Analysis and Implementation of DVFS Techniques in a Multitasking Embedded System

CSE237a Final Project Report
March 13, 2008
Jamie Bradley and Diana Fang

1. Introduction

A key concern in embedded systems today is power consumption. Increasing demands for longer battery life and heating issues associated with power have led researchers to develop new ways to reduce power and energy consumption in embedded systems. Dynamic voltage and frequency scaling (DVFS) techniques are used to dynamically alter the voltage and frequency levels of the CPU to the lowest possible level while still maintaining high performance.

The goal of this project was to research and analyze different DVFS policies. We designed and implemented one of these techniques and compared it to our own naive user-level DVFS technique implemented in a previous project.

This paper is organized as follows. Section 2 provides an overview of previous DVFS research. Section 3 describes the DVFS design implemented in this project along and the challenges encountered. Section 4 presents the testing and results. Section 5 concludes with lessons learned and areas for future work.

2. Previous Research

2.1 Background

Current research has yielded several different DVFS techniques. The first group of techniques uses known task arrival times, workload, and deadlines to implement algorithms at the task level in the operating system. The second group of techniques uses compiler or application support for performing DVFS. For example, at compile time, the compiler can identify which sections of code will perform many memory accesses and which will not. The third group of techniques, however, achieves DVFS without any compiler support or any information regarding task characteristics. Rather, these techniques use runtime statistics to identify the workload of a task and thus estimate and predict the optimal v/f setting. These techniques can be further classified as fine-grained or course-grained. Course-grained techniques determine the v/f setting on a task-by-task basis. Fine-grained techniques adjust the v/f setting within a task boundary and usually perform better than course-grained techniques. We specifically researched four different DVFS techniques described below that only use run-time statistics for setting CPU voltage and frequency.

Experiments done in [1] show that it is advantageous to reduce the CPU frequency for a memory-intensive task, but not for a CPU-intensive task. The performance of a task with high CPU utilization is linearly dependent on frequency, and thus will suffer significant throughput loss when the frequency is lowered. A memory intensive task, however, will suffer minimal performance loss when the frequency is reduced. If a task is constantly accessing memory, then the CPU is constantly stalling and waiting for memory. Power consumption can be reduced by lowering the frequency for a memory intensive task, and system performance can be increased by running a CPU-intensive task at the highest frequency.

2.2 Proposed Techniques

In [2], Weissel and Bellosa propose a course-grained DVFS technique for multitasking systems called Process Cruise Control. Their technique uses event counters, which are counters that monitor countable tasks such as instructions, memory requests, and clock cycles. The specifics of what actually defines an event counter are not mentioned in the paper. Their technique calculated instructions per clock cycle and memory requests per clock cycle from the data collected from the event counters. A matrix of frequency domains is built from event count monitoring and is used to choose a frequency domain for an executing task. This matrix is divided into frequency domains by taking advantage of the trend stated above - CPU and cache intensive tasks will not benefit from a lower frequency. This technique was implemented in Linux 2.4.18

running on the Intel 80200 platform.

In [4], Choi, Soma, and Pedram present a fine-grained DVFS technique using Workload Decomposition, which classifies CPU workload as either on-chip or off-chip. An on-chip workload represents CPU clock cycles required to perform the set of instructions executed solely inside the CPU. An off-chip workload represents the number of clock cycles needed to perform external memory transactions. This technique uses a special hardware unit called the Performance Monitoring Unit (PMU) to monitor the (1) data cache miss count, (2) CPU stall cycle count, (3) number of clock counts in a quantum, and (4) the number of executed instructions. With this information, they are able to decompose the workload of a task into on-chip and off-chip execution times, and then from there, choose the optimal voltage/frequency setting.

In [5], Herbert and Marculescu propose a fine-grained DVFS technique for chip-multiprocessors (CMP) using Voltage Frequency Islands (VFIs). A CMP is a processor where multiple cores are replicated on a single die. The CMP is divided so that different parts have different voltages and frequencies, which they call voltage frequency islands. They describe three possible DVFS algorithms that monitor and alter the VFIs. For example, one of the algorithms alters the v/f based on the VFI's utilization within a threshold. They used simulations to compare these techniques and showed that DVFS techniques can improve energy-efficiency of multi-tasking CMPs.

In [1], Ghiman and Rosing propose a course-grained DVFS technique for a general purpose, multitasking system. Their technique uses runtime statistics and an online learning algorithm to select a voltage/frequency setting for an executing task. Because of this characterization, their algorithm can predict the optimal v/f setting from runtime statistics, specifically (1) instructions executed, (2) data cache misses, (3) instruction cache stalls, (4) data dependency stalls, and (5) clock cycles. The technique was implemented in Linux 2.6.9 running on the Intel PXA27x platform, using the PMU to collect the runtime statistics.

3. Design

Upon researching and comparing various algorithms, we chose to implement the Online Learning Algorithm designed in [1]. Not only was this algorithm clearly described in the paper, but it was also previously implemented on the Intel PXA27x Platform using Linux 2.6.9. Our initial intention was to also implement Process Cruise Control from [2], for the purpose of comparing one course-grained technique with another, but due to time limitations and unexpected challenges, were unable to do so. The design of our project is taken directly from the algorithm and implementation presented in [1], although the specifics of the implementation and programming are our own work.

3.1 Implementation

The online learning algorithm was implemented using a Linux Loadable Kernel Module (LKM) [6]. An LKM is a program that can be dynamically loaded and unloaded into the Linux kernel using simple shell commands. LKMs extend operating system functionality without having to recompile the entire kernel. However, for our project, we had to modify the kernel slightly to support the algorithm, but the main work was all done in the LKM.

The main idea of the algorithm is to create a weight vector for each task, where each weight corresponds to a different voltage/frequency setting. Using runtime statistics, a machine learning algorithm can update the weight vectors, and then select the voltage/frequency setting corresponding to the highest weight for the current task.

Our design uses function pointers that communicate with the DVFS LKM to call the appropriate DVFS LKM function that supports the algorithm. Figure 1 depicts our system level implementation. We use a PMU to gather runtime statistics of cache misses, instructions executed, clock cycles, and stalls. When a new task is created, the weight vector needs to be initialized in fork.c. During each scheduler tick, the DVFS LKM is notified by sched.c, and the current task's weight vector is updated according to the statistics collected from the PMU. The PMU is then restarted. At each context switch, the DVFS LKM is also notified by sched.c, and an expert (specific v/f setting) is selected based on the highest weight value in the vector of the current task. The v/f setting is then applied using a power manager interface. For the detailed weight update calculations of the algorithm, see [1].

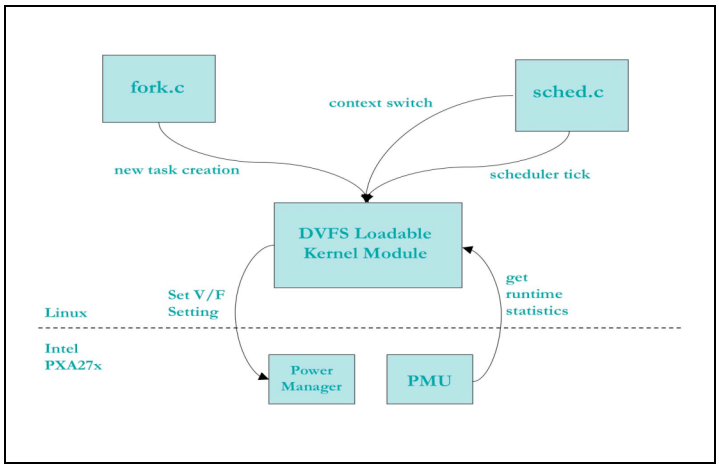

Figure 1. System Level Implementation Model

3.2 Challenges

Throughout this project, we encountered several unexpected challenges, which included using the PMU and the Power Manager Interface, implementing an LKM, doing fixed point conversion, and managing energy consumption.

Initially, we faced difficulties while working with the Performance Monitoring Unit [3]. The PMU consists of four counters that can be programmed to measure different system characteristics; the PXA27x manual [3], however, does not specify how to actually access and use these counters. Fortunately, one of the authors of [1] pointed us to an interface for the PMU already in the operating system that provides simple function calls to interact with the PMU.

Another significant challenge was using the Power Manager Interface to change the frequency and voltage. The kernel has many critical sections, two of which are found in context switches and scheduler ticks. During these critical sections, the initial power manager interface functions we used caused errors due to scheduling while atomic and scheduling during an idle thread. To fix this problem, we created our own function in the interface that mimicked an existing function with minor changes.

Additionally, we struggled with implementing and using an LKM [6,8]. Because we were unfamiliar with LKMs, we first implemented and tested a trivial "Hello World" module. Modules have specific requirements that must be met to work correctly, such as specifying a license in order to reference exported symbols from the kernel. After succeeding with this module, we began the implementation of the DVFS LKM and had few problems thereafter with using the LKM.

Because embedded systems typically do not support floating point arithmetic, we used fixed point conversion to scale the fractions used in the algorithm. This conversion is fairly simple when adding or subtracting, but in dividing and multiplying, it becomes more complicated. For the algorithm, since it uses probabilistic methods, all the numbers used were between the range of 0 and 1. We decided to scale our numbers from 0 to 1000. One of the issues occurs with boundary cases where our number became less than 0 and 1000. We had to figure out what these cases meant and how to handle them.

One of the biggest challenges and an area for improvement is measurement of energy consumption. Due to time constraints, we used a previously written program to collect the statistics for our experiments, rather than writing a new one tailored to kernel level DVFS. The previously written program only tracks energy consumption per second as it was done at the user level, while the LKM being a kernel module was able to change the V/F setting nearly every millisecond. This discrepancy caused unpredictable and imprecise results. For more accurate measurements a measurement of the voltage across a resistor could be used to calculate power and energy.

4. Results

To test the effectiveness of our implementation, we ran the 'mplayer' software on the Xscale platforms using various media files. We ran the 'mplayer' software with both our naive user-level DVFS implementation and with our LKM. In contrast to the online learning algorithm, our naïve implementation sets the v/f setting according to CPU utilization alone. Results are shown in Figure 2a and 2b. The DVFS LKM implemented in this paper reduces energy consumption. Unfortunately, we have found the DVFS LKM to have unpredictable behavior and therefore, cannot say with confidence that the algorithm works accurately. Also, the measurements that we collected are not precise enough to determine accuracy. In theory, DVFS on a per task basis should accurately change the frequency according to the current task.

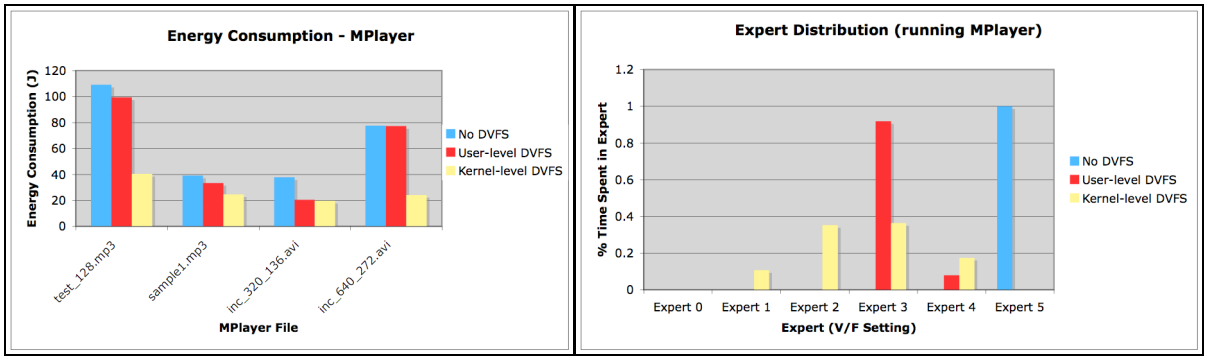

Figure 2a. Energy Consumption

Figure 2b. Expert Distribution

5. Future Work and Conclusions

There are currently many different dynamic voltage and frequency scaling methods proposed and being researched. Because of the need for energy minimization in embedded systems, DVFS will continue to be researched and improved upon for years to come. Some ideas for future work include perfecting DVFS techniques for multi-tasking systems and combining DVFS techniques with Dynamic Power Management.

We have shown that low level DVFS methods such as implementing algorithms in the kernel as opposed to the user-level is more effective. However, we have learned that kernel-level programming is significantly more difficult and dangerous than user-level programming. One bad pointer can cause the entire system to stall, and, when this occurs, rebooting is the only solution. Kernel modifications also need to be lightweight to accommodate the limited amount of memory on embedded systems.

As defined in [7], experiential learning is the "knowledge, skills, and/or abilities attained through observation, simulation, and/or participation that provides depth and meaning to learning by engaging the mind and/or body through activity, reflection, and application." Experiential learning best describes the lessons we learned from this project. No classroom could have ever taught us the skills and knowledge that we learned from actually working with the Linux kernel and facing the many challenges researchers and developers face.

6. References

[1] Dhiman, G. and Rosing, T. S. 2007. "Dynamic voltage frequency scaling for multi-tasking systems using online learning." In Proceedings of the 2007 International Symposium on Low Power Electronics and Design (Portland, OR, USA, August 27-29, 2007). ISLPED '07. ACM, New York, NY, 207-212

[2] Weissel, A. and Bellosa, F. 2002. "Process cruise control: event-driven clock scaling for dynamic power management." In Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis For Embedded Systems(Grenoble, France, October 08 - 11, 2002). CASES '02. ACM, New York, NY, 238-246.

[3] "Intel XScale Core Developer's Manual," http://download.intel.com/design/intelxscale/27347302.pdf

[4] Choi, K., Soma, R., and Pedram, M. 2004. "Dynamic voltage and frequency scaling based on workload decomposition." In Proceedings of the 2004 International Symposium on Low Power Electronics and

Design (Newport Beach, California, USA, August 09 - 11, 2004). ISLPED '04. ACM, New York, NY, 174-179.
[5] Herbert, S. and Marculescu, D. 2007. "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors." In Proceedings of the 2007 International Symposium on Low Power Electronics and Design (Portland, OR, USA, August 27 - 29, 2007). ISLPED '07. ACM, New York, NY, 38-43.
[6] "The Linux Kernel Module Programing Guide," http://www.tldp.org/LDP/lkmpg/2.6/html/index.html
[7] "Experiential Learning," http://people.uleth.ca/~steve.craig/whatis.htm
[8] Henderson, Bryan. "Linux Loadable Module HOWTO," http://tldp.org/HOWTO/Module-HOWTO/
[9] Dhiman, Gaurav and Orkin, Daniel. personal communication, February and March 2008.