Review and Background
Amdahl’s Law

\[ \text{Speedup} = \frac{\text{time}_{\text{without enhancement}}}{\text{time}_{\text{with enhancement}}} \]

An enhancement speeds up fraction \( f \) of a task by factor \( S \)

\[ \text{time}_{\text{new}} = \text{time}_{\text{orig}} \cdot ( (1-f) + \frac{f}{S} \) \]

\[ S_{\text{overall}} = \frac{1}{(1-f) + \frac{f}{S}} \]

\[ \text{time}_{\text{orig}} \]

\[ (1 - f) \quad \frac{f}{S} \]

\[ \text{time}_{\text{new}} \]

\[ (1 - f) \quad \frac{f}{S} \]
The *Iron Law* of Processor Performance

\[
\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}
\]

Total Work In Program

Algorithms, Compilers, ISA Extensions

CPI or 1/IPC

Microarchitecture

1/f (frequency)

Microarchitecture, Process Tech

We will concentrate on CPI, others are important too!
Performance

- **Latency** (execution time): time to finish one task
- **Throughput** (bandwidth): number of tasks/unit time
  - Throughput can exploit parallelism, latency can’t
  - Sometimes complimentary, often contradictory

Example: move people from A to B, 10 miles
- Car: capacity = 5, speed = 60 miles/hour
- Bus: capacity = 60, speed = 20 miles/hour
- Latency: car = 10 min, bus = 30 min
- Throughput: car = 15 PPH (count return trip), bus = 60 PPH

No right answer: pick metric for your goals
Performance Improvement

• Processor A is $X$ times faster than processor B if
  – $\text{Latency}(P,A) = \text{Latency}(P,B) / X$
  – $\text{Throughput}(P,A) = \text{Throughput}(P,B) \times X$

• Processor A is $X\%$ faster than processor B if
  – $\text{Latency}(P,A) = \text{Latency}(P,B) / (1+X/100)$
  – $\text{Throughput}(P,A) = \text{Throughput}(P,B) \times (1+X/100)$

• Car/bus example
  – Latency? Car is 3 times (200\%) faster than bus
  – Throughput? Bus is 4 times (300\%) faster than car
Partial Performance Metrics Pitfalls

• Which processor would you buy?
  – Processor A: CPI = 2, clock = 2.8 GHz
  – Processor B: CPI = 1, clock = 1.8 GHz
  – Probably A, but B is faster (assuming same ISA/compiler)

• Classic example
  – 800 MHz Pentium III faster than 1 GHz Pentium 4
  – Same ISA and compiler
Averaging Performance Numbers (1/2)

• Latency is additive, throughput is not
  \[ \text{Latency}(P1+P2,A) = \text{Latency}(P1,A) + \text{Latency}(P2,A) \]
  \[ \text{Throughput}(P1+P2,A) \neq \text{Throughput}(P1,A)+\text{Throughput}(P2,A) \]

• Example:
  – 180 miles @ 30 miles/hour + 180 miles @ 90 miles/hour
  – 6 hours at 30 miles/hour + 2 hours at 90 miles/hour
    • Total latency is 6 + 2 = 8 hours
    • Total throughput is not 60 miles/hour
      – Total throughput is only 45 miles/hour! (360 miles / (6 + 2 hours))

Arithmetic mean is not always the answer!
Averaging Performance Numbers (2/2)

- **Arithmetic**: times
  - proportional to time
  - e.g., latency

- **Harmonic**: rates
  - inversely proportional to time
  - e.g., throughput

- **Geometric**: ratios
  - unit-less quantities
  - e.g., speedups

Memorize these to avoid looking them up later.
Parallelism: Work and Critical Path

- **Parallelism**: number of independent tasks available
- **Work** (T1): time on sequential system
- **Critical Path** (T∞): time on infinitely-parallel system

- **Average Parallelism**: 
  \[ P_{\text{avg}} = \frac{T_1}{T\infty} \]

- For a \( p \)-wide system:
  \[ T_p \geq \max\{ \frac{T_1}{p}, T\infty \} \]
  \[ P_{\text{avg}} \gg p \implies T_p \approx \frac{T_1}{p} \]

Can trade off frequency for parallelism
Locality Principle

• Recent past is a good indication of near future

**Temporal Locality**: If you looked something up, it is very likely that you will look it up again soon

**Spatial Locality**: If you looked something up, it is very likely you will look up something nearby soon
Power vs. Energy (1/2)

• **Power**: instantaneous rate of energy transfer
  – Expressed in Watts
  – In Architecture, implies conversion of electricity to heat
  – \( \text{Power(Comp1+Comp2)} = \text{Power(Comp1)} + \text{Power(Comp2)} \)

• **Energy**: measure of using power for some time
  – Expressed in Joules
  – \( \text{power} \times \text{time} \) (joules = watts \( \times \) seconds)
  – \( \text{Energy(OP1+OP2)} = \text{Energy(OP1)} + \text{Energy(OP2)} \)
CSE502: Computer Architecture

Power vs. Energy (2/2)

Power is height of curve

Lower power design could simply be slower

Approach 1

Approach 2

time

Energy is area under curve

Total energy needed to complete operation

Approach 1

Approach 2

time

Does this example help or hurt?
Why is energy important?

• Because electricity consumption has costs
  – Impacts battery life for mobile
  – Impacts electricity costs for tethered
    • Delivering power for buildings, countries
    • Gets worse with larger data centers ($7M for 1000 racks)
Why is power important?

• Because power has a peak
• All power “spent” is converted to heat
  – Must dissipate the heat
  – Need heat sinks and fans
• What if fans not fast enough?
  – Chip powers off (if it’s smart enough)
  – Melts otherwise
• Thermal failures even when fans OK
  – 50% server reliability degradation for +10oC
  – 50% decrease in hard disk lifetime for +15oC
Power

• Dynamic power vs. Static power
  – Static: “leakage” power
  – Dynamic: “switching” power

• Static power: steady, constant energy cost
• Dynamic power: transitions from 0→1 and 1→0
Power: The Basics (1/2)

• **Dynamic Power**
  – Related to switching activity of transistors (from $0 \rightarrow 1$ and $1 \rightarrow 0$)

  \[
  \text{Dynamic Power} \propto C V_{dd}^2 A f
  \]
  – $C$: capacitance, function of transistor size and wire length
  – $V_{dd}$: Supply voltage
  – $A$: Activity factor (average fraction of transistors switching)
  – $f$: clock frequency
  – About 50-70\% of processor power
• Static Power
  – Current leaking from a transistor even if doing nothing (steady, constant energy cost)

anskis Power ∝ \( V_{dd} \) and \( e^{-c_1 V_{th}} \) and \( e^{c_2 T} \)
  – This is a first-order model
  – \( c_1, c_2 \): some positive constants
  – \( V_{th} \): Threshold Voltage
  – \( T \): Temperature
  – About 30-50% of processor power
Thermal Runaway

• Leakage is an exponential function of temperature

• ↑ Temp leads to ↑ Leakage

• Which burns more power

• Which leads to ↑ Temp, which leads to...

Positive feedback loop will melt your chip
Why Power Became an Issue? (1/2)

• Ideal scaling was great (aka Dennard scaling)
  – Every new semiconductor generation:
    • Transistor dimension: x 0.7
    • Transistor area: x 0.5
    • C and $V_{dd}$: x 0.7
    • Frequency: $1 / 0.7 = 1.4$
  – Constant dynamic power density
  – In those good old days, leakage was not a big deal

$Dynamic\ Power: CV_{dd}^2 Af$

40% faster and 2x more transistors at same power
Why Power Became an Issue? (2/2)

• Recent reality: $V_{dd}$ does not decrease much
  – Switching speed is roughly proportional to $V_{dd} - V_{th}$
    • If too close to threshold voltage ($V_{th}$) → slow transistor
    • Fast transistor & low $V_{dd}$ → low $V_{th}$ → exponential leakage increase
  → Dynamic power density keeps increasing
  – Leakage power has also become a big deal today
    • Due to lower $V_{th}$, smaller transistors, higher temperatures, etc.

• Example: power consumption in Intel processors
  – Intel 80386 consumed ~ 2 W
  – 3.3 GHz Intel Core i7 consumes ~ 130 W
  – Heat must be dissipated from 1.5 x 1.5 cm$^2$ chip
  – This is the limit of what can be cooled by air

Referred to as the Power Wall
How to Reduce Power? (1/3)

- **Clock gating**
  - Stop switching in unused components
  - Done automatically in most designs
  - Near instantaneous on/off behavior

- **Power gating**
  - Turn off power to unused cores/caches
  - High latency for on/off
    - Saving SW state, flushing dirty cache lines, turning off clock tree
    - Carefully done to avoid voltage spikes or memory bottlenecks
  - Issue: Area & power consumption of power gate
  - Opportunity: use thermal headroom for other cores
How to Reduce Power? (2/3)

• Reduce Voltage (V): quadratic effect on dyn. power
  – Negative (~linear) effect on frequency

• Dynamic Voltage/Frequency Scaling (DVFS): set frequency to the lowest needed
  – Execution time = IC * CPI * f

• Scale back V to lowest for that frequency
  – Lower voltage → slower transistors
  – Dyn. Power ≈ C * V^2 * F

Not Enough! Need Much More!
How to Reduce Power? (3/3)

• Design for E & P efficiency rather than speed

• New architectural designs:
  – Simplify the processor, shallow pipeline, less speculation
  – Efficient support for high concurrency (think GPUs)
  – Augment processing nodes with accelerators
  – New memory architectures and layouts
  – Data transfer minimization
  – ...

• New technologies:
  – Low supply voltage ($V_{dd}$) operation: Near-Threshold Voltage Computing
  – Non-volatile memory (Resistive memory, STTRAM, ...)
  – 3D die stacking
  – Efficient on-chip voltage conversion
  – Photonic interconnects
  – ...
Processor Is Not Alone

SunFire T2000

- Processor: < ¼ System Power
- Memory: 23%
- I/O: 20%
- Disk: 14%
- Services: 10%
- Fans: 9%
- AC/DC Conversion: 4%

Need whole-system approaches to save energy
ISA: A contract between HW and SW

• **ISA**: Instruction Set Architecture
  – A well-defined hardware/software interface

• The “contract” between software and hardware
  – Functional definition of operations supported by hardware
  – Precise description of how to invoke all features

• No guarantees regarding
  – How operations are implemented
  – Which operations are fast and which are slow (and when)
  – Which operations take more energy (and which take less)
Components of an ISA

• Programmer-visible states
  – Program counter, general purpose registers, memory, control registers

• Programmer-visible behaviors
  – What to do, when to do it

Example “register-transfer-level” description of an instruction

if imem[rip]==“add rd, rs, rt” then
  rip ← rip+1
  gpr[rd]=gpr[rs]+gpr[rt]

• A binary encoding

ISAs last forever, don’t add stuff you don’t need
RISC vs. CISC

• Recall Iron Law:
  – (instructions/program) * (cycles/instruction) * (seconds/cycle)

• **CISC** (Complex Instruction Set Computing)
  – Improve “instructions/program” with “complex” instructions
  – Easy for assembly-level programmers, good code density

• **RISC** (Reduced Instruction Set Computing)
  – Improve “cycles/instruction” with many single-cycle instructions
  – Increases “instruction/program”, but hopefully not as much
    • Help from smart compiler
  – Perhaps improve clock cycle time (seconds/cycle)
    • via aggressive implementation allowed by simpler instructions

Today’s x86 chips translate CISC into ~RISC
Prototypical Processor Organization

Addr-gen.  Fetch  Decode  Issue  Execute  Memory

PC +4 Instruction Access

Register File

ALU

Data Access

(Write-back)