based on the scaling granularity: Inter-task and intra-task voltage scheduling. Inter-task voltage scheduling determines the supply voltage on task-by-task basis, i.e., coarse-grained, while intra-task voltage scheduling adjusts the supply voltage within an individual task boundary, i.e., fine-grained. Many scheduling policies for hard real-time applications are classified as inter-task scheduling and multi-task scheduling in the operating system (OS) is the focus of [4][5][6][7][8]. More precisely, scheduling is performed at task level by the OS so as to reduce energy consumption while meeting hard timing constraints for each task. There are also a number of studies that implement intra-task DVFS as part of compile-time optimization or by modifying application program itself. In [9], an intra-task voltage scheduling technique was proposed in which the application code is divided into many segments and the worst-case execution time of each segment (which is obtained from a static timing analysis) is used to determine a suitable voltage for the next segment. In [10] a method based on a software feedback loop was proposed. In this method, a deadline for each time slot is provided.

There are different DVFS approaches that make use of the asynchrony of memory access to the CPU clock during task execution. In [11] and [12], compiler-assisted DVFS techniques 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 [13] a microarchitecture-driven DVFS technique was proposed in which cache miss drives the voltage scaling. In [14] IPC (instruction per cycle) rate of a program execution was used to guide the voltage scaling. Reference [15] 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 [16], 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.

In this paper, we propose a DVFS policy for non real-time applications, which makes use of the same ratio of on-chip to off-chip workload. However, our work is different from that of [16] in a number of key points. First, the platform used in [16] was the Apollo Testbed II which uses an Intel's XScale processor, an Intel 80312 memory controller with separate controllers (USB, PCMCIA, FireWire, etc.) connected to the CPU through a PCI bus interface. In contrast, our experimental platform - BitsyX designed and built by ADS Inc. [17] - is built around the Intel's PXA255 processor with integrated controllers and no PCI bus. This platform is more representative of a typical low-power, embedded system. Second, in [16], the external bus clock frequency was fixed at 100 MHz (for SDRAM access) or 33 MHz (for frame buffer access through the PCI bus.) In contrast, in BitsyX platform, the memory bus clock frequency also is scaled in synchrony with the CPU frequency. As a result, the problem we must solve here is more complex than the one solved in [16]. More precisely, we must estimate the on-chip and off-chip execution times differently in order to account for the synchronized scaling effect of the memory bus frequency. Third, the PMU in Xscale processor reports the number of off-chip accesses directly and so the task of estimating the off-chip execution time in [16] was rather straightforward. Unfortunately, the PXA255's PMU does not provide this information. Instead it provides other statistics that can only indirectly capture the required off-chip event count. More precisely, we propose a novel technique whereby the off-chip execution time is calculated based on two separate events: the data cache miss count and the CPU stall cycle count. Finally, whereas in the XScale processor case two events were sufficient to differentiate between on-chip and off-chip workloads, in the PXA255 case, we ought to monitor three different events. However, the PXA255's PMU only reports two events. So we present a scheme whereby the events of interests are read in a time-multiplexed fashion.

3. WORKLOAD DECOMPOSITION

3.1 Energy-performance trade-offs

A software program consists of a stream of instructions to be executed. Execution time of the program can be represented in terms of the CPI, the number of instructions being executed, and the CPU frequency as follows [18]:

$$T = \frac{\sum_{i=1}^{n} CPI_i}{f^{CPU}} \tag{1}$$

where n is the total number of instructions in the instruction stream, CPI_i is the number of CPU clock cycles for the ith instruction, and f^{CPU} is the CPU frequency. Note that as we will show later, eq. (1) is only valid for CPU-intensive application programs.

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 branch misprediction, and the *off-chip stall* cycle count due to instruction/data (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. Thus, the workload of a program, W, can be written as:

$$\begin{aligned} W &= N \cdot CPI^{avg} \\ &= N \cdot (CPI_0 + CPI^{avg}_{branch_miss} + CPI^{avg}_{stall_onchip} + CPI^{avg}_{stall_offchip}) \end{aligned} \tag{2}$$

where N is the number of instructions, CPI_0 is the ideal CPI which is 1 for a single-issue general-purpose microprocessor, $CPI^{avg}_{branch_miss}$ denotes the number of CPU clock cycles due to branch misprediction overhead, and $CPI^{avg}_{stall_onchip}$ and $CPI^{avg}_{stall_offchip}$ denote the numbers of CPU clock cycles due to on-chip stalls and off-chip stalls, respectively.

During an off-chip access (which is asynchronous with respect to the CPU clock), the CPU stalls until the requested memory transactions are completed. Thus, $N*CPI^{avg}_{stall_offchip}$ CPU clock cycles are wasted without doing any useful work. Furthermore, the off-chip access time is solely determined by the external access clock cycle, not by the CPU clock cycle. Considering this fact, it is obvious that eq. (1) does not hold for memory-intensive applications in which frequent memory accesses occur.

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 complete 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 the def. 1 and 2 and eq. (2), W^{ON} and W^{OFF} are written as:

$$W^{ON} = N \cdot CPI_{on}^{avg} = N \cdot (CPI_0 + CPI_{branch_miss}^{avg} + CPI_{stall_onchip}^{avg}) \qquad (3)$$
$$W^{OFF} = N \cdot CPI_{stall_offchip}^{avg} = M \cdot CPI_{off}^{avg}$$

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:

$$T = T^{ON} + T^{OFF} = \frac{N \cdot CPI_{on}^{avg}}{f^{CPU}} + \frac{M \cdot CPI_{off}^{avg}}{f^{EXT}} \qquad (4)$$

Notice that this breakdown of the total execution time is inexact when the target processor supports out-of-order execution whereby instructions after the instruction that has 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 tends to be quite small considering that the external 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. (4). When the CPU frequency changes, the change in T is solely due to T^{ON}:

$$\frac{\Delta T}{\Delta f^{CPU}} = \frac{\Delta T^{ON}}{\Delta f^{CPU}}, \quad \frac{\Delta T^{OFF}}{\Delta f^{CPU}} \approx 0 \qquad (5)$$

The increased execution time of a program due to lowered clock frequency represents the performance loss (PF_{loss}), which is defined as follows:

$$PF_{loss} = \frac{T_{f^{CPU}}}{T_{f^{CPU}_{max}}} - 1 \qquad (6)$$

where f^{CPU}_{max} is the maximum frequency of the CPU, $T_{f^{CPU}}$ and $T_{f^{CPU}_{max}}$ are the total task execution times at CPU frequencies of f^{CPU} and f^{CPU}_{max}, respectively. From eq. (4) and (6), the optimal frequency, f^{CPU}_{target}, for a given PF_{loss} value is calculated as follows:

$$f^{CPU}_{target} = \frac{f^{CPU}_{max}}{1 + PF_{loss} \cdot \left[1 + \left(\frac{T^{OFF}}{T^{ON}}\right) \cdot \left(\frac{f^{CPU}_{max}}{f^{CPU}}\right)\right]} \qquad (7)$$

Notice that f^{CPU}_{target} denotes the target frequency for the next time slot whereas f^{CPU} is the CPU frequency of the current time slot. From the above equation, f^{CPU}_{target} is closely related to the ratio of T^{OFF} and T^{ON} of a program. Consequently, accurate calculation of T^{OFF} *and* T^{ON}, *i.e.,* W^{OFF} *and* W^{ON}, is quite important to the effectiveness of our proposed DVFS approach. We will show how this calculation can be done online for the BitsyX system with PXA255 as the main processor (cf. Section 4.)

3.2 Scaling granularity

The ideal DVFS can instantaneously change the voltage/frequency values. In reality, however, it takes time to change the CPU frequency/voltage due to 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 500 μsec [19]. 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. Therefore, 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 to decision as to whether or not the CPU voltage/frequency is changed, and if so, we then scale the voltage/frequency of the CPU.

4. SYSTEM DESCRIPTION

4.1 BitsyX

Our target system for DVFS is the BitsyX system from ADS Inc. [17]. 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 0.8V 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 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). 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. Therefore, the effect of f^{INT} and f^{EXT} on the total program execution time should also be considered.

Table 1: Frequency combinations in BitsyX system

No	CPU (MHz)	Internal bus (MHz)	External bus (MHz)
F_1	100	50	100
F_2	200	50	100
F_3	300	50	100
F_4	200	100	100
F_5	300	100	100
F_6	400	100	100
F_7	400	200	100
F_8	133	66	133
F_9	265	133	133

4.2 Execution time model for BitsyX system

To derive a suitable execution time model of our target system, three different applications were run over all frequency sets, F_1 to F_9, and the total execution time for each case was measured. Figure 1 provides the execution time for each frequency setting normalized to the execution time with the maximum performance setting, i.e., setting F_7. From this Figure, we can easily see that "djpeg" is more CPU-intensive (i.e., $T^{ON} >> T^{OFF}$) than the "gzip" and "qsort" applications since lowering the CPU frequency for "djpeg" introduces significant execution time increase compared to "gzip" and "qsort". 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 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.

Clearly, T^{OFF} is strongly dependent on the external clock frequency. However, an important observation from the data reported in Figure

1 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 lack of exact timing information about these two operations which 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:

$$T^{OFF} = \frac{\alpha \cdot W^{OFF}}{f^{INT}} + \frac{(1-\alpha)\cdot W^{OFF}}{f^{EXT}} \qquad (8)$$

where α is the ratio between the data transfer time and the data fetch time and f^{INT} is internal bus clock frequency.

Based on the experimental results on various application programs, an α value of ~0.35 was obtained for all applications. For this value, the error in predicting the execution time was less than 3% for all nine frequency settings with tested applications.

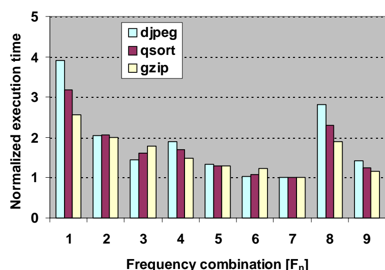


Figure 1. Execution time variation over different frequency combinations.

4.3 Events monitored through PXA255's PMU

Static calculation of W^{ON} and W^{OFF} of a program e.g., during the compilation time, is very difficult because on/off-chip latencies are greatly affected by dynamic behavior such as cache statistics and different access overheads for various external devices. So, these dynamic behaviors should be captured at run time. This task can be accomplished by using PXA255's PMU. The PMU supports monitoring of 15 performance 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 [15] and can thus be ignored. However, there is a limitation in using these events in the sense that only two events can be monitored at the same time along with the number of clock counts in a quantum (CCNT).

For our DVFS technique with workload decomposition, we need CPI^{avg}_{on} to separate workload. We performed many experiments to figure out which events can give useful information about the workload decomposition. Based on our experimental results, the following three events proved to be most helpful: (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). INSTR is required to get the CPI value, which indirectly represents the amount of off-chip workload. STALL captures the number of clock cycles when the CPU is stalled due to data dependency either because of on-chip stalls from internal register dependencies or off-chip stalls from external memory access. Note that DMISS is not exactly equivalent to the off-chip access count, because of the "miss-under-miss" capability using the "fill buffer" and "pending buffer" in PXA255-microarchitecture [3]. When a D-cache miss requests data in the same cache line as a previous D-cache miss event, then an external memory access does not occur for the current D-cache miss. In spite of this complication, the D-cache miss event can be used as an approximate metric to determine whether a task is CPU or memory-bound.

4.4 Calculating the average on-chip CPI

Using INSTR and STALL event statistics along with CCNT, CPI^{avg}_{on} can be extracted. At the start of each quantum, the PMU reports the CCNT, INSTR, and STALL. From these parameter values, we can calculate the average CPU cycles per instruction (CPI^{avg}) for the instruction stream as the ratio of CCNT to INSTR. Similarly, we can calculate the *average number of stall cycles per instruction* (SPI^{avg}). Here, SPI^{avg} accounts for both on-chip (SPI^{avg}_{on}) and off-chip access (SPI^{avg}_{off}) stalls. In Figure 2, we plot CPI^{avg} on the y-axis and SPI^{avg} on the x-axis for (a) "gzip" and (b) "djpeg" applications under different frequency settings per Table 1. Each dot in the plot represents one PMU report. From this figure, we can easily see that CPI^{avg} is linearly related to SPI^{avg}:

$$CPI^{avg} = k \cdot SPI^{avg} + c \qquad (9)$$

k is the slope (~1). The intercept c is equal to the average on-chip CPI without any stall cycles, CPI^{min}_{on}. Furthermore, CPI^{min}_{on} is equal to $CPI_0+CPI^{avg}_{branch}$ in eq. (3). Finally note that $SPI^{avg}=CPI^{avg}_{stall_onchip}+CPI^{avg}_{stall_offchip}$.

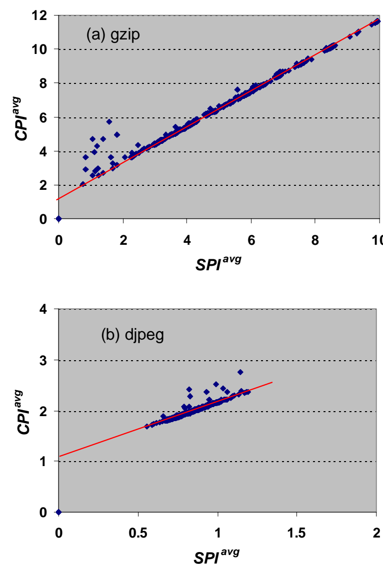


Figure 2. Contour plots of CPI^{avg} versus SPI^{avg} for different clock frequencies combinations.

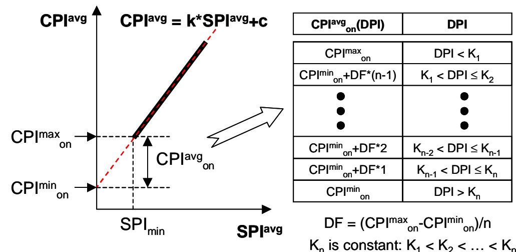


Figure 3. SPI^{avg}_{on} extraction using *DPI*.

To obtain CPI^{avg}_{on}, which is obviously equal to $CPI^{min}_{on}+SPI^{avg}_{on}$, it is required to extract SPI^{avg}_{on} from SPI^{avg}. Figure 3 shows a method of obtaining SPI^{avg}_{on} from the D-cache miss statistics. The range in which CPI^{avg}_{on} can exist is CPI^{min}_{on} to CPI^{max}_{on} which is the CPI value at the minimum SPI^{avg} (SPI_{min}) reported. Based on the experimental results in Figure 2, it is found that CPI^{avg}_{on} tends to be closer to CPI^{max}_{on} in case of CPU-intensive ("djpeg") and closer to

CPI^{min}_{on} in case of memory-bound program ("gzip"). Let *DPI* denotes *D-cache miss count* per instruction, defined as DMISS/INSTR. When there are many D-cache miss events, there is a higher probability of off-chip accesses (although a D-cache miss does not always result in an off-chip access as explained previously.) So, 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 $CPI^{avg}_{on} = CPI^{avg}_{min} + CPI^{avg}_{k}(DPI_k)$, where $CPI^{avg}_{k}(DPI_k)$ is the CPI^{avg} value for the corresponding DPI_k value.

4.5 Determining the optimal frequency setting

After obtaining the average on-chip *CPI* value for the current quantum *i*, $CPI^{avg}_{on,i}$, we calculate the on-chip and off-chip execution times for this quantum, T^{ON}_i and T^{OFF}_i, as follows:

$$T_i^{ON} = \frac{N_i \cdot CPI^{avg}_{on,i}}{f_i^{CPU}}, \quad T_i^{OFF} = T_i - T_i^{ON} \tag{10}$$

where N_i is the number of executed instructions in this quantum, and T_i and f^{CPU}_i are the execution time and the CPU frequency during 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 def. 1 and eq. (8). It is assumed that W^{ON}_{i+1} and W^{OFF}_{i+1} are equal to W^{ON}_i and W^{OFF}_i, respectively. Next, a frequency setting for the quantum i+1, F^{opt}_{i+1}, which satisfies the following equation is chosen as the optimal frequency setting:

$$T^{i+1}_{F^{i+1}_{opt}} \le (1 + PF_{loss}) \cdot T^{i}_{F_{max}} \tag{11}$$

where $T^{i+1}_{F^{i+1}_{opt}}$ is the expected execution time of quantum *i+1* at F^{opt}_{i+1} and $T^{i}_{F_{max}}$ is the execution time of quantum *i* at F_{max}. If there are more than one frequency settings that satisfy the above condition, then the setting that gives the expected execution time, which is closest to the target execution time, will be chosen.

5. IMPLEMENTATION

We implemented the proposed policy on the BitsyX platform, which runs Linux (v2.4.17). In particular, 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. Figure 4 shows the software architecture for our DVFS implementation.

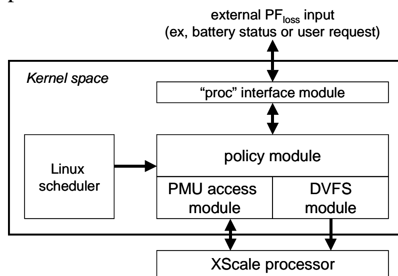


Figure 4. Software architecture of our DVFS implementation.

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. During the context switch, the PMU values for the previous process are read and the ideal frequency calculation for the next quantum is performed as described in section 4. A regression equation is maintained for each process, which consists of no more than five long-type variables, resulting in little space overhead for implementing our DVFS policy.

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, and 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 [22].

6. EXPERIMENTAL RESULTS

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 30% at steps of 10%. As seen in this figure, we obtained actual performance loss values very close to the target values for all programs (i.e., actual average within 2.5% of the target.) Figure 7 depicts the power consumption waveform of the BitsyX system when running "djpeg" for two cases: (a) without DVFS and (b) with DVFS In case (a), the program is run with the maximum performance frequency setting, i.e., F_7, and a 20% target PF_{loss}. As mentioned previously, power consumptions of the CPU core and the main memory could not be separately measured. We went about calculating the power consumption of the CPU core and main memory in BitsyX as $P_{active} – P_{idle}$, where P_{active} and P_{idle} denote the total system power consumptions when the BitsyX is active (performing some task) and when it is idle. Based on this experimental setup, for this benchmark, we were able to achieve a 25% energy saving in the CPU and main memory at the cost of a 22% increase in the total execution time. The measured energy savings (in CPU and main memory) for all benchmarks are shown in Figure 8. From these measurements, we conclude that our proposed DVFS technique results in energy savings of 20-40% for CPU-bound applications ("crc", "djpeg", and "math"), and 10-20% energy savings for memory-bound applications ("qsort" and "gzip") under 10-30% performance loss bounds. The lower energy saving results for memory-bound applications should be understood in light of the fact that in these applications most of the energy is consumed in accessing the main memory, and of course, the memory energy consumption is fixed (since operating voltage of memory is fixed although memory clock frequency varies.) It would be interesting to report the actual energy saving values for the CPU only as in [16]. Unfortunately, we cannot do this because of the current limitation of the BitsyX platform (no separate power planes for the CPU and main memory are provided.) So we go about reporting this data in an approximate manner as explained below.

When we consider energy savings of the proposed DVFS approach for the PXA255 processor only, higher energy savings can be obtained for memory-bound applications. For example, in case of "gzip", the total execution time without DVFS is 3.463sec and 1.6016sec is spent for data fetch from memory. Power consumption of memory chips can be calculated using the specified value (130mA in active mode) in the memory manufacturer's data sheet [21]. Since two memory chips, each of size 32MB, are used in the BitsyX platform, power consumption of the main memory in BitsyX board is 3.3V * (130mA + 130mA) = 858mW. CPU power is calculated as 400mW/(0.85*0.85) = 553.6mW by considering that two DC-DC converters, one for 12V to 3.3V conversion and the other for 3.3V to CPU operating voltage conversion with a conversion efficiency of 0.85 are used in the variable voltage generator. From these approximate calculations, we conclude that the ratio of the energy consumptions in the CPU and the main memory is about 0.75. Now, with this ratio, we can go ahead and estimate the CPU energy saving from the actual power consumption

data for the CPU plus main memory. Doing this calculation, we obtain a CPU energy saving of 80% for the memory-bound applications and 20% for the CPU-bound applications under a 20% performance loss bound.

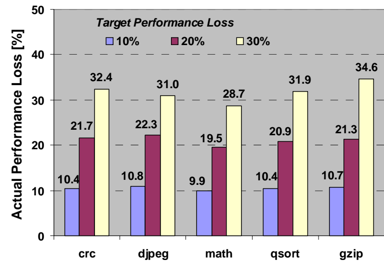
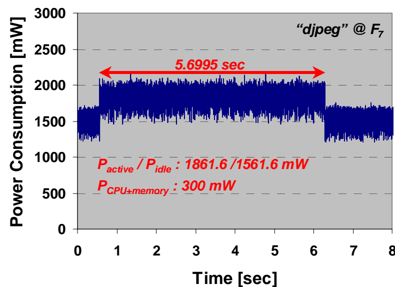
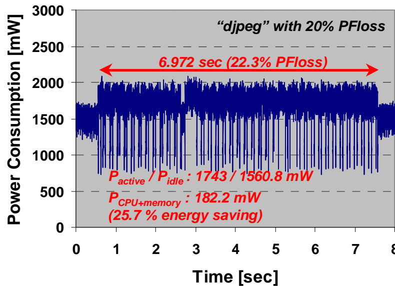


Figure 6. Actual performance loss values as a function of the corresponding target values.



(a) without DVFS - at maximum frequency setting



(b) with DVFS under a 20% performance loss constraint

Figure 7. CPU power consumption of with/without DVFS.

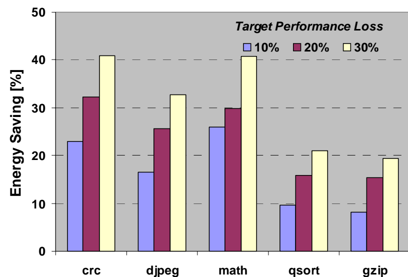


Figure 8. Energy saving (CPU+memory) for various applications.

7. CONCLUSION

In this paper, a regression-based DVFS policy for finely-tunable energy-performance trade-off was proposed and implemented on an PXA255-based platform. In the proposed DVFS approach, a program execution time is decomposed into two parts: on-chip computation and off-chip access latencies. The CPU voltage/frequency is scaled based on the ratio of the on-chip and off-chip latencies for each process under a given performance degradation factor. 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 energy saving of 20-40% with 10-30% performance loss for CPU-bound applications, whereas 10-20% saving was achieved for memory-bound applications. For both CPU and memory-bound programs, target performance degradation was finely controlled.

8. REFERENCES

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

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

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

[4] F. Yao, A. Demers, and S. Shenker, " A scheduling model for reduced CPU energy," IEEE Annual Foundations of Computer Science, 1995, pp.374-382

[5] 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

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

[7] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," Proc. of The International Symposium on Low Power Electronics and Design, pp.197-202, Monterey, Aug. 1998

[8] 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, March 2002.

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

[10] 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, 2000, pp. 381-386.

[11] 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, August 2002.

[12] 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, February 2002.

[13] D. Marculescu, "On the use of microarchitecture-driven dynamic voltage scaling," Workshop on Complexity-Effective Design, 2000.

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

[15] A. Weissel and F. Bellosa, "Process Cruise Control," CASES 2002, October 2002, Grenoble, France.

[16] 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

[17] http://www.applieddata.net/products_bitsyX.asp

[18] J. Hennessy and D. Patterson, "Computer Architecture – A Quantitative Approach," Morgan Kaufmann Publishers, Inc. 1996

[19] Developer's manual: "Intel XScale Microarchitecture for the PXA255 Processor" http://www.intel.com/design/pca/applications processors /manuals/278693.htm

[20] http://www.eecs.umich.edu/mibench

[21] http://download.micron.com/pdf/datasheets/dram/sdram/256MSDRAM_G.pdf

[22] http://www.instrument.com/pci/udas.asp