Power and Energy Management

Advanced Operating Systems, Semester 2, 2011, UNSW
Etienne Le Sueur
etienne.lesueur@nicta.com.au
Outline

• Introduction,
• Hardware mechanisms,
• Some interesting research,
• Linux,
• Your questions.
Power vs. Energy

- Energy is a quantity, unit is Joules.
- Power is a rate, unit is Watts, which is Joules per second.
- The terms **power** and **energy** are often used interchangeably in computing. Don’t confuse them.
- However, they are closely related, high power draw leads to high energy consumption.
Why is energy important?

• Battery powered devices have limited energy resources
  – phones
  – laptops/tablets/netbooks
  – etc

• High cost of energy
  – data-centers (powering servers, air-conditioning for cooling, etc)
  – in the home, a 300W desktop can cost upwards of $1 per day if left on
  – environmental costs are difficult to quantify, but we know they probably exist. Some people are convinced, others aren’t. If not already, this will become the biggest political issue in the near future, and computing is at the forefront.
Why is power important?

• High power draw not only leads to high energy consumption but also heat generation.
• This heat needs to be dissipated somewhere.
  – It could be into the atmosphere,
  – or into a pool...
• Heat also needs to be managed to reduce hardware failures and to reduce cooling costs.
Computer hardware and power/energy

- What hardware makes up a computer system?
- What draws the most power?
- What consumes the most energy?
  - This varies depending on the type of system.
Hard drives

~5-10 W

~0.1-1 W
Memory (DRAM)

~5 W active, ~0 W idle

~20 W active, ~0 W idle
CPU

Core i7 quad, <10 W idle, 110 W full load!
Radio power output is measured in mW, plus some circuitry. Less than 5 W
The rest

Voltage converters, bus controllers, etc. Switching regulators are pretty efficient.

Maybe 95% efficient. That means that for a 400W power supply, 20W is being lost as heat inside that grey box.
Hang on a sec.

• That’s only inside the box. What about...

Insubstantial. These run off 2 AA batteries and last several months.

Likewise

Depends on size, but can be considerable. Lets say 30-50 W.
What about a pie graph?

- CPU: 48%
- Monitor: 22%
- Other: 9%
- Power supply: 9%
- Memory: 9%
- Hard drive: 4%
There’s a reason why there’s a giant heatsink on your desktop’s processor and why your pants catch on fire if your laptop is doing something strenuous.

The processor is the largest single drawer of power in a desktop or server system.
What about phones?

• ‘Embedded’ systems are a bit different
  – They spend a lot of time asleep in your pocket.
  – Their screens must be very bright to be visible in daylight.
  – The hardware is designed from the ground up to be efficient because it is absolutely critical.
  – Software is often the problem!
  – Carroll and Heiser determined that the most energy costly component of a phone is the LCD’s backlight. Furthermore, it depends on the type of screen! An OLED display draws a lot of power for a white screen, while a normal LCD draws more power to display a black screen.
  – So, turn the brightness down!

Aaron Carroll and Gernot Heiser
An analysis of power consumption in a smartphone
Proceedings of the 2010 USENIX Annual Technical Conference, Boston, MA, USA, June, 2010
So the CPU then...

• Mechanisms are available to reduce power draw
  – Clock gating
  – Clock duty cycle throttling
  – Dynamic voltage and frequency scaling (DVFS)
  – Idle states
  – Dynamic cache resizing
Clock gating
Clock duty cycle throttling

• Essentially skips clock cycles
• Does **not** reduce clock frequency, reduces *activity*
• Does **not** reduce supply voltage
• **Does** slow things down!
• It is used mainly for thermal management, as it can be enabled and disabled very quickly (low latency)
Aside: CMOS power draw

\[ P = \alpha C f V^2 + I_{\text{leakage}} V \]

In words: the power drawn by a CMOS circuit (i.e. a processor) is determined by alpha, the ‘activity’ of its circuit elements, C the capacitance of the wires and gates, f the frequency at which it is clocked and V the voltage being supplied. There is an additional ‘static’ term which is determined by leakage currents and temperature.
Dynamic voltage and frequency scaling (DVFS)

- So, if we could reduce the voltage, we’d get large reductions in power draw because of the quadratic dependence!
- But alas, we can’t just reduce the voltage, because the faster we want to clock CMOS gates, the higher we need the voltage.
- But we could reduce the frequency as well, right?
  - Exactly!
  - We get even less power draw,
  - But code takes longer to run.

This is called dynamic voltage and frequency scaling, or **DVFS**.
Idle states

• Take a look at top, Task Manager or Activity Monitor.
Servers in datacenters

Figure 1. Average CPU utilization of more than 5,000 servers during a six-month period. Servers are rarely completely idle and seldom operate near their maximum utilization, instead operating most of the time at between 10 and 50 percent of their maximum utilization levels.

Idle states

• Intel *knows* your CPU is idle.
  – Hence, they now give us very good idle *states*.

• Idle states are **the most** important mechanism that has come to CPU power management in recent years.

• There are often more than one idle state:
Different idle states for cores

- C0 is the active state, no power saving actions are taken.
- C1 is often called halt because you can use the halt instruction to enter it. Very little is done, maybe some clock gating.
- C3 is special because it flushes caches and powers down some areas of the core.
- C6 powers the whole core down to zero volts reducing leakage as well.
Power draw when idle
Package C-states

- What if all cores go into a deep state?
  - We can flush the shared L3 cache as well! The L3 cache is about half the total die area of a recent chip.
So, what to do with the mechanisms?

• We could
  – Use DVFS to slow the CPU down to some frequency, reducing the time spent in idle states, or
  – Run the CPU at full speed, increasing the time in idle states.

• But:
  – What CPU frequency should we choose?
  – Which idle state should we choose?
  – How often can/should we make transitions between states?
Naive DVFS

Assumption: The lowest frequency has the lowest power, therefore it should be the lowest energy!

This is a fallacy. All instructions take longer to execute at reduced frequency! The longer runtime could offset any energy saved by the reduced power draw!
Aside: performance under DVFS

Answer me some questions

• Do all instructions take the same number of cycles to complete?
• OK, so what sort of instructions take the longest?
• Yes! Memory-related instructions
  – On Load/Store architectures, it’s easy to work out which are memory related. Loads and Stores.
  – On x86, many instructions might result in accesses to memory. It’s not so easy... MOV$ can touch a lot of memory in a single instruction.
Aside: performance under DVFS

- On most systems, memory runs at a different clock rate.

- If we reduce the CPU frequency, relatively, memory gets faster.
What does this mean?

... some workloads can be very memory-bound
Performance of two different workloads

• GZIP is CPU-bound, MCF is memory-bound
Performance of two different workloads

Normalised CPU cycles for 164.gzip and 181.mcf

- 164.gzip (cpu-bound)
- 181.mcf (memory-bound)
Performance under DVFS

Performance under DVFS is very workload dependent
Estimating the effect of DVFS

• If we can estimate how long it will take to run some code at a particular frequency, and how much power the CPU will draw at that frequency, we can estimate the energy consumed.

• Then, we can use this information to make an informed choice about which frequency to run at.

• This is ideal in theory. However, in practice, it is very difficult to accurately model the behaviour of a complex CPU.
The intricacies of modelling

• First, we need to figure out what we want to predict.
  – Performance, i.e. cycles to execute at a particular frequency
  – Power draw at particular frequency

\[ \text{Energy} = \text{Power} \times \text{Time} \]

• Next, we decide on a structure for the model
• Next, we need to work out what we can actually measure.
  – Hardware performance counters, x86 has hundreds
Performance counters

- Most CPUs have a performance-monitoring unit (PMU)
- On x86, it can measure hundreds of event types, but
- You can’t measure them all at once.
  - There are a limited number of registers, usually 2 or 4, but sometimes more - the P4 had 20.
- You can measure things like:
  - Unhalted cycles
  - LLC cache misses
  - TLB misses
  - Instructions retired
  - Branch misses
  - etc.
- They can be split into kernel/user mode as well.
A model for execution time

• Time summed in frequency domains

\[ T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{mem}}{f_{mem}} \]

• Cycles in a domain can be measured or predicted with performance counters

\[ C_{mem} = \alpha PMC_1 + \beta PMC_2 + \ldots \]

• Actual CPU cycles (\(C_{tot}\) measured using CPU cycle counter)

\[ C_{cpu} = C_{tot} - \frac{f_{cpu}}{f_{mem}} C_{mem} \]
A model for execution time

• We can predict cycles at different frequencies now

\[ C'_{tot} = C_{tot} - \frac{f_{cpu}}{f_{mem}} C_{mem} + \frac{f'_{cpu}}{f'_{mem}} C_{mem} \]

• Relative performance

\[ s = \frac{f_{cpu}}{f'_{cpu}} \times \frac{C'_{tot}}{C_{tot}} \]
A model for power draw

• We know that

\[ P \propto fV^2 \]

• But power drawn by the CPU is workload dependent
  – Use the performance counters!

\[
P = V_{cpu}^2 (\gamma_1 f_{cpu} + \gamma_2 f_{mem}) + V_{cpu}^2 (\alpha_1 PMC_1 + \ldots + \alpha_n PMC_n) + \gamma_3 f_{mem} + \beta_1 PMC_1 + \beta_n PMC_n + P_{static}
\]
The intricacies of modelling

- CPUs, especially x86, are extremely complex.
  - Memory hierarchies are multi-level
  - Multiple cores share caches
  - There are hundreds of performance counters, we can only measure ~4 so which ones do we choose for our model?
  - What about effects that we can’t measure, like prefetching or other architecture optimisations?
Aside: Prefetching

- Memory access patterns can be predictable
- Memory is slow!
- The CPU tries to *fetch* operands before the CPU issues the instruction that needs them
- This basically uses extra memory bandwidth that the CPU isn’t using.
Model parameters: prefetching disabled
Model parameters: prefetching enabled
How well does it work?
How well does it work?
How does Linux do it?

Linux Kernel

cpufreq

ondemand  conservative  performance  ‘powersave’
How does Linux do it?

- Simply keeps the CPU at its **lowest** frequency

Will this save any energy?
How does Linux do it?

- cpufreq
  - ondemand
  - conservative
  - performance
  - ‘powersave’

• Simply keeps the CPU at its **highest** frequency

Will this save any energy?
How does Linux do it?

- Monitors system load at some interval (80ms default)
- If load is below a threshold, reduce frequency one step
- If load is above a threshold, increase frequency one step
How does Linux do it?

- Monitors system load at some interval (80ms default)
- If load is below a threshold, reduce frequency one step
- If load is above a threshold, increase frequency to max

Is this the correct way to use DVFS?
Load-based DVFS

**Assumption:** If the system is under-utilised, lowering the CPU frequency won’t affect our performance and we can save energy.

**This is a fallacy.** If we have awesome idle states, there is no point to this and it will only hurt our energy consumption.
A critical point

Low power != Low energy
The laws of diminishing returns

- Processor technology is changing **very** quickly
  - Transistor size and supply voltage
  - Better idle states
  - Larger caches
  - etc...

- How does this affect power management?
# OK, lets look at some numbers

<table>
<thead>
<tr>
<th>Die codename</th>
<th>Sledgehammer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>2003</td>
</tr>
<tr>
<td>Core count</td>
<td>1</td>
</tr>
<tr>
<td>Frequency range</td>
<td>0.8 - 2.0GHz</td>
</tr>
<tr>
<td>Voltage range</td>
<td>0.9 - 1.5</td>
</tr>
<tr>
<td>Process</td>
<td>130nm</td>
</tr>
<tr>
<td>TDP</td>
<td>89W</td>
</tr>
<tr>
<td>Die area</td>
<td>193mm²</td>
</tr>
<tr>
<td>Transistor count</td>
<td>106M</td>
</tr>
</tbody>
</table>
Reducing frequency can potentially save energy

Energy and runtime of 181.mcf

Energy and runtime (Normalized) vs. Frequency (GHz)

- Lower frequencies correspond to lower energy consumption and longer runtime.
- The chart illustrates the trade-off between energy and runtime at different frequencies.
A bit more recent...

<table>
<thead>
<tr>
<th>Die codename</th>
<th>Sledgehammer</th>
<th>Santa Rosa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>2003</td>
<td>2006</td>
</tr>
<tr>
<td>Core count</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Frequency range</td>
<td>0.8 - 2.0GHz</td>
<td>1.0 - 2.4GHz</td>
</tr>
<tr>
<td>Voltage range</td>
<td>0.9 - 1.5</td>
<td>0.9 - 1.35V</td>
</tr>
<tr>
<td>Process</td>
<td>130nm</td>
<td>90nm</td>
</tr>
<tr>
<td>TDP</td>
<td>89W</td>
<td>95W</td>
</tr>
<tr>
<td>Die area</td>
<td>193mm²</td>
<td>230mm²</td>
</tr>
<tr>
<td>Transistor count</td>
<td>106M</td>
<td>243M</td>
</tr>
</tbody>
</table>
Energy and runtime of 181.mcf on Santa Rosa

- Frequency (GHz)
- Normalised energy/runtime

- Energy 1 instance
- Runtime 1 instance
- Energy 2 instances
- Runtime 2 instances
Even more recent...

<table>
<thead>
<tr>
<th>Die codename</th>
<th>Sledgehammer</th>
<th>Santa Rosa</th>
<th>Shanghai</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>2003</td>
<td>2006</td>
<td>2009</td>
</tr>
<tr>
<td>Core count</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Frequency range</td>
<td>0.8 - 2.0GHz</td>
<td>1.0 - 2.4GHz</td>
<td>0.8 - 2.7GHz</td>
</tr>
<tr>
<td>Voltage range</td>
<td>0.9 - 1.5</td>
<td>0.9 - 1.35V</td>
<td>1.0 - 1.35V</td>
</tr>
<tr>
<td>Process</td>
<td>130nm</td>
<td>90nm</td>
<td>45nm</td>
</tr>
<tr>
<td>TDP</td>
<td>89W</td>
<td>95W</td>
<td>75W</td>
</tr>
<tr>
<td>Die area</td>
<td>193mm²</td>
<td>230mm²</td>
<td>285mm²</td>
</tr>
<tr>
<td>Transistor count</td>
<td>106M</td>
<td>243M</td>
<td>463M</td>
</tr>
</tbody>
</table>
Energy and runtime of 181.mcf on Shanghai

Shanghai
Last words

• Processor technology changes at an impressive rate
• They are getting more efficient all the time
  – Smaller, more efficient transistor designs
• Energy has become a critical design constraint
  – Understanding how systems consume energy will make you a better programmer.
• Go and get a power meter!
Questions?