18-447 Lecture 12: Energy and Power

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

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

• Notices
  – Lab 2, due this week
  – HW 3, due next Wednesday
  – Handout #10: Lab 3, due the week of 3/26

• Readings
  – Synthesis Lectures (advanced optional):
    • Comp Arch Techniques for Power-Efficiency, 2008
    • Power-Efficient Comp Arch: Recent Advances, 2014
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
Make sure we agree what is what

- Electric bill is for energy (kilowatt-hour) not power
- “Bigger” phone battery holds more energy
- “Faster” charger refills battery’s energy in a shorter time, hence, higher power

  Chargers rated for max power not max energy

- For a given amount of energy (say a fully charged battery), lower-power device runs longer
- For a given device, bigger battery lasts longer
- World energy consumption > $10^{20}$ Joule in 1 year
- A nuclear generator generates $\sim 10^9$ Joule in 1 sec
Some 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

• \( T = \frac{\text{Work}}{k_{\text{perf}}} \)
  – runtime to perform a task
  – \( k_{\text{perf}} \) 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

- \( E = E_{\text{switch}} + E_{\text{static}} = \left( k_{\text{switch}} + \frac{k_{\text{static}}}{k_{\text{perf}}} \right) \cdot \text{Work} \)  
  
  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 ...
  - \( k \)'s are neither scalar nor constant

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

More complicated \( \mu \text{arch} \) increases \( k_{\text{switch}} \)
Faster transistors increases \( k_{\text{static}} \)

\[ \text{Power} \approx \text{Perf}^{\alpha > 1} \]
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:
    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

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
  \[ \frac{1}{2}CV^2 f \]

• 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
Frequency Scaling (by itself)

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

$$\left(\frac{\text{Work}}{k_{\text{perf}}}/T_{\text{bound}}\right) < s_{\text{freq}} < 1$$

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

• $E' = \left(k_{\text{switch}} + \frac{k_{\text{static}}}{k_{\text{perf}} s_{\text{freq}}]\right) \cdot \text{Work}$
  – 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_{64K}

![Graph showing frequency scaling with FFT_{64K}.](image)

- At 0.45GHz:
  - Frequency vs. runtime:
    - Frequency: 0.45GHz
    - Runtime (ms): 60
  - Frequency vs. cycle time:
    - Frequency: 0.45GHz
    - Cycle time (ns): 3.6GHz

circa 2005, 90nm
Intel P4 660 Frequency Scaling: FFT<sub>64K</sub>

- Leakage power
- Switching power

- $k_{perf} = 145$ FFT64K/sec
- $k_{switching} = 0.24$ J/FFT64K
- $k_{static} = 49.4$ J/sec
Intel P4 660 Frequency Scaling: FFT_{64K}

![Graph showing energy versus frequency.]

- $k_{\text{perf}} = 145$ FFT64K/sec
- $k_{\text{switching}} = 0.24$ J/FFT64K
- $k_{\text{static}} = 49.4$ J/sec

more energy-per-fft to run slower!!
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
  - $k_{\text{switch}}'' = k_{\text{switch}} \cdot s_{voltage}^2$
  - $k_{\text{static}}'' = k_{\text{static}} \cdot s_{voltage}^{2\sim3}$ \(\leftarrow\) 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\textsubscript{64K}

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

- model freq. scaling only
- model freq\&volt scaling, \(x^2\)
- model freq\&volt scaling, fitted
- model freq\&volt scaling, \(x^3\)

circa 2005, 90nm

能源（mJoule）

频率（MHz）
Parallelization:
run faster at lower energy-per-op
Cost of Performance in Power

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

Power ≈ Perf^{1.75}

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

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

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

- 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

2005~??
Remember: it is all about Perf/Watt and Ops/Joules

We talk about HW specialization in a later lecture