CS2410
Computer Architecture

Dept. of Computer Science
University of Pittsburgh


Flynn’s Taxonomy

<table>
<thead>
<tr>
<th>SISD</th>
<th>(SIMD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single instruction stream</td>
<td>Single instruction stream</td>
</tr>
<tr>
<td>Single data stream</td>
<td>Multiple data stream</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MISD</th>
<th>(MIMD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple instruction stream</td>
<td>Multiple instruction stream</td>
</tr>
<tr>
<td>Single data stream</td>
<td>Multiple data stream</td>
</tr>
</tbody>
</table>

*Does not make much sense*

*classic von Neumann*
What is Computer Architecture? (§1.3)

**Models of Parallel executions:**
- Instruction Level parallelism (ILP)
- Data-level parallelism (DLP)
- Thread-level parallelism (TLP)
- Request-level parallelism (RLP)

Trends in Technology(§1.4)

- Integrated circuit technology
  - Transistor density: +35%/year (feature size decreases)
  - Die size: +10-20%/year
  - Integration overall: +40-55%/year

- DRAM capacity: +25-40%/year (growth is slowing)
  (memory usage doubles every year)

- Flash capacity: +50-60%/year
  - 8-10X cheaper/bit than DRAM

- Magnetic disk technology: +40%/year
  - 8-10X cheaper/bit then Flash and 200-300X cheaper/bit than DRAM

- Clock rate stopped increasing
Bandwidth and Latency

Latency lags bandwidth (in the last 30 years)

- Bandwidth or throughput
  - Total work done in a given time
  - 32,000 - 40,000X improvement for processors
  - 400 - 2400X improvement for memory and disks

- Latency or response time
  - Time between start and completion of an event
  - 50 - 90X improvement for processors
  - 8 - 9X improvement for memory and disks

Transistors and wires

- Feature size
  - Minimum size of transistor or wire
  - 10 microns in 1971 to 22 nm in 2012 to 16 nm in 2017 (7 nm is being developed)
  - Transistor performance scales linearly
    - Wire delay per unit length does not improve with feature size!
  - Integration density scales quadratically

Moore’s note (1965)
Switches

• Building block for digital logic
  – NAND, NOR, NOT, ...

• Technology advances have provided designers with switches that are
  – Faster;
  – Lower power;
  – More reliable (e.g., vacuum tube vs. transistor); and
  – Smaller.

• Nano-scale technologies will not continue promising the same good properties

History of switches

Called "relay"; Mark I (1944)

Vacuum tubes; ENIAC (1946, 18k tubes)

Bell lab. (1947); Kilby’s first IC (1957)

Solid-state MOS devices
### Power and Energy (§1.5)

- Need to get power in and out (thermal implications)
- Dynamic energy (to switch transistors)
  - Energy is proportional to $V^2$
  - Power is proportional to $(V^2 \times \text{Frequency})$
- Dynamic Frequency Scaling (reduces power not energy) $\rightarrow$ voltage scaling
- Static power is proportional to the voltage
- Memory power modes and turning off cores

### Computer Engineering Methodology

Technology trends

Evaluate Existing Systems for Bottlenecks

Simulate New Designs and Organizations

Implement Next Generation System
Integrated Circuits Costs (§1.6)

\[
\text{Dies per Wafer} = \frac{\pi}{\text{Die Area}} \left( \frac{(\text{Wafer diam}/2)^2}{\text{Die Area}} \right) - \frac{\pi}{\sqrt{2}} \frac{\text{Wafer diam}}{\text{Die area}}
\]

Die yield = wafer yield * \((1 + \text{Defect per unit area} \times \text{Die area})^N\)

Where \(N = \) process complexity factor = 7.5 – 9.5 (28nm, 2017)

\[
\begin{align*}
\text{Die Cost} & = \frac{\text{Wafer Cost}}{\text{Dies per Wafer} \times \text{Die Yield}} \\
\text{IC Cost} & = \frac{\text{Die Cost} + \text{Testing Cost} + \text{Packaging Cost}}{\text{Final Test Yield}}
\end{align*}
\]

Dependability (§1.7)

- Fault: failure of a component
- Error: manifestation of a fault
- Faults may or may not lead to system failure

- Reliability measure: mean time to failure (MTTF)
- Repair efficiency: mean time to repair (MTTR)
- Mean time between failures
  \(\text{MTBF} = \text{MTTF} + \text{MTTR}\)
- Availability = MTTF / MTBF
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair
Performance (§1.8)

Measuring Performance

- **Time to run the task (latency)**
  - Execution time, response time, CPU time, …
- **Tasks per day, hour, week, sec, ns, …**
  - Throughput, bandwidth

Performance measurement Tools

- Hardware prototypes: Cost, delay, area, power estimation
- Simulation (many levels, ISA, RT, Gate, Circuit, …)
- Benchmarks (Kernels, toy programs, synthetic), Traces, Mixes
- Analytical modeling and Queuing Theory
Benchmarks

• SPEC2017: Standard Performance Evaluation Corporation
• PARSEC: Princeton Application Repository for Shared-Memory Computers
• MediaBench: Multimedia and embedded applications
• Transaction processing- TPC-C, SPECjbb
• Embedded Microprocessor Benchmark Consortium – EEMBC: Networking, telecom, digital cameras, cellular phones, ...
• Stanford parallel benchmarks: For parallel architecture and shared memory multiprocessors
• NAS: For massively parallel processor systems
• Rodinia: for GPU applications

How to Summarize Performance

• Arithmetic mean (weighted arithmetic mean)
  – ex: tracks execution time: \( \frac{\sum_{i=1}^{n} W_i \cdot T_i}{\sum_{i=1}^{n} W_i} \) or \( \sum_{i=1}^{n} \frac{T_i}{n} \)

• Harmonic mean (weighted harmonic mean) of rates
  – ex: track MFLOPS: \( \frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}} \)

• Normalized execution time is handy for scaling performance (e.g., X times faster than Pentium 4)

• Geometric mean \( \Rightarrow \left( \prod_{i=1}^{n} \text{execution ratio}_{i} \right)^{\frac{1}{n}} \)
  where the execution ratio is relative to a reference machine
Performance Evaluation

- Good products created when we have:
  - Good benchmarks
  - Good ways to summarize performance

- For better or worse, benchmarks shape a field.

- Given that sales is a function, in part, of performance relative to competition, companies invest in improving performance summary

- If benchmarks/summary are inadequate, then choose between improving product for real programs vs. improving product to get more sales \( \implies \) Sales almost always wins!

- Reproducibility is important (should provide details of experiments)

Execution time and power are the main measure of computer performance!

---

Amdahl's Law (§ 1.9)

Speedup due to some enhancement \( E \):

\[
\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{without } E}}{\text{ExTime}_{\text{with } E}} = \frac{\text{Performance}_{\text{with } E}}{\text{Performance}_{\text{without } E}}
\]

Suppose that enhancement \( E \) accelerates a fraction of the task by a factor \( S \), and the remainder of the task is unaffected

\[
\frac{\text{ExTime}_{\text{with } E}}{\text{ExTime}_{\text{without } E}} = \left(1 - \text{fraction}_{\text{enhanced}}\right) + \frac{\text{fraction}_{\text{enhanced}}}{S}
\]

Example: Floating point instructions can be improved to run 2\( X \); but only 10\% of actual instructions are FP. What is the overall speedup?
Computing CPU time

Average Cycles per Instruction (CPI) = \( \sum_{j=1}^{n} CPI_j \times F_j \)

Where \( CPI_j \) is the number of cycles needed to execute instructions of type \( j \), and \( F_j \) is the percentage (fraction) of instructions that are of type \( j \).

**Example**: Base Machine (Reg / Reg)

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI_j * F_j (%) Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5 (33%)</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>.4 (27%)</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
<td>.2 (13%)</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4 (27%)</td>
</tr>
</tbody>
</table>

Typical Mix

\[ \sum_{j=1}^{n} CPI_j \times F_j = 1.5 \]

Instructions Per Cycle (IPC) = \( 1 / CPI \)

---

\[ \text{CPU time} = \text{Cycle time} * \sum_{j=1}^{n} CPI_j \times I_j \]

Where \( I_j \) is the number of instructions of type \( j \), and

*Cycle time* is the inverse of the *clock rate*.

**Example**: For some programs,

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0
Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

What machine is faster for this program, and by how much?
### Aspects of CPU Performance

$$CPU\_time = \frac{Seconds\_program}{Instructions\_program} \times \frac{Cycles}{Instructions} \times \frac{Seconds\_Cycle}{Cycle}$$

<table>
<thead>
<tr>
<th></th>
<th>Inst Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Inst. Set.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

### Improving CPI using caches

**An example:**

- CPU: 10 n.sec
- Cache: 70% reduction in access time
- Memory: 100 n.sec

What is the improvement (speedup) in memory access time?

**Caching works because of the principle of locality:**

- Locality found in memory access instructions
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon
- 90/10 locality rule
  - A program executes about 90% of its instructions in 10% of its code
- We will look at how this principle is exploited in various microarchitecture techniques