
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).
  - \( T = T_{\text{onchip}} + T_{\text{offchip}} \)
  - \( T_{\text{onchip}} \): varies with the CPU frequency
    - Stalls due to data dependency
    - Cache hit rate
    - TLB hit rate, ...
  - \( T_{\text{offchip}} \): 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

- \( T = T_{\text{onchip}} + T_{\text{offchip}} \)
  - \( T_{\text{onchip}} = \sum_{i=1}^{n} \frac{n \cdot CPI_{\text{onchip}}}{f_{\text{cpu}}} \)
  - \( T_{\text{offchip}} = \sum_{j=1}^{m} \frac{m \cdot CPI_{\text{offchip}}}{f_{\text{mem}}} \)

- When all parameters are known and the target performance loss factor \((PF_{\text{loss}})\) is specified, then CPU frequency may be calculated as:
  - \( f_{\text{argmax}} = \frac{n \cdot CPI_{\text{onchip}}}{(1 + PF_{\text{loss}}) \cdot \frac{n \cdot CPI_{\text{onchip}}}{f_{\text{max}}} + PF_{\text{loss}} \cdot \frac{m \cdot CPI_{\text{offchip}}}{f_{\text{mem}}}} \)

  \( PF_{\text{loss}} = 0 \Rightarrow f_{\text{argmax}} = f_{\text{max}} \)
  \( PF_{\text{loss}} \uparrow \Rightarrow f_{\text{argmax}} \downarrow \)
  \( PF_{\text{loss}} \downarrow \Rightarrow f_{\text{argmax}} \uparrow \)
  \( T_{\text{offchip}} \uparrow \Rightarrow f_{\text{argmax}} \downarrow \)
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

![Regression Equation Diagram]

\[ CP_{\text{avg}} = b(f) \cdot MP_{\text{avg}} + c \]

- \( N \) : No. of regression points, e.g., 25
- \( x_i \) : \( MPI_{\text{avg}} \) for the \( i \text{th} \) point
- \( y_i \) : \( CPI_{\text{avg}} \) for the \( i \text{th} \) point

How the PMU Data is Used in DVFS

- Target frequency for a given \( PF_{\text{loss}} \)

\[ f_{\text{target}} = \frac{n \cdot CPI_{\text{onchip}}}{(1 + PF_{\text{loss}}) \cdot CPI_{\text{onchip}}} + \frac{m \cdot CPI_{\text{offchip}}}{PF_{\text{loss}} \cdot CPI_{\text{offchip}}} \]

- 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) \( \rightarrow \) INSTR
  - \( m \) (no. of offchip events) \( \rightarrow \) MEM
  - \( CPI_{\text{onchip}} \) \( \rightarrow \) Average onchip CPI?
  - \( CPI_{\text{offchip}} \) \( \rightarrow \) Average offchip CPI?
Calculating CPI\textsubscript{onchip}

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

\[ CPI\textsubscript{onchip} = c \]

\[
c = \frac{\sum_{i=1}^{n-1} y_i}{N} - b \cdot \frac{\sum_{i=1}^{n-1} x_i}{N}
\]

\[
b = \frac{N \cdot (\sum_{i=1}^{n-1} x_i \cdot y_i) - (\sum_{i=1}^{n-1} x_i) \cdot (\sum_{i=1}^{n-1} y_i)}{N \cdot (\sum_{i=1}^{n-1} x_i^2) - (\sum_{i=1}^{n-1} x_i)^2}
\]

Calculating CPI\textsubscript{offchip}

- It is difficult to get CPI\textsubscript{offchip} directly from the PMU events
  - CPI\textsubscript{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\textsubscript{offchip} is only needed to calculate T\textsubscript{offchip}
- We can calculate T\textsubscript{offchip} directly as shown below
  - \[ T = T\textsubscript{onchip} + T\textsubscript{offchip} = CCNT/ f\text{cpu} \]
  - \[ T\textsubscript{offchip} = CCNT/ f\text{cpu} - T\textsubscript{onchip} \]
Prediction Error Adjustment (I)

- Error adjustment

\[ q^{t-1} \quad q^t \quad q^{t+1} \]

\[ T^{t-1} \quad T^t \quad T^{t+1} \]

; quantum sequence

\[ T_{\text{exp}}^{t-1} \quad T_{\text{exp}}^t \quad T_{\text{exp}}^{t+1} \]

; ET at \( f_{\text{max}} \)

\[ T_{\text{act}}^{t-1} \quad T_{\text{act}}^t \quad T_{\text{act}}^{t+1} \]

; expected ET with a given \( PF_{\text{loss}} \)

\[ S^{t-1} \quad S^t \quad S^{t+1} \]

; actual ET (slack generation)

\[ q^{t-1} \quad q^t \quad q^{t+1} \]

\[ T^{t-1} \quad T^t \quad T^{t+1} \]

\[ T_{\text{exp}}^{t-1} \quad T_{\text{exp}}^t \quad T_{\text{exp}}^{t+1} \]

\[ T_{\text{act}}^{t-1} \quad T_{\text{act}}^t \quad T_{\text{act}}^{t+1} \]

\[ S^{t-1} = T_{\text{exp}}^{t-1} - T_{\text{act}}^{t-1} \]

\[ S^t = T_{\text{act}}^t + T_{\text{act}}^{t-1} - T_{\text{act}}^{t+1} = T_{\text{exp}}^{t-1} - T_{\text{act}}^{t-1} + S^{t-1} \]

\[ S^{t+1} = T_{\text{exp}}^{t+1} + T_{\text{exp}}^t + T_{\text{exp}}^{t-1} - T_{\text{act}}^{t+1} - T_{\text{act}}^t - T_{\text{act}}^{t-1} = T_{\text{exp}}^{t+1} - T_{\text{act}}^{t+1} + S^t \]

ET : Execution time

\[ T_{\text{exp}}^k = T^k (1 + PF_{\text{loss}}) \quad (k = t-1, t, t+1) \]

Prediction Error Adjustment (II)

- Target frequency selection
  - without adjustment
    \[ f^{t+1} = \frac{f_{\text{max}}}{1 + PF_{\text{loss}}} \left( 1 + \beta^t \cdot \left( \frac{f_{\text{max}}}{f^t} \right) \right) \]
    \[ \beta^t \cdot \frac{T_{\text{offchip}}}{T_{\text{onchip}}} \]
    \[ f^{t+1} = \frac{f_{\text{max}}}{1 + PF_{\text{loss}}} \left( 1 + \beta^t \cdot \left( \frac{f_{\text{max}}}{f^t} \right) \right) \]
  - with adjustment
    \[ f^{t+1} = \frac{f_{\text{max}}}{1 + PF_{\text{loss}}} \left( 1 + \beta^t \cdot \left( \frac{f_{\text{max}}}{f^t} \right) \right) + PF_{\text{loss}} \cdot \frac{S^t}{T_{\text{onchip}}} \cdot \left( \frac{f_{\text{max}}}{f^t} \right) \]
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
- $T_{\text{onchip}}$ and $T_{\text{offchip}}$ are calculated as:

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

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

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

$$f_{t+1} = \min\left( \frac{f_{\text{max}}}{1 + \beta \cdot \left( \frac{1}{f_{\text{max}}} \right)} \cdot \left[ 1 + \beta \cdot \left( \frac{f_{\text{max}}}{T_{\text{onchip}}} \right) \right] \right)$$

Implementation (I)

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

```
Kernel space

<table>
<thead>
<tr>
<th>&quot;proc&quot; interface module</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy module</td>
</tr>
<tr>
<td>PMU access module</td>
</tr>
<tr>
<td>DVFS module</td>
</tr>
<tr>
<td>XScale processor</td>
</tr>
</tbody>
</table>

Linux scheduler

external PF$_{loss}$ input (ex, battery status or user request)
```
Implementation (II)

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

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Voltage (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>333</td>
<td>0.91</td>
</tr>
<tr>
<td>400</td>
<td>0.99</td>
</tr>
<tr>
<td>466</td>
<td>1.05</td>
</tr>
<tr>
<td>533</td>
<td>1.12</td>
</tr>
<tr>
<td>600</td>
<td>1.19</td>
</tr>
<tr>
<td>666</td>
<td>1.26</td>
</tr>
<tr>
<td>733</td>
<td>1.49</td>
</tr>
</tbody>
</table>

Experimental Results (I)

- Power consumption vs. performance degradation
  - without OL-DVFS
  - with OL-DVFS

\[
\Delta V = V_{DUT} - V_1
\]

\[
I = \frac{\Delta V}{R}
\]

\[
P = I \times V_1
\]
**Experimental Results (II)**

- Comparison of the actual $\text{PF}_{\text{loss}}$ with the target performance loss

![Graph showing comparison of actual performance loss with target performance loss.](image)

**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