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

Kihwan Choi

Ramakrishna Soma

Massoud Pedram

University of Southern California

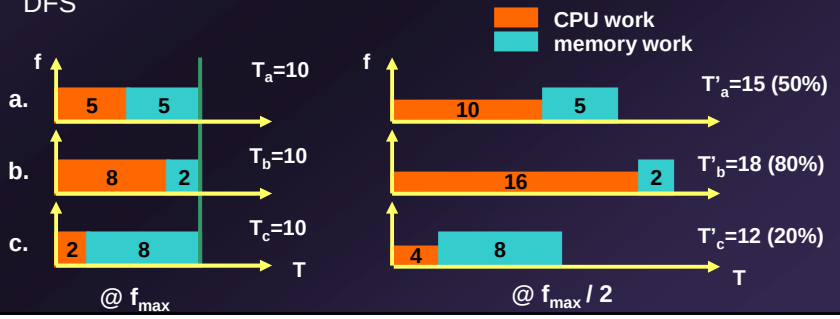
Dept. of EE

Outline

- Introduction
- XScale's Performance Monitoring Unit (PMU)
- Proposed Fine-grained DVFS Policy
- Experimental Results
- Conclusion

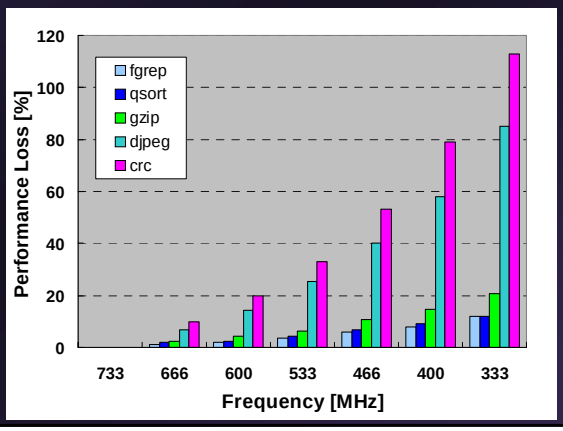
Energy and Performance Trade-off

- Dynamic Voltage and Frequency Scaling (DVFS)
  - Provide just enough power to meet the performance requirements
- The execution trace of an application program consists of CPU and memory instructions
- On a memory miss, the CPU has to stall until the external memory access is completed
  - If DVFS is applied during the CPU stall times, then the CPU energy is saved with little performance loss
  - Memory-bound applications exhibit lower performance penalty with DFS



Motivation

- Performance degradation of the target system (i.e., the Apollo Testbed II) for different applications at various frequencies
- For a given performance loss target (say 20%), higher CPU energy saving is possible for memory-intensive applications because the CPU frequency can be scaled more aggressively



The Program Execution Time

- The amount of CPU and memory workload for an application program must be determined
- Execution time of a program is the sum of the On-chip (CPU work) and the Off-chip Latency (memory work)
  - $T = T_{onchip} + T_{offchip}$
- T_{onchip} : varies with the CPU frequency
  - Stalls due to data dependency
  - Cache hit rate
  - TLB hit rate, …
- $T_{offchip}$: is does not vary with the CPU frequency
  - Access latency to external memory such as the SDRAM or the frame buffer memory through the PCI is a function of the external bus frequency only

Calculating the Program Execution Time

- $T = T_{onchip} + T_{offchip}$

$$T_{onchip} = \frac{\sum_{i=1}^{n} CPI^{i}_{onchip}}{f_{cpu}} \qquad T_{offchip} = \frac{\sum_{j=1}^{m} CPI^{j}_{offchip}}{f_{mem}}$$

n : number of onchip instructions
CPI^{i}_{onchip} : CPU clocks per instruction
f_{cpu} : CPU clock frequency (variable)

m : number of offchip events
$CPI^{i}_{offchip}$: memory clocks per offchip event
f_{mem} : memory clock frequency (fixed)

- When all parameters are known and the target performance loss factor (PF_{loss}) is specified, then CPU frequency may be calculated as:

$$f_{target} = \frac{n.CPI_{onchip}}{\frac{(1+PF_{loss}) \cdot n.CPI_{onchip}}{f_{max}} + \frac{PF_{loss} \cdot m.CPI_{offchip}}{f_{mem}}}$$

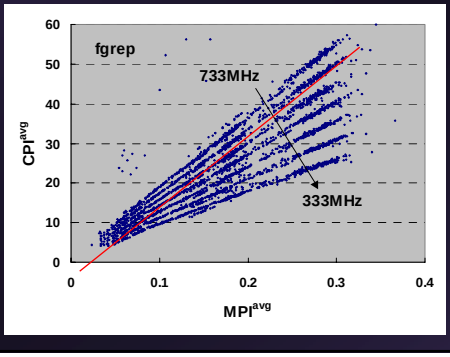
$PF_{loss} = 0 \Rightarrow f_{target} = f_{max}$
$PF_{loss} \uparrow \Rightarrow f_{target} \downarrow$
$PF_{loss} \downarrow \Rightarrow f_{target} \uparrow$
$T_{offchip} \uparrow \Rightarrow f_{target} \downarrow$

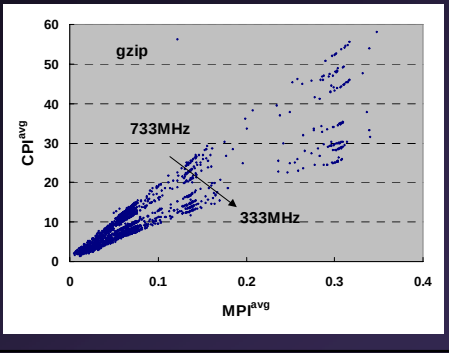
Performance Monitoring Unit (PMU)

- PMU on the XScale processor chip can report up to 20 different dynamic events during execution of a program
  - Cache hit/miss counts
  - TLB hit/miss counts
  - No. of external memory accesses
  - Total no. of instructions being executed
  - Branch misprediction counts
- However, only two events can be monitored and reported at any given time
- For DVFS, we use PMU to generate statistics for
  - Total no. of instructions being executed (INSTR)
  - No. of external memory accesses (MEM)
- We also record the no. of clock cycles from the beginning of the program execution (CCNT)

Plot of CPI vs. MPI

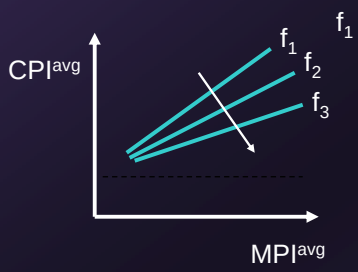
- PMU is read at every OS quantum (~50msec)
- We define MPI as the ratio of memory access count to the total instruction count
  - CPI^{avg} = CCNT / INSTR, during a quantum
  - MPI^{avg} = MEM / INSTR, during a quantum
- Plots of CPI^{avg} vs. MPI^{avg} for two different applications and various clock frequencies





Regression Equation Modeling

- A linear regression equation can be generated for each CPU clock frequency



$$CPI^{avg} = b(f)*MPI^{avg} + c$$

$$b = \frac{N \cdot (\sum_{i=t}^{t-N+1} x_i \cdot y_i) - (\sum_{i=t}^{t-N+1} x_i) \cdot (\sum_{i=t}^{t-N+1} y_i)}{N \cdot (\sum_{i=t}^{t-N+1} x_i^2) - (\sum_{i=1}^{t-N+1} x_i)^2},$$

$$c = \frac{\sum_{i=t}^{t-N+1} y_i}{N} - b \cdot \frac{\sum_{i=t}^{t-N+1} x_i}{N}$$

N : No. of regression points, e.g., 25
x_i : MPI^{avg} for the i^{th} point
y_i : CPI^{avg} for the i^{th} point

How the PMU Data is Used in DVFS

- Target frequency for a given PF_{loss}

$$f_{target} \approx \frac{n.CPI_{onchip}}{(1+PF_{loss}) \cdot n.CPI_{onchip}} + \frac{PF_{loss} \; m.CPI_{offchip}}{f_{max} \qquad\qquad f_{mem}}$$

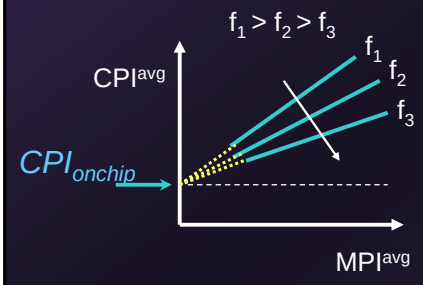
known
unknown

733 MHz — f_{max}
given — PF_{loss}
100 MHz or 33 MHz — f_{mem}

- The four unknown parameters (circled in red) must be calculated from CCNT and the two reported values by the PMU (INSTR & MEM)
  - *n* (no. of executed instructions) ← INSTR
  - *m* (no. of offchip events) ← MEM
  - CPI_{onchip} ← Average onchip CPI ?
  - $CPI_{offchip}$ ← Average offchip CPI ?

Calculating CPI_{onchip}

- Notice that CPI_{onchip} denotes the CPI value without the offchip accesses; So it is equal to the y intercept of the CPI vs. MPI plot



$$CPI_{onchip} = c$$

$$c = \frac{\sum_{i=t}^{t-N+1} y_i}{N} - b \cdot \frac{\sum_{i=t}^{t-N+1} x_i}{N}$$

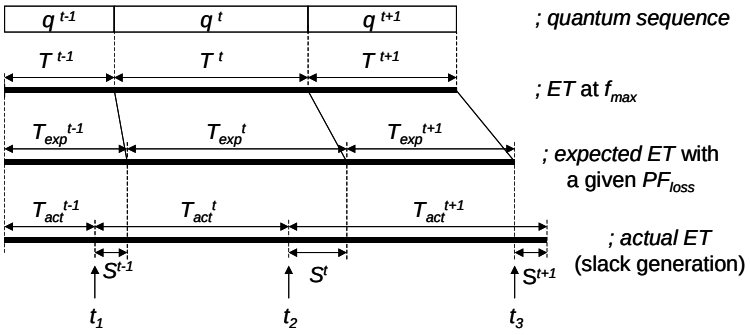
$$b = \frac{N \cdot (\sum_{i=t}^{t-N+1} x_i \cdot y_i) - (\sum_{i=t}^{t-N+1} x_i) \cdot (\sum_{i=t}^{t-N+1} y_i)}{N \cdot (\sum_{i=t}^{t-N+1} x_i^2) - (\sum_{i=1}^{t-N+1} x_i)^2},$$

Calculating $CPI_{offchip}$

- It is difficult to get $CPI_{offchip}$ directly from the PMU events
  - $CPI_{offchip}$ accounts for both the SDRAM access (100MHz) and the PCI device access (33MHz) in the Apollo Testbed II system
  - MEM captures both offchip events
- Recall that $CPI_{offchip}$ is only needed to calculate $T_{offchip}$
- We can calculate $T_{offchip}$ directly as shown below
  - $T = T_{onchip} + T_{offchip} = CCNT / f_{cpu}$
  - $T_{offchip} = CCNT/f_{cpu} - T_{onchip}$

Prediction Error Adjustment (I)

- Error adjustment



$S^{t-1} = T_{exp}^{t-1} - T_{act}^{t-1}$

$S^{t} = T_{exp}^{t} + T_{exp}^{t-1} - T_{act}^{t} - T_{act}^{t-1}$
$= T_{exp}^{t} - T_{act}^{t} + S^{t-1}$

$S^{t+1} = T_{exp}^{t+1} + T_{exp}^{t} + T_{exp}^{t-1} - T_{act}^{t+1} - T_{act}^{t} - T_{act}^{t-1}$
$= T_{exp}^{t+1} - T_{act}^{t+1} + S^{t}$

ET : Execution time

$T_{exp}^{k} = T^{k} \cdot (1+PF_{loss})$
$(k = t-1, t, t+1)$

Prediction Error Adjustment (II)

- Target frequency selection
  - without adjustment

$$f^{t+1} = \frac{f_{max}}{1+PF_{loss} \cdot \left[1+\beta^{t} \cdot \left(\frac{f_{max}}{f_{cpu}}\right)\right]}$$

$$\beta^{t} \approx \frac{T^{t}_{offchip}}{T^{t}_{onchip}}$$

  - with adjustment

$$f^{t+1} = \frac{f_{max}}{1+PF_{loss} \cdot \left[1+\beta^{t} \cdot \left(\frac{f_{max}}{f^{t}}\right)+\frac{S^{t}}{PF_{loss} \cdot T^{t}_{onchip}} \cdot \left(\frac{f_{max}}{f^{t}}\right)\right]}$$

Fine-grained DVFS Policy

- Scaling is performed at every OS quantum(~50msec)
- Optimal frequency for the next quantum is chosen based on the statistics of the previous quanta
- T_{onchip} and $T_{offchip}$ are calculated as :

$$T_{onchip} = \frac{\sum_{i=1}^{n} CPI_{onchip}^{i}}{f_{cpu}} = \frac{n \cdot CPI_{onchip}}{f_{cpu}}$$

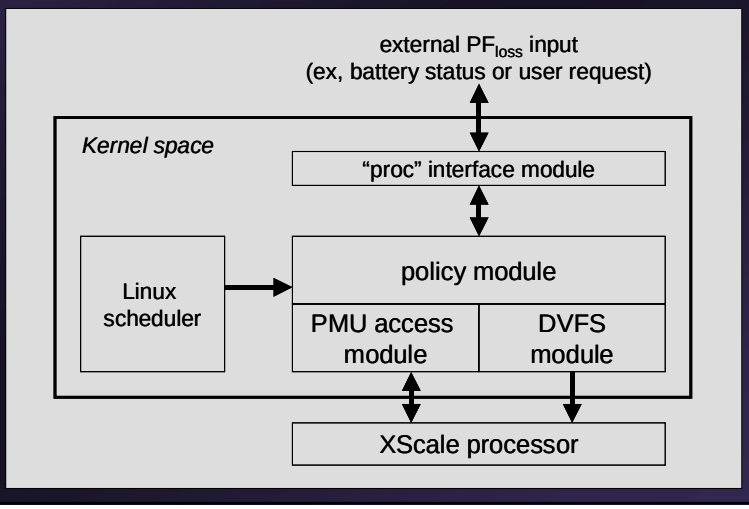
$$T_{offchip} = \frac{\sum_{j=1}^{m} CPI_{offchip}^{j}}{f_{mem}} = T - T_{onchip}$$

- Frequency for the next quantum (t+1), f^{t+1}, is calculated as:

$$f^{t+1} = \frac{f_{max}}{1 + PF_{loss} \cdot \left[1 + \beta^{t} \cdot \left(\frac{f_{max}}{f^{t}} \right) + \frac{S^{t}}{PF_{loss} \cdot T_{onchip}^{t}} \cdot \left(\frac{f_{max}}{f^{t}} \right) \right]}$$

Implementation (I)

- Offchip Latency-driven DVFS (OL-DVFS)
  - Software architecture



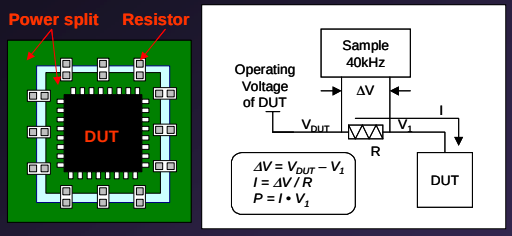
Implementation (II)

- A voltage is mapped to each CPU frequency
- Voltage control circuitry is on-board
- Power measurement with DAQ (Data Acquisition)

CPU Freq. vs. Volt. Relation

Frequency (MHz)	Voltage (V)
333	0.91
400	0.99
466	1.05
533	1.12
600	1.19
666	1.26
733	1.49

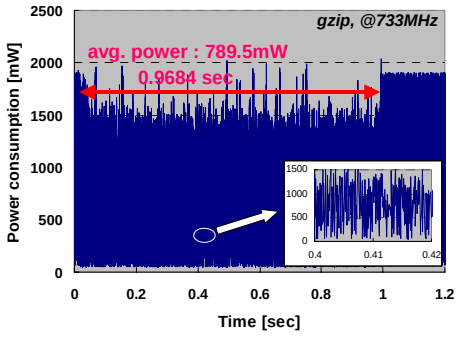
Data Acquisition system



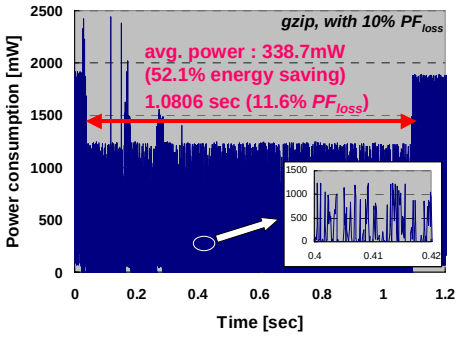
Experimental Results (I)

- Power consumption vs. performance degradation

without OL-DVFS

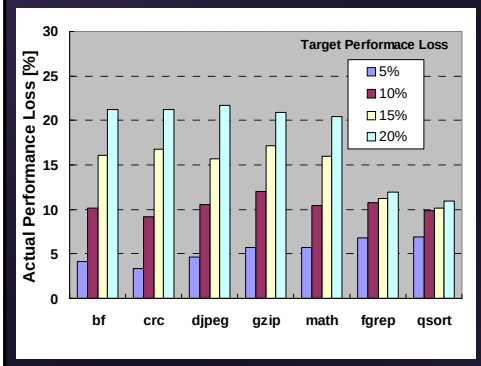


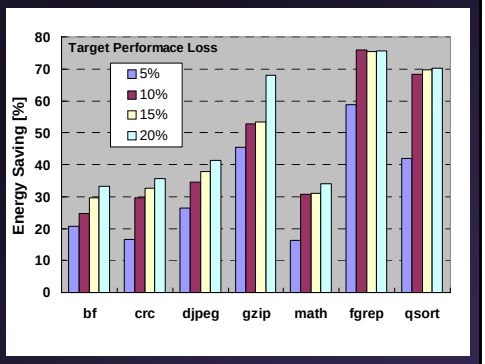
with OL-DVFS



Experimental Results (II)

- Comparison of the actual PF_{loss} with the target performance loss





Conclusions

- A fine-grained DVFS technique was proposed and implemented in XScale-based platform
- From actual measurements
  - For memory-bound programs, more than 70% CPU energy savings is achieved with 12% of performance degradation
  - For CPU-bound programs, 15~60% CPU energy savings is achieved at the cost of a 5~20% performance penalty
- Future work will focus on extending this technique to a PXA255-based embedded system