Power Management
Motivation for Power Management

- Power consumption is a critical issue in system design today
  - Mobile systems face battery life issues
  - High performance systems face heating issues

$7B for powering and cooling data centers in the US
Intuition

- Systems and components are:
  - Designed to deliver *peak performance*, but …
  - Not needing peak performance most of the time

- Dynamic Power Management (DPM)
  - Shut down components during idle times

- Dynamic Voltage Frequency Scaling (DVFS)
  - Reduce voltage and frequency of components
Power and Energy Relationship

\[ E = \int P \, dt \]
Low Power vs. Low Energy

- Minimizing the **power consumption** is important for:
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - cooling

- Minimizing the **energy consumption** is important due to:
  - restricted availability of energy (mobile systems)
    - limited battery capacities (only slowly improving)
    - very high costs of energy (solar panels, in space)
  - dependability
  - long lifetimes, low temperatures
Dynamic Voltage Frequency Scaling (DVFS)

Power consumption of CMOS circuits (ignoring leakage):

\[ P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f \]

- \( \alpha \): switching activity
- \( C_L \): load capacitance
- \( V_{dd} \): supply voltage
- \( f \): clock frequency
Variable-voltage/frequency

From Intel’s Web Site
Basic Idea of DVFS

\[ E \propto N_{\text{cycle}} \cdot V_{DD}^2 \]

- **Power**
  - No power-down
    - 50MHz
    - 12.5x10^8 cycle
    - 5.0V
    - 31.25J
  - Power-down
    - 50MHz
    - 5x10^8 cycle
    - 5.0V
    - 12.5J
  - Dynamic voltage scaling
    - 20MHz
    - 5x10^8 cycle
    - 2.0V
    - 2.0J

\[ \rightarrow \text{Slow and Steady wins the race!} \]
Commercial DVFS Processors

- Transmeta Crusoe
- AMD K2+ (PowerNow Technology)
- Intel SpeedStep
- XScale
DVFS ::
PXA27x Clock Overview

External 13Mhz Oscillator

Ex L = 16, N = 2
Turbo Mode = 416Mhz

Core PLL

Peripheral PLL

Run Mode
Turbo Mode
Half Turbo
Memory/LCD
UART
I2C
MMC
……
DVFS :: How to Change?

- **Reconfigure PLL**
  - Modify a few registers
  - Operate in a completely new configuration
  - Take a lot of time (~ a few ms)

- **Normal<->Turbo**
  - Modify only CP14 register
  - Change between preset frequencies
  - Take only a little time (under us)

For detailed information, refer to PXA27x Developer’s Manual
### DVFS ::

#### PXA27x Operating Modes

<table>
<thead>
<tr>
<th>Core Run Freq (MHz)</th>
<th>CLKCFG[T]</th>
<th>Core Turbo Freq (MHz)</th>
<th>CLKCFG[H]</th>
<th>CLKCFG[L]</th>
<th>CCR[L][2:0]</th>
<th>System Bus (MHz)</th>
<th>CLK_MEM (MHz)</th>
<th>CCR[A][1:0]</th>
<th>SDClock&lt;2:1&gt;</th>
<th>SDRAM Clocks (MHz)</th>
<th>MDREF[K2DB2]</th>
<th>MDREF[K0DB4]</th>
<th>MDREF[K0DB2]</th>
<th>LCD (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13†</td>
<td>X</td>
<td>—</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>13</td>
<td>X</td>
<td>X</td>
<td>13</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>13 or 26††</td>
</tr>
<tr>
<td>91†††</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>45</td>
<td>0</td>
<td>91</td>
<td>0</td>
<td>45</td>
<td>1</td>
<td>22.5</td>
<td>1</td>
</tr>
<tr>
<td>104</td>
<td>0</td>
<td>104</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>104</td>
<td>1</td>
<td>104</td>
<td>1</td>
<td>104</td>
<td>0</td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>156</td>
<td>0</td>
<td>156</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>104</td>
<td>1</td>
<td>104</td>
<td>1</td>
<td>104</td>
<td>0</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>208</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>104</td>
<td>0</td>
<td>104</td>
<td>0</td>
<td>104</td>
<td>0</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>208</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>3</td>
<td>104</td>
<td>0</td>
<td>104</td>
<td>0</td>
<td>104</td>
<td>0</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>312</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>3</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>312</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>4</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>416</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>5</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>520</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>6</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>0</td>
<td>624†††††</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>6</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>208</td>
<td>1</td>
<td>52</td>
<td>1</td>
</tr>
</tbody>
</table>

*Notes:*
- †: Indicates special conditions or limitations.
- †††: Indicates default or baseline configurations.
- ††††: Indicates advanced or optimized modes.
DVFS :: Usage

- dvfm
  - voltage and frequency scaling utility

Usage
- # dvfm l_value 2n_value fast_bus_mode turbo_mode mem_clk_conf

Examples
- # dvfm 16 5 1 2 1
  - 520Mhz Turbo Mode
- # dvfm 16 3 1 2 1
  - 312Mhz Normal Mode
# Xscale DVFS Settings

<table>
<thead>
<tr>
<th>V-f setting (MHz/V)</th>
<th>Active Power (mW)</th>
<th>Idle Power (mW)</th>
<th>dvfm setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>624/1.5</td>
<td>925</td>
<td>260</td>
<td>dvfm 16 6 1 2 1</td>
</tr>
<tr>
<td>520/1.5</td>
<td>747</td>
<td>222</td>
<td>dvfm 16 5 1 2 1</td>
</tr>
<tr>
<td>416/1.4</td>
<td>570</td>
<td>186</td>
<td>dvfm 16 4 1 2 1</td>
</tr>
<tr>
<td>312/1.3</td>
<td>390</td>
<td>154</td>
<td>dvfm 16 3 1 2 1</td>
</tr>
<tr>
<td>208/1.2</td>
<td>279</td>
<td>129</td>
<td>dvfm 16 2 1 2 1</td>
</tr>
<tr>
<td>104/1.1</td>
<td>116</td>
<td>64</td>
<td>dvfm 8 2 1 2 1</td>
</tr>
</tbody>
</table>
Successful DVFS Technique

1. Understand *workload variations* of your target

2. Devise efficient ways to detect them

3. Devise efficient ways to utilize the detected *workload variations* using available *H/W supports*
Non Real-Time Jobs

- No timing constraints
- No periodic executions
- Unknown execution time

It is hard to predict the future workload!!
DVFS for Non Real-Time Jobs

Basic Approach:

- Predict workload based on history information
- Usually based on *some variations of interval scheduler*
  - PAST, FLAT
  - LONG_SHORT, AGED_AVERAGE
  - CYCLE, PATTERN, PEAK
Key Question

How can we predict the future workload?

- Based on long term history:
  - Hard to adapt quickly for the changed workload
- Based on short term history:
  - Too many clock/voltage changes
PAST

- Looking a fixed window into the past

- Assume the next window will be like the previous one

- If the past window was
  - mostly busy ⇒ increase speed
  - mostly idle ⇒ decrease speed
Example: PAST

Utilization = \frac{\text{busy time}}{\text{window size}}
FLAT

- Try to smooth speed to a global average

- Make the utilization of next window=\langle\text{const}\rangle
  - Set speed fast enough to complete the predicted new work being pushed into the coming window
Example: FLAT

$<\text{Const}> = 0.7$
LONG-SHORT

- Look up the last 12 windows
  - Short-term past : 3 most recent windows
  - Long-term past : the remaining windows

- Workload Prediction
  - The utilization of next window will be a weighted average of these 12 windows’ utilizations
Example: LONG-SHORT

utilization = \# cycles of busy interval / window size

\[
\frac{0 + 0.3 + 0.5 + 1 + 1 + 1 + 0.8 + 0.5 + 0.3 + 4(0.1 + 0 + 0)}{9 + 4(3)} = 0.276
\]

\[f_{clk} = 0.276 \times f_{max}\]
AGED-AVERAGE

- Employs an exponential-smoothing method

- Workload Prediction
  - The utilization of next window will be a weighted average of all previous windows’ utilizations
    - geometrically reduce the weight
Example: AGED_AVERAGE

utilization = \# cycles of busy interval / window size

average = \frac{1}{3} \cdot 0 + \frac{2}{9} \cdot 0 + \frac{4}{27} \cdot (0.1) + \frac{8}{81} \cdot (0.3) + \cdots

f_{clk} = \text{average} \times f_{\text{max}}
PATTERN

- Workload Prediction
  - Convert the n-most recent windows’ utilizations into a pattern in alphabet {A, B, C, D}.
  - Find the same pattern in the past
  - Use the pattern to predict utilization
Example: PATTERN

Pattern = ABCDD

Pattern = ABCD

Predict: The next utilization will be D
Dynamic Power Management

- Power manageable components support multiple power states. Eg:
  - Active
  - Off

\[ P_{tr} = P_{off->active} + P_{active->off} \]

\[ T_{tr} = T_{off->active} + T_{active->off} \]
Power Manageable Components

- **Working**: 1.8 W (spinning + IO)
  - Read/write:
  - IO complete
- **Idle**: 0.8 W (spinning)
- **Standby**: 0.2 W (stop spinning)
  - Spin up: 3.8 W, 2.5 sec
  - Shut down: 1 W, 0.8 sec
Energy/Performance Tradeoff for DPM

Workload

State

Request

Request

T_{\text{idle}}

Power

P_{\text{active}}

T_{\text{active} \rightarrow \text{off}}

T_{\text{off} \rightarrow \text{active}}

T_{\text{of f}}

P_{\text{active} \rightarrow \text{off}}

P_{\text{off} \rightarrow \text{active}}

Time

Shutdown Cost

Wakeup Cost
Break Even Time

- Minimum idle time for amortizing the cost of component shutdown

\[ T_{\text{off}} P_{\text{off}} + T_{\text{tr}} P_{\text{tr}} = T_{\text{idle}} P_{\text{active}} \]

\[ T_{\text{be}} = T_{\text{idle}} = T_{\text{tr}} + T_{\text{off}} \]
DPM Applicability

- If idle period length < $T_{be}$
  - Not possible to save energy by turning off
- Need accurate estimation of upcoming idle periods:
  - Underestimation: potential energy savings lost
  - Overestimation: performance delay + energy loss
- Challenge: Manage energy/performance tradeoff
DPM Policies

- Control procedures that take DPM decisions
- Can be implemented in hardware or software
  - Software offers higher degree of configurability for complex systems

**Goal**: Maximize energy savings while minimizing performance delay
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
Timeout Policies

- Use elapsed idle time to predict the total idle period duration
  - Assumption:
    - \( P(T_{idle} > (T_o + T_{be}) | T_{idle} > T_o) \approx 1 \)
  - \( T_o \) is referred to as the timeout
    - Can be fixed or adaptive
Fixed Timeout Policies

- $T_o = T_{be}$ is $2$-competitive (Karlin et al, SODA’90)
  - Energy consumption in worst case is twice that of oracle policy

In this use case, energy consumption of Karlin’s algorithm would be twice that of oracle policy. If $T_o = T_{tr}$, oracle policy would sleep in first half and transition to active in second half. $T_{tr}$ and $P_{tr} > P_{active}$.

Energy consumption (Karlin) = $P_{active} \cdot T_{be} + P_{tr} \cdot T_{tr} = 2P_{tr} \cdot T_{tr}$.
Fixed Timeout Policy

- However, the approach limited to components with 2 power states
- *Irani et al, TECS’03* extend this work for components with multiple power states
  - Define a sets of timeouts: one for each state
  - Enter the low power state as its respective timeout expires
Fixed Timeout Policy

This algorithm is 2-competitive (Irani et al, TECS’03)
Adaptive Timeout Policy

- Dynamically adjust $T_o$ based on their observation of the workload

- *Douglsis et al, USENIX’95* propose several heuristics to adapt timeout. Eg:
  - Initialize $T^n_o$ to $T_{be}$
  - Observe length of idle period $T_{idle}$
  - If $T^n_{idle} > (T^n_o + T_{be})$, then $T^{n+1}_o = T^n_o - x$
  - Else, $T^{n+1}_o = T^n_o + x$

- Set floor and ceiling values to avoid getting too aggressive or conservative
Timeout Policies

Advantages
- Extremely simple to implement
- Safety (in terms of performance delay) can be easily improved by increasing $T_o$

Disadvantages
- Waste energy while waiting for $T_o$ to expire
- Heuristic in nature: no guarantee on energy savings or performance delay
- Always incur performance penalty on wakeup: no mechanism to wake before request arrival
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
Predictive Policies

- Take the DPM decision as soon as the idle period begins by predicting the length of upcoming idle period
  - If $T_{\text{pred}} > T_{\text{be}}$, perform shutdown
  - Else, stay awake

- Addresses the *energy wastage* issue of timeout policies while waiting for timeout to expire
Predictive Policies

- Hwang et al propose an online predictive policy
  - Uses exponential average of previous idle period lengths to perform prediction:
    \[ T_{n \text{pred}}^n = \alpha T_{n-1 \text{idle}}^n + (1-\alpha) T_{n-1 \text{pred}}^n \]
  - Value of \( \alpha \) controls the tradeoff between recent and past history
Predictive Policies

- Chung et al use adaptive learning trees
- Capable of managing multiple power states
  - Transforms sequence of optimal decisions in previous idle periods into discrete events
  - Maintains a window of previous events
  - Adaptive Learning Tree updates PCL and tree structure

![Diagram of Predictive Policies]

Prediction for $t_n$
Predictive Policies

- **Advantages**
  - More aggressive than timeout policies

- **Disadvantages**
  - Depend a lot on correlation between past and future events
  - Tend to be aggressive in shutdown and hence higher performance latency
  - Heuristic with no performance guarantees
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
Stochastic Policies

- Try to derive optimal policies for the given power and performance constraints
  - Referred to as *policy optimization*
- Model the system and workload as stochastic processes
- Policy optimization reduces to a stochastic optimization problem
System Model

- Service Requester (SR)
- Service Queue (SQ)
- Service Provider (SP)
- Power Manager (PM)
Discrete Time Markov Model  
(Paleologo et al, DAC’98)

- Models the system using Markov decision processes
- Discrete time setting (time slots)
  - System composition of SP, SR, SQ models
  - Defines cost metrics for each state and command pair

![Discrete Time Markov Model Diagram](image)
Power Manager

- PM observes the system state (S X R X Q) every time slot and issues command ‘a’
- Markovian stationary policies optimal for this system model
- Policy derived through policy optimization under given energy/performance constraints

\[
\begin{pmatrix}
  s_{\text{on}} & s_{\text{off}} \\
  x_1 & 0.4 & 0.6 \\
  x_2 & 0.2 & 0.8 \\
  x_3 & 0.5 & 0.5 \\
  x_4 & 1.0 & 0.0 \\
  x_5 & 0.4 & 0.6 \\
  x_6 & 0.8 & 0.2 \\
  x_7 & 0.8 & 0.2 \\
  x_8 & 1.0 & 0.0
\end{pmatrix}
\]

(Benini et al, TCAD’99)
Discrete Time Model

- **Advantages:**
  - Optimal policy $\Rightarrow$ no longer heuristic

- **Disadvantages**
  - Discrete time $\Rightarrow$ high overhead
  - Optimal only if the modeling assumptions hold
    - Geometric state transition times
    - Stationary workload
Continuous Time Markov Model
(Qui et al, TCAD’01)

- Model the system in continuous time space using CTMDP
- Cost associated with commands and states
- Policy optimization done to derive optimal policy
- Event driven policy, i.e. when system state changes
  - State transition times exponential
Continuous Time Model

- Advantages:
  - Optimal policy
  - Event driven => lesser overhead

- Disadvantages
  - Optimal only if the modeling assumptions hold
    - Exponential state transition times
    - Stationary workload
Workload and Device Characteristics
Simunic et al, TCAD’01

- Request inter-arrival times -> depends on state of SP:
  - If idle: pareto
  - If active: exponential

- State transition times of SP:
  - Follows uniform distribution

Important from DPM perspective

(Simunic et al, TCAD’01)
TISMDP Model

TISMDP Model

SMDP Model

Time Indexed SMDP Model

Allows modeling of non exponential distributions like Pareto and uniform for state transition probabilities

(Simunic et al, TCAD'01)
TISMDP Model

- **Advantages:**
  - Optimal and event driven policy
  - System model based on real world device and workload characteristics

- **Disadvantages**
  - Sub-optimal for non stationary workloads
    - Stationary workload assumption
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
Hybrid Policies

- Extension of policies from previous three classes
- Ideas:
  - Stochastic policies for non stationary workloads
  - Perform selection among multiple DPM policies
Stochastic Policies for non stationary workloads

- Chung et al, TC’02 extend the DTMDP model (TCAD’99) for handling non stationary workloads

- Key ideas:
  - Policy pre-characterization
  - Parameter learning
  - Policy interpolation
Policy Pre-characterization

(Chung et al, TC’02)
Parameter Learning

At time $T_n$

\[
\begin{array}{ccccccc}
\text{SR} & W(0) & W(1) & W(2) & W(3) & W(k-1) & W(k) \\
\hline
0 & 1 & 1 & 1 & \ldots & 0 & 1 \\
\end{array}
\]

At time $T_{n+1}$

\[
\begin{array}{ccccccc}
\text{SR} & W(0) & W(1) & W(2) & W(3) & W(k-1) & W(k) \\
\hline
0 & 0 & 1 & 1 & \ldots & 0 & 0 \\
\end{array}
\]

Dynamically estimates $R_0^\wedge$ and $R_1^\wedge$

Uses Maximum Likelihood Estimation
Policy Interpolation

Performs linear interpolation on selected decision tables for probability distribution of actions

(Chung et al, TC’02)
Policy interpolation model

- **Advantages:**
  - Takes non stationary workloads into account
    - Adapts with changing workloads
- **Disadvantages**
  - Not globally optimal
  - Discrete time model $\Rightarrow$ overhead
  - Markovian workload assumption
Selection among multiple policies

- Dhiman et al, ICCAD’06 propose a framework for selection among multiple policies
- Based on the observation that different policies outperform each other under different workloads
- Employ an online learning algorithm to perform policy selection
- Learning algorithm provides performance bounds on convergence to the best suited policy
Online Learning for DPM

DPM Policies (Working Set)

DPM 1  DPM 2  DPM 3  ……  DPM n

Selected policy manages power for the idle period

Evaluates performance of all policies for that idle period

Selects the best performing policy for managing power

Device

DPM Controller
Online learning based model

- Advantages:
  - Adapts with changing workloads
    - Controller converges to the best suited policy

- Disadvantages
  - Performance as good as that of the best policy in the set
## Quantitative Comparison

(Lu et al)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$P$</th>
<th>$N_{sd}$</th>
<th>$N_{wd}$</th>
<th>$T_{88}$</th>
<th>$T_{bs}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>off-line</td>
<td>1.64</td>
<td>164</td>
<td>0</td>
<td>166</td>
<td>0</td>
</tr>
<tr>
<td>SM</td>
<td>1.92</td>
<td>156</td>
<td>25</td>
<td>147</td>
<td>18.2</td>
</tr>
<tr>
<td>CA</td>
<td>1.94</td>
<td>160</td>
<td>15</td>
<td>142</td>
<td>17.6</td>
</tr>
<tr>
<td>SW</td>
<td>1.97</td>
<td>168</td>
<td>26</td>
<td>134</td>
<td>18.7</td>
</tr>
<tr>
<td>$\tau = 30$</td>
<td>2.05</td>
<td>147</td>
<td>18</td>
<td>142</td>
<td>30.0</td>
</tr>
<tr>
<td>LT</td>
<td>2.07</td>
<td>379</td>
<td>232</td>
<td>62</td>
<td>5.7</td>
</tr>
<tr>
<td>ATO3</td>
<td>2.09</td>
<td>147</td>
<td>26</td>
<td>138</td>
<td>29.9</td>
</tr>
<tr>
<td>ATO1</td>
<td>2.19</td>
<td>141</td>
<td>37</td>
<td>135</td>
<td>27.6</td>
</tr>
<tr>
<td>ATO2</td>
<td>2.22</td>
<td>595</td>
<td>430</td>
<td>41</td>
<td>4.1</td>
</tr>
<tr>
<td>$\tau = 120$</td>
<td>2.52</td>
<td>55</td>
<td>3</td>
<td>238</td>
<td>120.0</td>
</tr>
<tr>
<td>DM</td>
<td>2.60</td>
<td>105</td>
<td>39</td>
<td>130</td>
<td>48.9</td>
</tr>
<tr>
<td>EA</td>
<td>2.99</td>
<td>595</td>
<td>503</td>
<td>30</td>
<td>7.6</td>
</tr>
<tr>
<td>always-on</td>
<td>3.48</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>