18-447 Lecture 12: Energy and Power

James C. Hoe
Department of ECE
Carnegie Mellon University
# Midterm 1 Statistics

<table>
<thead>
<tr>
<th>points</th>
<th>P1 pareto (8)</th>
<th>P2 align (9)</th>
<th>P3 diagram (6)</th>
<th>P4 pipeline (18)</th>
<th>P5 assbl (14)</th>
<th>Total (55)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>3.4</td>
<td>3.6</td>
<td>4.2</td>
<td>7.8</td>
<td>10.2</td>
<td>29.2</td>
</tr>
<tr>
<td>std dev.</td>
<td>3.1</td>
<td>3.0</td>
<td>1.9</td>
<td>5.0</td>
<td>3.5</td>
<td>10.2</td>
</tr>
<tr>
<td>max</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>18</td>
<td>14</td>
<td>49</td>
</tr>
<tr>
<td>median</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>31</td>
</tr>
<tr>
<td>mode</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>min</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>
Diagnostic Power
Midterm 1 Histogram

C’ish ≥ 19

B’ish ≥ 29

A’ish ≥ 39
Problem 6

A scatter plot showing the relationship between actual values and guesses. The plot is divided into quadrants labeled D’ish, C’ish, B’ish, and A’ish.
Housekeeping

• Your goal today
  – a working understanding of energy and power
  – appreciate their significance in comp arch today

• Notices
  – Lab 2, due end of Tuesday’s lab (3/3)
  – HW 3, due Wednesday 3/4 Friday 3/6
  – Handout #10: Lab 3, due the week of 3/23

• Readings
  – Synthesis Lectures (advanced optional):
    • Comp Arch Techniques for Power-Efficiency, 2008
    • Power-Efficient Comp Arch: Recent Advances, 2014
First some intuitions
Energy and Power

- CMOS logic transitions involve charging and discharging of parasitic capacitances
- Energy (Joule) dissipated as resistive heat when “charges” flow from VDD to GND
  - takes a certain amount of energy per operation (e.g., addition, reg read/write, (dis)charge a node)
  - to the first order, energy $\propto$ amount of compute
- Power (Watt=Joule/s) is rate of energy dissipation
  - more op/sec then more Joules/sec
  - to the first order, power $\propto$ performance

Power worries usually more about heat removal
Heat and Thermal Resistance

- Resistive heat in the circuit must be removed in steadystate (www.youtube.com/watch?v=BSGcnRanYMM)
- Can summarize everything between circuit and ambient by characteristic $R_{\text{thermal}} = K/W$
  - Convey power $W$ in heat across temperature difference $K = T_{\text{circuit}} - T_{\text{ambient}}$
- To dissipate more power in circuit
  1. Let $T_{\text{circuit}}$ gets hotter (to a point)
  2. Turn-up AC $\Rightarrow$ lower $T_{\text{ambient}}$
  3. Better cooling $\Rightarrow$ lower $R$
- Economics/market driven choices
Work and Perf. from Joules and Watt

• Fastest without energy/power awareness won’t be fastest once constrained
  – power bounds performance directly
  – energy bounds work directly; want for lower J/op bounds performance indirectly

• Consider in context
  – mobile device: limited energy source, hard-to-cool form factor
  – desktop: cooler size and noise
  – data-center: electric bill, cooling capacity and cost

Ultimately driven by size/weight/$\$$
Some (first-order) nitty-gritty
Work and Runtime

- **Work**
  - scalar quantity for “amount of work” associated with a task
  - e.g., number of instructions to compute a SHA256 hash

- **\( T = \frac{Work}{k_{perf}} \)**
  - runtime to perform a task
  - \( k_{perf} \) is a scalar constant for the rate in which work is performed, e.g., “instructions per second”
Energy and Power

- $E_{\text{switch}} = k_{\text{switch}} \cdot \text{Work}$
  - “switching” energy associated with task
  - $k_{\text{switch}}$ is a scalar constant for “energy per unit work”

- $E_{\text{static}} = k_{\text{static}} \cdot T = k_{\text{static}} \cdot \text{Work} / k_{\text{perf}}$
  - “leakage” energy just to keep the chip powered on
  - $k_{\text{static}}$ is the so-called “leakage power”

  *Faster execution means lower leakage energy???

- $E_{\text{total}} = E_{\text{switch}} + E_{\text{static}} = k_{\text{switch}} \cdot \text{Work} + k_{\text{static}} \cdot \text{Work} / k_{\text{perf}}$

- $P_{\text{total}} = E_{\text{total}} / T = k_{\text{switch}} \cdot k_{\text{perf}} + k_{\text{static}}$

Static power can be 50% in high-perf processors
In Short

• \( T = \frac{\text{Work}}{k_{\text{perf}}} \) 
  less work finishes faster

• \( E = E_{\text{switch}} + E_{\text{static}} = \left( k_{\text{switch}} + k_{\text{static}}/k_{\text{perf}} \right) \cdot \text{Work} \)
  less work use less energy

• \( P = P_{\text{switch}} + P_{\text{static}} = k_{\text{switch}} \cdot k_{\text{perf}} + k_{\text{static}} \)
  power independent of amount of work

• Reality check
  – \textbf{Work} not a simple scalar, inst mix, dependencies ...
  – \textbf{k}'s are neither scalar nor constant

\( k_{\text{perf}}: \text{inst/sec} \)
\( k_{\text{switch}}: \text{J/inst} \)
\( k_{\text{static}}: \text{J/sec} \)
$k_{\text{switch}}, k_{\text{static}}, k_{\text{perf}}$ not independent

More complicated $\mu$arch increases $k_{\text{switch}}$ and $k_{\text{static}}$
Faster transistors increases $k_{\text{static}}$

Power $\approx \text{Perf}^\alpha > 1$
Why so important now?
Technology Scaling for Dummies

• Planned scaling occurs in discrete “nodes” where each is ~0.7x of the previous in linear dimension

• Take the same design, reducing linear dimensions by 0.7x (aka “gate shrink”) leads to **ideally**
  – die area = 0.5x
  – delay = 0.7x; frequency=1.43x
  – capacitance = 0.7x
  – Vdd = 0.7x (constant field) or 1x (constant voltage)
  – power = 0.5x (const. field) or 1x (const. voltage)

• Take the same area, then
  – transistor count = 2x
  – power = 1x (const field) or 2x (const voltage)
The Other Moore’s Law
Moore’s Law $\rightarrow$ Performance

- According to scaling theory
  @constant complexity ("gate-shrink"):  
  1x transistors at 1.43x frequency  
  $\Rightarrow$ 1.43x performance at 0.5x power

@max complexity:  
  2x transistors at 1.43x frequency  
  $\Rightarrow$ 2.8x performance at constant power

- Historically though, for high-perf CPUs
  - $\sim$2x transistors
  - $\sim$2x frequency (note: faster than scaling predicts)
  - all together, $\sim$2x performance at $\sim$2x power

Why?
Performance (In)efficiency

• To hit “expected” performance target
  – push frequency harder by deepening pipelines
  – used the 2x transistors to build more complicated microarchitectures so fast/deep pipelines don’t stall (i.e., caches, BP, superscalar, out-of-order)

• The consequence of performance inefficiency is the limit of economical cooling [ITRS]

2005, Intel P4 Tehas 150W

[Borkar, IEEE Micro, July 1999]
Moore’s Law without Dennard Scaling

2013 Intl. Technology Roadmap for Semiconductors

- logic density
- VDD

Under fixed power ceiling, more ops/second only achievable if less Joules/op?
Frequency and Voltage Scaling: run slower at lower energy-per-op
Frequency and Voltage Scaling

- Switching energy per transition is
  \[ \frac{1}{2}CV^2 \] (modeling parasitic capacitance)

- Switching power at \( f \) transitions-per-sec is
  \[ \frac{1}{2}CV^2f \]

- To reduce power, slow down the clock

- If clock is slower (\( f' \)), reduce supply voltage (\( V' \))
  too since transistors don’t need to be as fast
  - reduced switching energy, \( \frac{1}{2}CV^2 \rightarrow \frac{1}{2}CV'^2 \)
  - lower \( V' \) also reduced leakage current/power

18-447-S20-L12-S23, James C. Hoe, CMU/ECE/CALCM, ©2020
Frequency Scaling (by itself)

- If $\frac{\text{Work}}{k_{\text{perf}}} < T_{\text{bound}}$, we can derate performance by frequency scaling by a factor $s_{\text{freq}}$
  
  $\frac{(\text{Work}/k_{\text{perf}})}{T_{\text{bound}}} < s_{\text{freq}} < 1$

  s.t. $k_{\text{perf}}' = k_{\text{perf}} s_{\text{freq}}$

- $T' = \frac{\text{Work}}{(k_{\text{perf}} s_{\text{freq}})}$
  
  - $1/s_{\text{freq}}$ longer runtime

- $E' = (k_{\text{switch}} + \frac{k_{\text{static}}}{k_{\text{perf}} s_{\text{freq}}}) \cdot \text{Work}$
  
  - higher (leakage) energy due to longer runtime

- $P' = k_{\text{switch}} \cdot k_{\text{perf}} s_{\text{freq}} + k_{\text{static}}$
  
  - lower (switching) power due to longer runtime

Not such a good idea
Intel P4 660 Frequency Scaling: FFT\textsubscript{64K}

circa 2005, 90nm

\[ k_{\text{perf}} = 145 \text{ FFT64K/sec}; \quad k_{\text{switching}} = 0.24 \text{ J/FFT64K}; \quad k_{\text{static}} = 49.4\text{J/sec} \]
Intel P4 660 Frequency Scaling: FFT\textsubscript{64K}

- More energy-per-fft to run slower!!

\[ k_{\text{perf}} = 145 \text{ FFT64K/sec} \]
\[ k_{\text{switching}} = 0.24 \text{ J/FFT64K} \]
\[ k_{\text{static}} = 49.4 \text{ J/sec} \]
Frequency + Voltage Scaling

- Frequency scaling by $s_{freq}$ allows supply voltage to be scaled by a corresponding factor $s_{voltage}$
- $E \propto V^2$ thus
  - $k_{\text{switch}}'' = k_{\text{switch}} \cdot s_{voltage}^2$
  - $k_{\text{static}}'' = k_{\text{static}} \cdot s_{voltage}^{2\sim 3}$ ← very gross approximation

- $T'' = \frac{\text{Work}}{(k_{\text{perf}} \cdot s_{freq})}$
  - $1/s_{freq}$ longer runtime

- $E'' = (k_{\text{switch}} \cdot s_{voltage}^2 + k_{\text{static}} \cdot s_{voltage}^3 / k_{\text{perf}} \cdot s_{freq}) \cdot \text{Work}$
- $P'' = k_{\text{switch}} \cdot s_{voltage}^2 k_{\text{perf}} \cdot s_{freq} + k_{\text{static}} \cdot s_{voltage}^3$
  - superlinear reduction in power and energy to performance degradation
Intel P4 660 F+V Scaling: FFT$_{64K}$

circa 2005, 90nm
Intel P4 660 F+V Scaling: FFT\textsubscript{64K}

- Blue line: model freq. scaling only
- Orange line: model freq\&volt scaling, x\(^2\)
- Gray line: model freq\&volt scaling, fitted
- Yellow line: model freq\&volt scaling, x\(^3\)

Energy (mJoule) vs. Frequency (MHz)

circa 2005, 90nm
Parallelization:
run faster at lower energy-per-op
Cost of Performance in Power

Energy per Instruction Trends in Intel®
Microprocessors, Grochowski et al., 2006

Better to replace 1 of this by 2 of these;
Or N of these

Power ≈ Perf^{1.75}

486

Pentium 4

technology normalized power (Watt)

technology normalized performance (op/sec)
Parallelization

• Ideal parallelization over $N$ CPUs
  
  $T = \frac{Work}{k_{\text{perf}} \cdot N}$
  
  $E = \left( \frac{k_{\text{switch}} + k_{\text{static}}}{k_{\text{perf}}} \right) \cdot Work$
  $\quad N$-times static power, but $N$-times faster runtime

  $P = N \left( k_{\text{switch}} \cdot k_{\text{perf}} + k_{\text{static}} \right)$

• Alternatively, forfeit speedup for power and energy reduction by $s_{freq} = 1/N$ (assume $s_{voltage} \approx s_{freq}$ below)
  
  $T = \frac{Work}{k_{\text{perf}}}$
  
  $E'' = \left( \frac{k_{\text{switch}}}{N^2} + \frac{k_{\text{static}}}{(k_{\text{perf}} \cdot N)} \right) \cdot Work$
  
  $P'' = k_{\text{switch}} \cdot k_{\text{perf}} / N^2 + k_{\text{static}} / N$

• Also works with using $N$ slower-simpler CPUs
So what is the problem?

• “Easy” to pack more cores on a die to stay on Moore’s law for “aggregate” or “throughput” performance

• How to use them?
  – life is good if your $N$ units of work is $N$ independent programs $\Rightarrow$ just run them
  – what if your $N$ units of work is $N$ operations of the same program? $\Rightarrow$ rewrite as parallel program
  – what if your $N$ units of work is $N$ sequentially dependent operations of the same program? $\Rightarrow$ ??

How many cores can you use up meaningfully?
Moore’s Law Scaling with Cores

1970~2005

Little core

Big Core

Little core

Little core

Little core

Little core

Little core

Little core

2005~??

Little core

Little core

Little core

Little core

Little core

Little core

Little core

Little core
Remember: it is all about \textbf{Perf/Watt} and \textbf{Ops/Joules}

We talk about HW specialization in a later lecture