ANALYSIS OF A DYNAMIC VOLTAGE AND FREQUENCY SCALING ALGORTIHM ON XSCALE.

CSC 714 Project.
Dr. Frank Mueller

Prasanth Ganesan, Shobhit Kanaujia, Ishdeep Sawhney
{pganesa, sokanauj, issawhne}@unity.ncsu.edu

TABLE OF CONTENTS

1. Design of DVS Scheduler

The WinCE scheduler uses a preemptive, priority based time slice algorithm to schedule threads irrespective of the process to which they belong. Priority inversion takes place when a low priority thread blocks a high priority thread for a resource and only in this case WinCE scheduler boosts the priority of the lower priority task. In all other cases the priority of the task is determined at creation time. However WinCE provides APIs that can be used to dynamically change the priority of a thread at the user level.

We intend to exploit this functionality and flexibility of the thread library to implement our own user level scheduler. This would schedule threads of the same process by manipulating the priority of the threads. The scheduler would also track the suspended, executing and exited threads. Data structures would maintain the current queue of threads.

We currently intend to implement the cycle conserving DVS algorithm with EDF scheduling. The specification of this algorithm has been obtained from [1].

To evaluate the performance of the DVS algorithm, each task will be an independent thread along with a scheduler thread. The scheduler thread will start the first task ("thread") by setting its time limit as its worst case execution time (WCET). After the completion of a task the scheduler will evaluate the processing requirement and set the voltage/ frequency accordingly for the next task. This cycle repeats till there are tasks to execute. The tasks the scheduler picks would be in a queue with the order determined by an EDF scheme.

A sample context switch between threads is indicated below:

Suppose thread 1 (Th1 – scheduler thread) is initially executing at THREAD_PRIORITY_TIME_CRITICAL and has to give control to thread 2 (Th2). Initially Th2 is in suspended state.
1. Th1 resumes Th2 by calling ResumeThread () function.
2. Th1 sets the time quantum for Th2 to its WCET.

3. Th1 makes Th2's priority as THREAD_PRIORITY_TIME_CRITICAL and relinquishes control.
4. After Th2 finishes, Th1 checks to see if Th2 terminated due to deadline miss (time quantum getting over) or has finished early.

Th1 is the scheduler thread in user level.

WinCE APIs to be used:
CreateThread ()
SetThreadPriority ()
ResumeThread ()
CeSetThreadQuantum ()

Assumption:
There is no resource sharing between the tasks, the tasks are independent.

2. Approaches to low-level hardware access

In order to do frequency scaling, we need to setup registers in the processor. Thus we need a way to write to these registers from the application-level. From our research on WinCE capabilities we have drawn upon the conclusion that there is a minimal protection for processes, thus writing to processor registers from the application-level is 'legal' and does not require any privileges. We draw our approaches from the UNIX world where we have most of our prior experiences.

Steps for frequency scaling:
The PXA250 and PXA210 applications processors define four power modes: run, turbo, idle, and sleep.
Run mode is the normal execution mode. Each of the other power modes must be entered from run mode. Each of the other power modes return to run mode when being exited. Turbo mode is designed for use in times requiring peak processing. Turbo mode frequency is a multiple of run mode. Both run and turbo mode frequencies are programmable. In order to change the frequency first, the Core Clock Configuration Register (CCCR) register must be programmed. The CCCR is a memory mapped register in the Clock Manager. The value in the CCCR specifies how the memory, run mode and turbo mode frequencies are calculated, based on the 3.6864MHz oscillator.

The memory frequency is calculated to be the oscillator frequency multiplied by the L field of the CCCR. The run mode frequency is determined as the product of the memory frequency and the value in the M field of the CCCR. The turbo mode frequency is the product of the run mode frequency and the value of the N field of the CCCR. Once the CCCR value has been written, to
have these settings take effect; the “Frequency Change Sequence (FCS)” must be entered.

The Frequency Change Sequence (FCS) is entered by writing a 1 to the Frequency Change bit in the CCLKCFG coprocessor register
If the FCS is performed after the system has been booted more fully, the following conditions must be satisfied before executing the FCS:
1. Configure the memory controller to ensure that SDRAM contents are preserved.
2. Set the SDRAM refresh timer in the memory controller to meet the maximum refresh time required by the slower of the 2 memory frequencies involved.
3. Disable the LCD controller.
4. Disable every enabled peripheral that is not able to tolerate a disruption of up to 500us in its DMA service.
5. Disable every enabled peripheral that is not able to tolerate a delay of up to 500us in its interrupt service.
6. Program CCCR to reflect the desired frequency.

We considered three different approaches to doing low-level hardware access:
Ioctl-approach:
KernerlIoControl () is an API provided by WinCE, which can be used to access hardware locations. Here we actually depend on the WinCE API to provide us an ioctl () call (i.e. an appropriate ioctl-code) for accessing clock/power management registers. Preliminary research on this has returned a negative. There is not great detail in documentation; the ioctl-codes that we could find did not seem to be ones for accessing the power/clock management registers. This would have been the cleanest way out, but we are against a wall with this approach.

Mmap- approach:
This is really the next cleanest approach to do hardware access. Most of the processor registers exist at well known addresses. We can obtain a virtual

address for those well-known addresses by using WinCE APIs. The APIs available for this approach are:
VirtualAlloc (), VirtualCopy (), MapIoSpace (), TransBusAddrToVirtual ()

Inline assembly:
CCLKCFG coprocessor register is not mapped to a well-known hardware address, so in order to access that register we'd most-probably have to do inline assembly.

We anticipate using Mmap-approach where possible and use Inline-Assembly for the CCLKCFG coprocessor register.

3. Schedule

Week 1 (Oct 31 – Nov 6)

- Investigation into the development environment for Pocket PC 2002.
  Status: Completed Handled By: Ishdeep, Shobhit and Prasanth
- Investigation into functionality and flexibility of the thread library provided by WinCE
  Status: Completed Handled By: Ishdeep
- Investigate hardware/register/api access to implement voltage and frequency scaling.
  Status: Completed Handled By: Shobhit
- Investigate remaining DVS algorithms, choose algorithm.
  Status: Completed Handled By: Prasanth
- Evaluate the different design approaches and finalize design
  Status: Completed Handled By: Prasanth, Shobhit and Ishdeep

Week 2 (Nov 7 – Nov13)

- Implement the low level frequency scaling/voltage scaling functions and create APIs to be used by the scheduler.
- Implement a prototype thread library without DVS scaling.
- Create data structures and implement mechanism to add threads in an EDF manner.

Week 3(Nov 14 – Nov 20)

- Integrate the above functionalities to have a complete simulator.
- Run dummy tasks to evaluate the correct functioning of the simulator

Week 4(Nov 21 – Nov 27)

- Evaluate the operation of the DVS algorithm and determine it's power saving benefits.
- Perform various performance measurements.

Week 5(Nov 27 – Dec 4)

- Back Up week.

References

[I]*Padmanabhan Pillai, Kang G. Shin*, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems (2001)", 18th ACM Symposium on Operating Systems Principles.