EE 4683/5683: COMPUTER ARCHITECTURE

Lecture 2: Performance Metrics

Avinash Kodi, kodi@ohio.edu

Performance Metrics

- Cost of Integrated Circuits
- Power (dynamic and static)
- Dependability, Availability
- Fundamental CPU Equation
- Amdahl's Law
- What computer architects bring to the table

Classes of Computers

Metrics of Efficiency

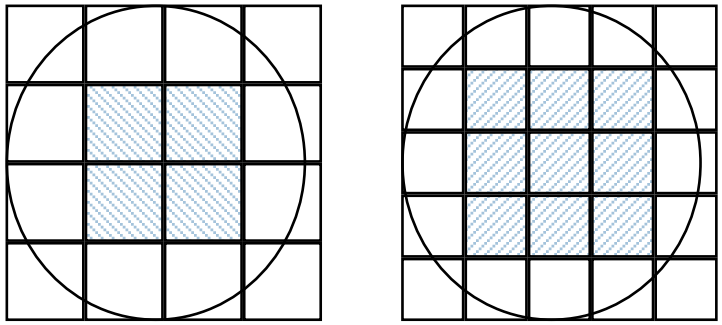
- Personal Mobile Device ($100 - $1000)
  - Metrics?
  - Example – ARM processors
- Desktop Computing ($300 - $2500)
  - Metrics: ?
  - Example – Intel Sandy Bridge, AMD Athlon, IBM PowerPC
- Server Computing ($5K - $10M)
  - Metrics: ?
  - Example – IBMPower5, Sun UltraSparc, AMD Opteron, Intel Xeon
- Warehouse-scale Computing ($100K - $200M)
  - Metrics?
  - Example – Google, amazon datacenters
- Embedded Computing ($10 - $100K)
  - Metrics: ?
  - Example – ARM, MIPS, Motorola 68K

Integrated Circuits Costs

$$\text{IC Cost} = \frac{\text{Die Cost} + \text{Testing Cost} + \text{Packaging Cost}}{\text{Final Test Yield}}$$

Yield : % of manufactured devices that survives the testing procedures

$$\text{Die Cost} = \frac{\text{Wafer Cost}}{\text{Dies Per Wafer} \times \text{Die Yield}}$$



Integrated Circuits Costs

$$\text{Die Cost} = \frac{\text{Wafer Cost}}{\text{Dies Per Wafer} \times \text{Die Yield}}$$

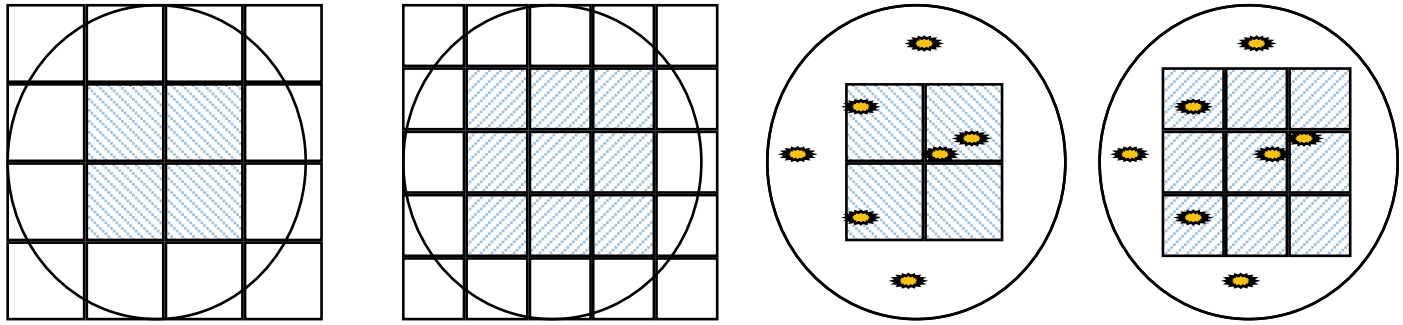
$$\text{Dies per Wafer} = \frac{\pi \times (\text{Wafer Diameter}/2)^2}{\text{Die Area}} - \frac{\pi \times \text{Wafer Diameter}}{\sqrt{2 \times \text{Die Area}}}$$

Example: Find the number of dies per 20 cm wafer diameter for a die that is 1.5 cm on the side ?

$$\text{Dies per Wafer} = \frac{\pi \times (20/2)^2}{2.25} - \frac{\pi \times 20}{\sqrt{2 \times 2.25}} = \frac{314}{2.25} = \frac{62.8}{2.12} = 110$$

Integrated Circuits Cost

- Previous calculation gives the maximum number of dies per wafer
- However, we want to know the percentage of good dies on the wafer



Integrated Circuits Costs

$$\text{Die Yield} = \text{Wafer Yield} \times \left[\left(\frac{1 + \text{Defect Density} \times \text{Die Area}}{\alpha} \right) \right]^{-\alpha}$$

- Defect density (defects per unit area): measure of random and manufacturing defects, in 2010 was 0.016-0.057 defects per cm^2 for 40 nm process technology
- α is a parameter that corresponds to the number of masking levels, a good estimate is 4.0
- Die cost scales roughly with die $area^2$

Other Cost Contributors

- Testing cost
  - Cost/die = (cost/hr x test time) / yield
  - Could be $10-$20 or more for complex chips
- IC Packaging
  - Depends on die size, number of pins, power dissipation
- Cost of cooling system
- And most important is the VOLUME,
  - Cost of the modern IC fabrication facility >$2B
  - Need volume to amortize all this cost

Quantify Power (1/4)

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called dynamic power

$$\text{Power}_{\text{Dynamic}} = 0.5 \times \text{Capacitive Load} \times \text{Voltage}^2 \times \text{Frequency}$$

- For mobile devices, energy better metric

$$\text{Energy}_{\text{Dynamic}} = \text{Capacitive Load} \times \text{Voltage}^2$$

- For fixed task, slowing clock rate (frequency switching) reduces power, but not energy

Quantify Power (2/4)

- Capacitive load is a function of the number of transistors connected to the output and technology (determines capacitance of wires and transistors)
- Dropping voltage helps both, so went from 5 V to 1 V and below
- To save energy and dynamic power, most CPUs now turn off the clock of inactive modules (Eg: Floating Point Unit)
  - Apply DVFS (dynamic voltage and frequency scaling)

Quantify Power (3/4)

- Suppose 15% reduction in voltage results in a 15% reduction in frequency, what is the impact on dynamic power?

$$\begin{aligned}
\text{Power}_{\text{dynamic}} &= 1/2 \times \text{Capacitive Load} \times \text{Voltage}^2 \times \text{Frequency Switched} \\
&= 1/2 \times \text{Capacitive Load} \times (.85 \times \text{Voltage})^2 \times .85 \times \text{Frequency Switched} \\
&= (.85)^3 \times OldPower_{dynamic} \\
&\approx 0.6 \times OldPower_{dynamic}
\end{aligned}$$

Quantify Power (4/4)

- As leakage current flows even when the transistor is off, now static power is important

$$Power_{static} = Current_{static} \times Voltage$$

- Leakage power increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even when they are turned off
- In 2011, goal for leakage is to be 25% and for high performance designs can be as high as 50% of total power

Dependability (1/3)

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:
  1. Service accomplishment, where the service is delivered as specified in SLA
  2. Service interruption, where the delivered service is different from the SLA
- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

Dependability (2/3)

- Module reliability = measure of continuous service accomplishment (or time to failure)
  1. *Mean Time To Failure* (*MTTF*) measures Reliability
  2. *Failures In Time* (*FIT*) = 1/MTTF, the rate of failures
     - Traditionally reported as failures per billion hours of operation
- *Mean Time To Repair* (*MTTR*) measures Service Interruption
  - *Mean Time Between Failures* (*MTBF*) = MTTF+MTTR
- *Module availability* measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Dependability (3/3)

- If modules have exponential distributed lifetimes (age of module does not affect probability of failure), overall failure rate is the sum of failure rate of the modules
- Calculate FIT, and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF) and 1 power supply (0.2M hour MTTF)

$$
\begin{aligned}
FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\
&= 10 + 2 + 5/1{,}000{,}000 \\
&= 17/1{,}000{,}000 \\
&= 0.000017 \\
MTTF &= 1{,}000{,}000/17 \\
&\approx 59{,}000 hours
\end{aligned}
$$

Computer Performance: TIME

- Response Time (Latency)
  - How long does it take for my job to run ?
  - How long does it take to execute a job ?
  - How long should I wait for the database query?
- Throughput
  - How many jobs can run at once on the machine ?
  - What is the average execution time ?
  - How much work is getting done ?

Defining Performance

- Reducing response time
  - To maximize performance, need to minimize response time

$$\text{Performance (x)} = \frac{1}{\text{Execution Time (x)}}$$

- If performance of X is greater than Y by a factor of "n"

$$\frac{\text{Performance (X)}}{\text{Performance (Y)}} = \frac{\text{Execution Time (Y)}}{\text{Execution Time (X)}} = n$$

- Decreasing the response time mostly increases the throughput

Performance Factors

- Another way of reporting the execution time is to use cycles

CPU execution time for a program = # CPU clock cycles for a program x clock cycle time

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Clock “ticks” indicate when to start an activity
- Cycle time = time between ticks = seconds per cycle
- Clock rate (frequency) = cycles per second

Cycles Per Instruction (CPI)

- A given program will require
  - Some number of instructions (machine instructions)
  - Some number of cycles
  - Some number of seconds to execute

CPU clock cycles for a program = # Instructions for a program x Average clock cycles per instruction

- Cycles per Instruction (CPI) : the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

Effective CPI

- Computing the overall effective CPI is done by summing the different instructions and their individual clock cycles

$$\text{Overall Effective CPI} = \sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i$$

  - Where IC_i is the count (percentage) of the number of instructions of class i executed
  - CPI_i is the (average) number of clock cycles per instruction for that instruction class
  - n is the number of instruction classes

- The overall effective CPI varies by instruction mix – a measure of dynamic frequency of instructions across many programs

CPI Example

	CPI for this instruction class		
	A	B	C
CPI	1	2	3

Code Sequence	Instruction count for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

- Which code sequence executes the most instructions? Which will be faster?

Total CPU Cycles$_1$ = 2 x 1 + 1 x 2 + 3 x 2 = 10 cycles
Total CPU Cycles$_2$ = 4 x 1 + 1 x 2 + 1 x 3 = 9 cycles

Performance Equation

- Our basic performance equation becomes

CPU Execution Time = Instruction Count x CPI x Clock Cycle

$$\text{CPU Execution Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- These factors separate three factors that affect performance
  - Can measure CPU execution time (run the program)
  - Clock rate is given
  - Can measure the overall instruction count using profilers/ simulators
  - CPI varies by the instruction type and implementation details

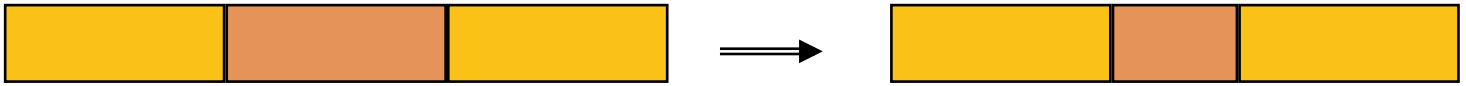
Amdahl's Law

- This law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
- The law defines the performance factor as the speedup: it tells us how much faster a task will run using the machine with the enhancement as opposed to the original machine.
- It gives us the overall performance behavior of a computer with two different modes and speeds of operation.
- A serial machine (one CPU) vs a parallel versions, multiple CPUs, etc.

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



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

Amdahl's Law

- It depends on two factors: $Fraction_{enhanced}$ and $Speedup_{enhanced}$
- $Fraction_{enhanced}$: fraction of the computation time in the original program that can be converted to take advantage of enhancement.
- $Speedup_{enhanced}$ the amount of improvement.

$$ExTime_{new} = ExTime_{old} \times \left[(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Amdahl's Law: Example 1

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$ExTime_{new}$ =

$Speedup_{overall}$ =

Amdahl's Law: Example 1

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times (0.9 + .1/2) = 0.95 \times \text{ExTime}_{old}$$

$$\text{Speedup}_{overall} = \frac{1}{0.95} = 1.053$$

Amdahl's Law: Example 2

- An enhancement that runs 10 times faster than the original machine but only for 40% of the time:

$Fraction_{enhanced} = 0.4$

$Speedup_{enhanced} = 10$

$Speedup_{overall} = 1/ (0.6 + 0.4/10) = 1/0.64 = 1.65$

Amdahl's law is applicable for parallel processors

Amdahl's Law summary

- If an enhancement is only suitable for a fraction of a task, we can't speed up the task by more than the reciprocal of 1 minus the fraction
- This of course assumes fixed-load problems: the problem size does not change with improvements.

Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications, called *benchmark suites*, are popular
- SPECCPU: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPECint2000 has 12 integer, SPECfp2000 has 14 integer programs
  - SPECCPU2006 has both integer and floating point
  - SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- Transaction Processing Council measures server performance and cost-performance for databases
  - TPC-C Complex query for Online Transaction Processing
  - TPC-H models ad hoc decision support
  - TPC-W a transactional web benchmark
  - TPC-App application server and web services benchmark

How to Summarize Performance (1/3)

- Arithmetic average of execution time of all programs?
  - But they vary by 4X in speed, so some would be more important than others in arithmetic average
- Could add a weights per program, but how pick weight?
  - Different companies want different weights for their products
- SPECRatio: Normalize execution times to reference computer, yielding a ratio proportional to performance = time on reference computer/time on computer being rated

How to Summarize Performance (2/3)

- If program SPECRatio on Computer A is 1.25 times bigger than Computer B, then

$$1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\frac{ExecutionTime_{reference}}{ExecutionTime_A}}{\frac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

$$= \frac{ExecutionTime_B}{ExecutionTime_A} = \frac{Performance_A}{Performance_B}$$

- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant

How to Summarize Performance (3/3)

- Since ratios, proper mean is geometric mean (SPECRatio unitless, so arithmetic mean meaningless)

$$\text{Geometric Mean} = \sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio}_i}$$

1. Geometric mean of the ratios is the same as the ratio of the geometric means
2. Ratio of geometric means
   = Geometric mean of performance ratios
   $\Rightarrow$ choice of reference computer is irrelevant!

- These two points make geometric mean of ratios attractive to summarize performance

What Computer Architecture brings to Table

- Quantitative Principles of Design
  1. Take Advantage of Parallelism
  2. Principle of Locality
  3. Focus on the Common Case
  4. Amdahl's Law
  5. The Processor Performance Equation
- Careful, quantitative comparisons
  - Define, quantity, and summarize relative performance
  - Define and quantity relative cost
  - Define and quantity dependability
  - Define and quantity power
- Culture of anticipating and exploiting advances in technology
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked