Comparison of DVFS Techniques for Multitasking Systems

Goal

The goal of this project is to research and evaluate several Dynamic Voltage and Frequency Scaling policies. We will implement and compare two previously researched DVFS techniques to accommodate multi-tasking environments. We will use the Intel PXA27x platform. If time allows, we will also seek to improve upon these policies.

Background

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 techniques are used to dynamically alter the voltage and frequency levels of the CPU to the lowest possible level to still yield high performance.

Current Policies

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. The third group of techniques, however, achieves DVFS without compiler support or information regarding task characteristics, using runtime statistics instead. We plan to investigate and compare techniques in this particular group.

In [1], Ghiman and Rosing propose a 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. Experiments done in [1] show that a task with high CPU utilization will not benefit from lowering the frequency, while a memory intensive task will benefit. Because of this characterization, their algorithm can predict the optimal V/F setting from runtime statistics, specifically the (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 a separate hardware unit to measure statistics, called a Performance Monitoring Unit.

In [2], Weissel and Bellosa also propose a DVFS technique called Process Cruise Control for multitasking systems. Their technique uses event counters, specifically instructions per clock cycle and memory requests per clock cycle, to choose a frequency domain for an executing task using a matrix. This matrix is divided into frequency domains by taking advantage of the trend stated in the previous paragraph - 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 DVFS technique using Workload Decomposition, which characterizes CPU workload as either on-chip or off-chip. Like [1], this technique uses the PMU to monitor the (1) data cache miss count, (2) CPU stall cycle count, (3) number of clock counts in a quantum, and (4) number of executed instructions.

In [5], Herbert and Marculescu propose a fine-grained DVFS technique for chip-multiprocessors using Voltage Frequency Islands (VFI). They describe three possible DVFS algorithms that monitor and alter the VFI. Their technique, however, is based on simulation and thus uses no PMU or event counter equivalent.

**Detailed Description**

After studying the above DVFS schemes, we have chosen to focus on [1] and [2] for our project. Both of these techniques were implemented in Linux on Xscale using the six available V/F levels; in addition, both rely solely on runtime statistics and are course-grained policies. In implementing these techniques, we intend to create a Loadable Kernel Module that receives notification from the process manager on (1) task creation, (2) context switching, and (3) each scheduler tick, to perform the DVFS policy. The task_struct will also need to be modified to include for [1] a weight vector and for [2] the event counters. Following implementation, we intend to test our policies using several benchmarks and compare the results. If time permits, we would like to modify one of the policies to improve energy savings.

**Challenges and Limitations**

One of the general challenges in developing the DVFS policies is in limiting the overhead. However, the two papers that we will be focusing on are reported to be lightweight and thus have negligible overhead. We hope that this will be the case in our implementation as well.

Some of the challenges that we have currently encountered include working with the performance monitoring unit (PMU) [3] of the XScale platform and working with loadable kernel modules (LKM) [6]. Because we are unfamiliar with the PMU and how to use it to retrieve the statistics that we need for our algorithms, we will need to do more research and receive guidance on its usage. In our attempt to create a trivial LKM for Xscale, we had compilation version problems; we are working to determine the source and solution of the errors.

Another challenge that we foresee is implementing the process cruise control on a different platform than the one described in [2]. Some modifications will be necessary to tailor the control algorithm to the XScale PXA277x platform.

**Execution Plan**

Work done so far:
- Researched different DVFS techniques
- Selected the techniques described in [1] and [2] to implement
- Looked into the Linux kernel source code
- Started the design of the two techniques
- Practiced writing a kernel module

Work still to be done:
- Implement the two DVFS techniques
- Run the two techniques on different benchmarks
- Collect energy statistics from running the two different techniques and compare

**Summary and Outlook**

Our project compares two different DVFS techniques described in [1] and [2]. In [1], an online learning algorithm is used to characterize tasks and selects the best frequency and voltage settings accordingly. In [2], event counters are used to select the frequency and voltage settings. Implementation of both techniques will be done on the Intel XScale PXA277x platform. Implementing both techniques on the same platform will provide good comparison statistics.

We recognize that there will be many challenges during implementation and foresee the possibility of not accomplishing all of our goals in the given amount of time. However, we are optimistic in at least implementing one of the techniques and measuring its performance.

**References**