POWER MANAGEMENT
AND ENERGY EFFICIENCY

* Adopted “Power Management for Embedded Systems, Minsoo Ryu”
Need for Power Management

- **Power consumption matters**
- **PCs**
  - Energy cost
  - Thermal dissipation
- **Mobile devices**
  - Battery lifetime
  - Thermal dissipation
- **Server systems**
  - Energy cost
  - Electrical infrastructure
  - Power usage effectiveness
Power and Performance

- Power $\propto$ voltage$^2$ * clock
- Clock $\propto$ voltage
- Therefore, Power $\propto$ clock$^3$
- Already server processors reached 150 Watt TDP
Power Consumers in a System

- **Processors**
  - Dominate power consumption
  - Usually consume 100 watts out of 300 watts

- **Memory**
  - Significant contributor

---

*Dual 4-core Intel Xeon®, 48GB DDR3 (12 DIMMs), SPEC CPU2006, all cores active. Measured AC power, analytically modeled memory power.*
Power Consumers in a System

- **Storages**
  - A server HDD consumes 5 to 10 watts
  - A laptop HDD consumes 1 to 5 watts
  - A SATA SSD consumes 1 to 5 watts
  - An NVME consumes 30~ watts

- **NIC**
  - A 10Gbps NIC consumes 5 to 20 watts

- **Peripherals**
  - Insignificant
Idle Power Consumption

- Only 30% of servers in data centers are fully utilized while keeping the other 70% in idle state.
- Idle servers consume between 60% and 66% of the peak load power consumption.

### Table: Idle Power Consumption Breakdown of Servers

<table>
<thead>
<tr>
<th>Component</th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>11.8 W</td>
<td>35.5 W</td>
<td>21.42 W</td>
</tr>
<tr>
<td>Memories</td>
<td>7.6 W</td>
<td>9.8 W</td>
<td>19.72 W</td>
</tr>
<tr>
<td>Hard Disks</td>
<td>2.7 W</td>
<td>2.3 W</td>
<td>1.13 W</td>
</tr>
<tr>
<td>Mainboard</td>
<td>45 W</td>
<td>70 W</td>
<td>70 W</td>
</tr>
<tr>
<td>Fans</td>
<td>27 W</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PSU</td>
<td>19 W</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>113 W</strong></td>
<td><strong>118 W</strong></td>
<td><strong>112 W</strong></td>
</tr>
</tbody>
</table>
Two Dimensions on Power Management

- Power management when the system is idle
  - Select the most efficient idle state
- Power management when the system is active
  - Dynamically change operating frequency and/or voltage
APM and ACPI

- **APM (Advanced Power Management)**
  - Activated when system becomes idle
    - Screen saver → sleep → suspend
  - Controlled by firmware (BIOS)
    - Need reboot for reconfiguration
  - OS has no knowledge

- **ACPI (Advanced Config. and Power Interfaces)**
  - Controlled by OS
  - First released in 1996 by Compaq, HP, Intel and MS
ACPI

- **Standard interface specification**
  - Brings power management under the control of the operating system
  - The specification is central to Operating System-directed configuration and Power Management (OSPM)

<table>
<thead>
<tr>
<th>Applications</th>
<th>OS Power Management</th>
<th>Software drivers</th>
<th>Hardware: CPU, BIOS etc.</th>
</tr>
</thead>
</table>

Real-Time Computing and Communications Lab., Hanyang University
http://rtcc.hanyang.ac.kr
ACPI Functions

- **System power management**
  - The entire computer

- **Processor power management**
  - When OS is idle but not sleeping, it puts processors in low-power states

- **Device power management**
  - ACPI tables describe motherboard devices, their power states, the power planes the devices are connected to
Firmware-Level ACPI Architecture

- Three components
  - ACPI tables
    - Contain definition blocks that describe all the hardware that can be managed through ACPI
    - Include both data and machine-independent byte-code
    - OS must have an interpreter for the AML bytecode
  - ACPI BIOS
    - Performs basic management operations on the hardware
    - Include code to help boot the system and to put the system to sleep or wake it up
  - ACPI registers
    - A set of hardware management registers defined by the ACPI specification
Firmware-Level ACPI Architecture
ACPI States

Diagram showing the ACPI states, including Legacy, G0 (S0) - Working, G1 - Sleeping, G2 (S5) - Soft Off, and G3 - Mech Off, with transitions and power states for various hardware components.
Global States

- **G0: Working (S0)**
  - Processor power states (C-state): C0, C1, C2, C3

- **G1: Sleeping (e.g., suspend, hibernate)**
  - Sleep State (S-state): S0, S1, S2, S3, S4

- **G2: Soft off (S5)**
  - Almost the same as G3 Mechanical Off, except that the power supply unit (PSU) still supplies power at a minimum.
  - Other components may remain powered so the computer can "wake" on input from the keyboard, clock, modem, LAN, or USB device.

- **G3: Mechanical off**
Processor States (C-State)

- Global state is G0 (working)
- Four processor states
  - C0: Operating
    - Performance state (P-State)
    - P0: highest performance, highest power
    - P1 ~ Pn: lower performance, lower power
  - C1: Halt
    - The processor is not executing instructions, but can return to an executing state essentially instantaneously
  - C2: Stop-Clock (optional)
    - The processor maintains all software-visible state, but may take longer to wake up
  - C3: Sleep (optional)
    - The processor does not need to keep its cache coherent, but maintains other state
## Processor States (C-State)

- Intel Pentium M at 1.6 Ghz

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Voltage</th>
<th>P-State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6 GHz</td>
<td>1.484 V</td>
<td>P0</td>
</tr>
<tr>
<td>1.4 GHz</td>
<td>1.420 V</td>
<td>P1</td>
</tr>
<tr>
<td>1.2 GHz</td>
<td>1.276 V</td>
<td>P2</td>
</tr>
<tr>
<td>1.0 GHz</td>
<td>1.164 V</td>
<td>P3</td>
</tr>
<tr>
<td>800 MHz</td>
<td>1.036 V</td>
<td>P4</td>
</tr>
<tr>
<td>600 MHz</td>
<td>0.956 V</td>
<td>P5</td>
</tr>
</tbody>
</table>
Device States (D-State)

- The device states D0–D3 are device-dependent
  - D0: Fully On
    - The operating state
  - D1 and D2
    - Intermediate power-states whose definition varies by device
  - D3: Off
    - The device is powered off and unresponsive to its bus
    - D3 Hot: Aux power is provided
    - D3 Cold: No power provided
Sleeping States (S-State)

- Four sleeping states
  - S1: Power on Suspend (POS)
    - All the processor caches are flushed
    - The power to the CPU(s) and RAM is maintained
    - Wakeup takes about 1 ~ 2 seconds on desktops
  - S2: CPU powered off
    - Dirty cache is flushed to RAM (Often not used)
  - S3: Suspend to RAM (STR), or Standby, Sleep
    - RAM remains powered
    - Wakeup takes about 3 ~ 5 seconds on desktops
  - S4: Suspend to Disk (STD) or hibernation
    - All content of the main memory is saved to non-volatile memory such as a hard drive, and is powered down
Dynamic Voltage and Frequency Scaling

- Adjusting clock speed and operating voltage dynamically
- Most modern processors provide
- Low clock switching overhead
- Usually within a few µs.
Four Considerations for DVFS

- Workload amount
  - Adjust the processor frequency depending on the load

- Workload characteristics
  - Compute-intensive vs. memory-intensive

- Deadline constraints
  - Lowest possible frequency for meeting deadlines

- Load balancing
  - Migrate or scale?
Workload Amount and DVFS

- **Static approaches**
  - Performance policy
    - CPU runs at the maximum frequency regardless of load
  - Power save policy
    - CPU runs at the minimum frequency regardless of load

- **Dynamic approaches**
  - On demand policy
    - Increase the clock speed to the maximum frequency when the system load goes above the predefined threshold
    - Decrease the clock speed gradually when the system load becomes below the predefined threshold
  - Conservative policy
    - Gracefully increase the CPU speed rather than jumping to the maximum speed
Workload Characteristics and DVFS

- Two types of workload
  - Compute-intensive
    - The program execution is exclusively bound to the processor
  - Memory-intensive
    - The program makes heavy access to memory
    - The processor would spend a significant fraction of the time waiting for memory

- A simple solution
  - High processor frequency and low memory frequency for compute-intensive load
  - Low processor frequency and high memory frequency for memory-intensive load
CPU VS Memory-Intensive

- Execution time variation
  - CPU frequency ranging from 733 MHz to 333 MHz
GPU and Memory-Intensive

- Compute-intensive applications
  - Dense matrix multiplication
  - Run on NVIDIA GeForce GTX 280 GPU
GPU and Memory-Intensive

- Memory-intensive applications
  - Dense matrix transpose
  - Run on NVIDIA GeForce GTX 280 GPU
Load Balancing and DVFS

- DVFS can be independently applied to each processor on multicore hardware
  - But this may not lead to optimal power saving from a global point of view

- A simple scenario
  - We need to decide whether to transfer a thread from processor A to an idle processor B, or increase the frequency of A
  - Compute $P_{\text{migrate\_from\_A\_to\_B}}$ and $P_{\text{increase\_A\_freq}}$
  - Transfer if $P_{\text{migrate\_from\_A\_to\_B}} < P_{\text{increase\_A\_freq}}$
  - Otherwise, increase the frequency of A
Case Study: Intel Core2 Duo E6850

**Table II**

<table>
<thead>
<tr>
<th>DVFS level</th>
<th>$V_{cpu}$</th>
<th>$f_{cpu}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>1.30</td>
<td>3.074</td>
</tr>
<tr>
<td>Level 2</td>
<td>1.25</td>
<td>2.852</td>
</tr>
<tr>
<td>Level 3</td>
<td>1.20</td>
<td>2.588</td>
</tr>
<tr>
<td>Level 4</td>
<td>1.15</td>
<td>2.281</td>
</tr>
<tr>
<td>Level 5</td>
<td>1.10</td>
<td>1.932</td>
</tr>
<tr>
<td>Level 6</td>
<td>1.05</td>
<td>1.540</td>
</tr>
</tbody>
</table>

**Table V**

<table>
<thead>
<tr>
<th>Level</th>
<th>$T_{uc}$</th>
<th>Total</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>2→1</td>
<td>4.77</td>
<td>9.77</td>
<td>30018</td>
</tr>
<tr>
<td>3→1</td>
<td>12.29</td>
<td>17.29</td>
<td>53141</td>
</tr>
<tr>
<td>3→2</td>
<td>5.95</td>
<td>10.95</td>
<td>33672</td>
</tr>
<tr>
<td>4→1</td>
<td>22.29</td>
<td>27.29</td>
<td>83894</td>
</tr>
<tr>
<td>4→2</td>
<td>14.69</td>
<td>19.69</td>
<td>60531</td>
</tr>
<tr>
<td>4→3</td>
<td>7.33</td>
<td>12.33</td>
<td>37921</td>
</tr>
<tr>
<td>5→1</td>
<td>34.81</td>
<td>39.81</td>
<td>122389</td>
</tr>
<tr>
<td>5→2</td>
<td>26.47</td>
<td>31.47</td>
<td>96733</td>
</tr>
<tr>
<td>5→3</td>
<td>27.33</td>
<td>32.33</td>
<td>99383</td>
</tr>
<tr>
<td>5→4</td>
<td>9.43</td>
<td>14.43</td>
<td>44361</td>
</tr>
<tr>
<td>6→1</td>
<td>57.68</td>
<td>62.68</td>
<td>192684</td>
</tr>
<tr>
<td>6→2</td>
<td>49.50</td>
<td>54.50</td>
<td>167525</td>
</tr>
<tr>
<td>6→3</td>
<td>31.89</td>
<td>36.89</td>
<td>113409</td>
</tr>
<tr>
<td>6→4</td>
<td>28.35</td>
<td>33.35</td>
<td>102531</td>
</tr>
<tr>
<td>6→5</td>
<td>12.14</td>
<td>17.14</td>
<td>52688</td>
</tr>
<tr>
<td>Downscale</td>
<td>0</td>
<td>5</td>
<td>15370</td>
</tr>
</tbody>
</table>

**Table VI**

<table>
<thead>
<tr>
<th>Level</th>
<th>Actual value (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{cpu}(V)$</td>
<td>$f_{cpu}(GHz)$</td>
</tr>
<tr>
<td>1.056</td>
<td>1.776</td>
</tr>
<tr>
<td>1.080</td>
<td>1.888</td>
</tr>
<tr>
<td>1.104</td>
<td>2.004</td>
</tr>
<tr>
<td>1.160</td>
<td>2.338</td>
</tr>
<tr>
<td>1.224</td>
<td>2.672</td>
</tr>
<tr>
<td>1.280</td>
<td>3.006</td>
</tr>
</tbody>
</table>
Case Study: Exynos 4210

### DVFS Overhead (Exynos 4210)

<table>
<thead>
<tr>
<th>DVFS level</th>
<th>$V_{cpu}$</th>
<th>$f_{cpu}$</th>
<th>DVFS level</th>
<th>$V_{cpu}$</th>
<th>$f_{cpu}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>1.2</td>
<td>1.4</td>
<td>Level 4</td>
<td>1.05</td>
<td>1.12870</td>
</tr>
<tr>
<td>Level 2</td>
<td>1.15</td>
<td>1.3122</td>
<td>Level 5</td>
<td>1.00</td>
<td>1.0327</td>
</tr>
<tr>
<td>Level 3</td>
<td>1.10</td>
<td>1.2218</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>$E_{uc}$ (µJ)</th>
<th>$E_{ir}$ (µJ)</th>
<th>Total (µJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1→2</td>
<td>1.00</td>
<td>-0.545</td>
<td>1.085</td>
</tr>
<tr>
<td>1→3</td>
<td>1.02</td>
<td>-0.47</td>
<td>1.175</td>
</tr>
<tr>
<td>1→4</td>
<td>3.09</td>
<td>-0.95</td>
<td>2.774</td>
</tr>
<tr>
<td>1→5</td>
<td>4.89</td>
<td>-1.16</td>
<td>4.35</td>
</tr>
<tr>
<td>2→3</td>
<td>0.57</td>
<td>-0.44</td>
<td>0.66</td>
</tr>
<tr>
<td>2→4</td>
<td>1.28</td>
<td>-0.61</td>
<td>1.21</td>
</tr>
<tr>
<td>2→5</td>
<td>3.63</td>
<td>-0.83</td>
<td>3.34</td>
</tr>
<tr>
<td>3→4</td>
<td>0.45</td>
<td>-0.35</td>
<td>0.55</td>
</tr>
<tr>
<td>3→5</td>
<td>1.47</td>
<td>-0.52</td>
<td>1.39</td>
</tr>
<tr>
<td>4→5</td>
<td>0.57</td>
<td>-0.26</td>
<td>0.66</td>
</tr>
<tr>
<td>2→1</td>
<td>-0.63</td>
<td>0.92</td>
<td>0.91</td>
</tr>
<tr>
<td>3→1</td>
<td>-1.32</td>
<td>1.96</td>
<td>1.27</td>
</tr>
<tr>
<td>3→2</td>
<td>-0.45</td>
<td>0.85</td>
<td>0.94</td>
</tr>
<tr>
<td>4→1</td>
<td>-2.33</td>
<td>2.90</td>
<td>1.19</td>
</tr>
<tr>
<td>4→2</td>
<td>-1.10</td>
<td>1.85</td>
<td>1.29</td>
</tr>
<tr>
<td>4→3</td>
<td>-0.42</td>
<td>0.78</td>
<td>0.81</td>
</tr>
<tr>
<td>5→1</td>
<td>-3.26</td>
<td>3.78</td>
<td>1.14</td>
</tr>
<tr>
<td>5→2</td>
<td>-1.83</td>
<td>2.78</td>
<td>1.48</td>
</tr>
<tr>
<td>5→3</td>
<td>-0.86</td>
<td>1.75</td>
<td>1.33</td>
</tr>
<tr>
<td>5→4</td>
<td>-0.34</td>
<td>0.72</td>
<td>0.74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>$T_{uc}$ (µs)</th>
<th>$T_{total}$ (µs)</th>
<th>$Cycles$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2→1</td>
<td>1.27</td>
<td>11.27</td>
<td>8051</td>
</tr>
<tr>
<td>3→1</td>
<td>2.95</td>
<td>12.95</td>
<td>9254</td>
</tr>
<tr>
<td>3→2</td>
<td>1.25</td>
<td>11.25</td>
<td>8039</td>
</tr>
<tr>
<td>4→1</td>
<td>5.30</td>
<td>15.30</td>
<td>10928</td>
</tr>
<tr>
<td>4→2</td>
<td>2.83</td>
<td>12.83</td>
<td>9165</td>
</tr>
<tr>
<td>4→3</td>
<td>1.24</td>
<td>11.24</td>
<td>8027</td>
</tr>
<tr>
<td>5→1</td>
<td>8.24</td>
<td>18.24</td>
<td>13029</td>
</tr>
<tr>
<td>5→2</td>
<td>5.17</td>
<td>15.27</td>
<td>10836</td>
</tr>
<tr>
<td>5→3</td>
<td>2.72</td>
<td>12.72</td>
<td>9088</td>
</tr>
<tr>
<td>5→4</td>
<td>1.21</td>
<td>11.21</td>
<td>8011</td>
</tr>
</tbody>
</table>
Linux Power Management Architecture

Amit Kucheria at 2011 Embedded Linux Conference
CPUIdle Architecture

User-level interfaces

Governors

Drivers

/l/sys/devices/system/cpu/cpu0/cpuidle
/l/sys/devices/system/cpu/cpu1/cpuidle

Generic cpuidle infrastructure

ladder

menu

acpi-cpuidle

halt_idle

ACPI processor driver

arch/platform specific drivers
CPUIdle Governors

- **Ladder Governor**
  - Takes a simple, step-wise approach to selecting an idle state
  - Enters the lightest state first, and will only move on to the next deeper state if a sleep was long enough

- **Menu Governor**
  - Picks the deepest possible idle state straight away
  - Considers the expected sleep time, latency requirements, previous C-state residency, etc
Idle Task

- When there are no runnable processes, and CFS schedules the idle task (PID 0)
Tickless Idle

- Traditional systems use a periodic interrupt 'tick'
  - Update the system clock
  - Tick requires wakeup from idle state
- Tickless idle eliminates the periodic timer tick when the CPU is idle
  - The CPU can remain in power saving states for a longer period of time, reducing the overall system power consumption
CPUFreq Architecture

User-level governors

In-kernel governors

CPU-specific drivers

powersaved

cpuspeed

performance

powersave

userspace

ondemand

cpufreq module (with /proc and /sys interfaces)

acpi-cpufreq

speedstep-centrino

... powernow-k8

ACPI processor driver
## CPUFreq Governor

<table>
<thead>
<tr>
<th>Governor</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>• Always set CPU to the highest frequency between scaling_min_freq and scaling_max_freq</td>
</tr>
<tr>
<td>Powersave</td>
<td>• Always set CPU to the lowest frequency between scaling_min_freq and scaling_max_freq</td>
</tr>
</tbody>
</table>
| Ondemand      | • Set frequency depending on the current usage  
|               | • Rapidly increase the frequency and gracefully decrease the frequency |
| Conservative  | • Basically operates like ondemand  
|               | • Gracefully increase and decrease the frequency |
| Userspace     | • Set CPU to the frequency using scaling_setspeed by user |
Ondemand Governor

\[ freq_{next} = freq_{min} + \frac{load}{100} \times (freq_{max} - freq_{min}) \]

up_threshold = 95%
Conservative Governor

\[
freq_{\text{next}} = \frac{5}{100} \times freq_{\text{max}}
\]

\[
freq_{\text{next}} = freq_{\text{curr}} - \frac{5}{100} \times freq_{\text{max}}
\]

Time

Freq.

max

min

Load

up\_threshold = 80%
down\_threshold = 20%
Basic Operations of CPUfreq

- Sample the processor utilization periodically
- Adjust frequency based on the utilization
- Adjust voltage based on frequency

Flowchart:

1. Measure CPU Utilization
2. Determine if utilization is greater than the upper threshold (U > Th_Up)
   - Yes: Adjust Frequency & Voltage
   - No: Determine if utilization is less than the lower threshold (U < Th_Down)
     - Yes: Adjust Frequency & Voltage
     - No: Wait for next sampling