CSE340: Computer Architecture

Handout_Chapter - 1: Computer Abstractions and Technology

Inspiring Excellence

Prepared by: Partha Bhoumik (PBK)
Course Coordinator, CSE340

Understanding Performance of a Program

The performance of a program depends on four main factors:

1. **Algorithm**: The algorithm here simply refers to the **original program** itself. The more optimized the algorithm the better the performance.
2. **Programming Language, Compiler and Architecture**: The software tools used to write and convert the program into machine instructions. This determines the number of machine instructions to be executed for the program written.
3. **Processor and memory system:** Determine how fast instructions can be executed.
4. **I/O system:** How efficiently the system can handle the input and output operations.

The first two factors are software-based, while the last two depend on the hardware.

Seven Great Ideas in Computer Architecture

Use ***abstraction*** to simplify design:

As the systems became more complex computer architects and programmers needed to find ways to work more efficiently, else designing computers and programs would take much longer.

Abstraction is the key technique used to solve this problem by simplifying complex systems into multiple layers, allowing designers to focus on higher-level tasks while hiding the details of lower levels.

Say you want to store the value 5 in a variable called x. You simply write:
int x = 5; // in ‘C’ programming language.

In reality, the value 5 is stored in a memory location whose exact details you, as a programmer, do not need to worry about. This is where abstraction comes in, hiding the underlying hardware's complexity.

Also at the hardware level, the value 5 is represented as high and low voltages in transistors, which physically store the binary value 00000101. The abstraction allows you to simply use int x = 5; in your code, without having to know how these details are managed by the hardware.

Make the ***common case fast***:

Fastening the tasks you do most **commonly** has a bigger impact on performance than the rare ones. Common tasks are usually easier to improve, but you need careful experimentation and measurement to know which tasks happen most commonly.

Performance via ***parallelism***:

Computer architects have designed systems to boost performance by performing multiple operations at the same time (**parallel computing**).

For example, in multi-core processors, each core can work on a different task or part of a task simultaneously, significantly speeding up performance compared to a single-core processor.

Performance via ***pipelining***:

A way to achieve parallelism. More will be discussed in Chapter - 4

Performance via ***prediction***:

It can be faster on average to make a prediction and start working rather than waiting until you are completely sure, as long as the process to recover from a misprediction is not too expensive and your prediction is relatively accurate.

For instance,

```
if (x > 10) {
   // Do something
} else {
   // Do something else
}
```

At this moment the processor encounters the if statement, it does not yet know the value of x, so it does not know which path to follow. Instead of waiting for the condition to be evaluated and slowing down the execution, modern processors use **branch prediction** to guess which path is more likely to be taken.

Here is how it works:

Prediction: The processor looks at the history of past if statements or similar conditions. If, for example, it has seen that x > 10 is often true in the past, it guesses that the "Do something" part will be executed.

Execution: The processor starts executing the "Do something" path immediately, without waiting to know for sure.

Correction: If the processor guessed wrong and x <= 10 turns out to be true, it has to undo the work done on the wrong path and start executing the "Do something else" part. This is called a misprediction penalty.

The main idea is if the misprediction penalty is not too high, and the prediction is usually accurate, then it is faster overall to make a guess and continue working instead of waiting to know the condition for sure. This can result in faster overall program execution.

Hierarchy of memories:

Programmers want memory to be:

- Fast (affects performance)
- Large (limits the size of problems)
- Cheap (memory cost is a major factor)

To balance these needs, architects use a memory hierarchy:

- Top: Fast, small, and expensive memory (ex: cache)
- Bottom: Slow, large, and cheap memory (ex: hard drives)

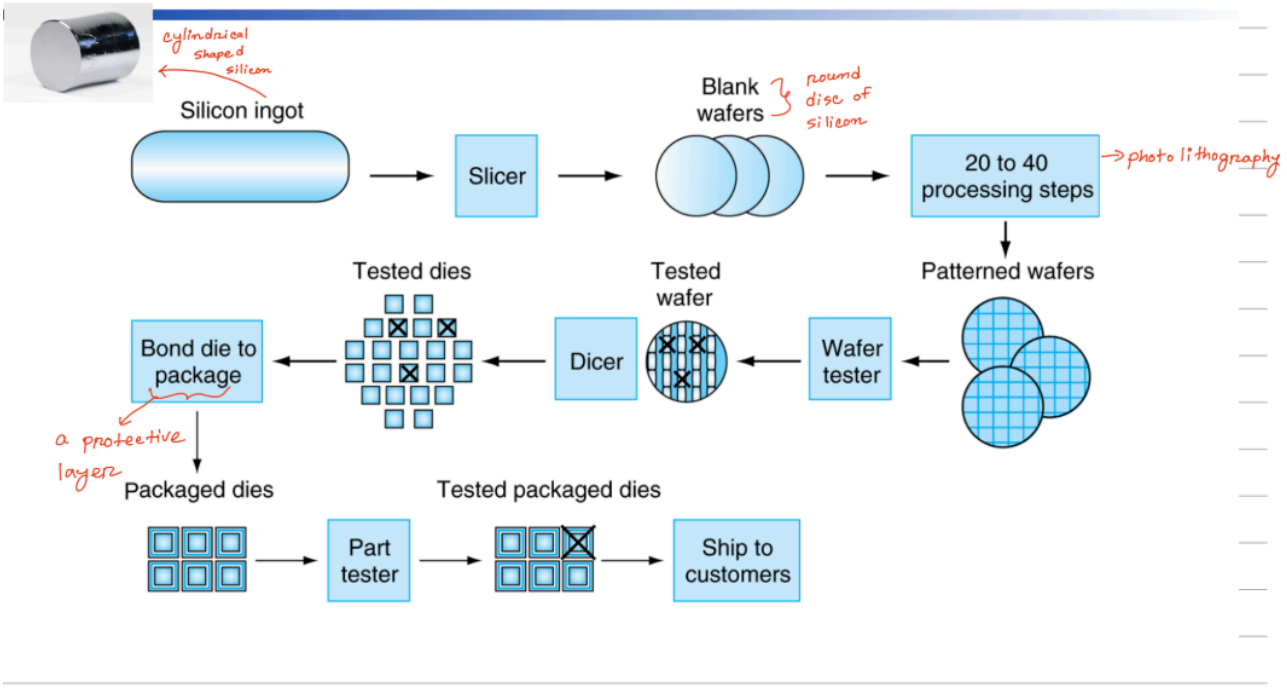
The hierarchy helps manage cost by using expensive memory for frequently accessed data and cheaper memory for less used data, making the overall system more efficient and affordable.
More will be discussed in Chapter - 5.

***Dependability** via* redundancy:

Computers not only need to be fast; they need to be dependable(reliable). Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures.

Example: In a RAID setup, data is duplicated across multiple hard drives so that if one drive fails, the system can continue working using the data from the other drives. For example, in **RAID 1**, data is mirrored between two drives. If one drive crashes, the other still has an exact copy of the data, ensuring the system remains reliable and available even in the event of hardware failure.

Technologies for Building Processors



Formulas for IC Cost Calculation

1. $Cost\ per\ die = \frac{Cost\ per\ wafer}{Dies\ per\ wafer \times yield}$
2. $Dies\ per\ wafer \approx \frac{Wafer\ area}{Die\ area}$
3. $Die\ area = Height \times Width$
4. $Wafer\ area = \pi \times r^2$
5. $Working\ dies\ per\ wafer = Dies\ per\ wafer \times Yield$
6. $Yield = \frac{1}{(1+(Defects\ per\ area \times Die\ area))^N}$ **Note**: N will be given.

Defining Performance:

Based on the table below which airplane has the best performance?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 737	240	3000	564	135,360
BAC/Sud Concorde	132	4000	1350	178,200
Boeing 777-200LR	301	9395	554	166,761
Airbus A380-800	853	8477	587	500,711

Figure: The capacity, range, and speed for a number of commercial airplanes

In terms of,
passenger capacity => Airbus A380-800
Cruising range => Boeing 777-200LR
Cruising Speed => BAC
Passenger throughput => Airbus A380-800

Keynote: Whenever we discuss performance, we must specify the feature or metric on which the performance is being measured.

Similarly, Computer performance can be defined in different ways depending on the context:

- For personal desktop computers, **response time** (the time it takes to complete a task) is a key metric.
- For data centers, **throughput** (the total amount of work done in a given time) is more important.

Thus, the performance metric (response time or throughput) must be clearly defined based on the specific context or system being discussed.

Replacing the processor in a computer with a faster version - both response time and throughput increases.

Adding additional processors to a system that uses multiple processors for separate tasks - only increases the throughput.

Understanding Different Terms for Performance Calculation

CPU time/ CPU execution time:

The amount of time the CPU spends actively processing a task, excluding time spent on other activities like waiting for input or I/O operations.

User CPU time:

User CPU time is time spent on the processor running your program's code
Example: A program spends 10 seconds executing a loop in its own code.

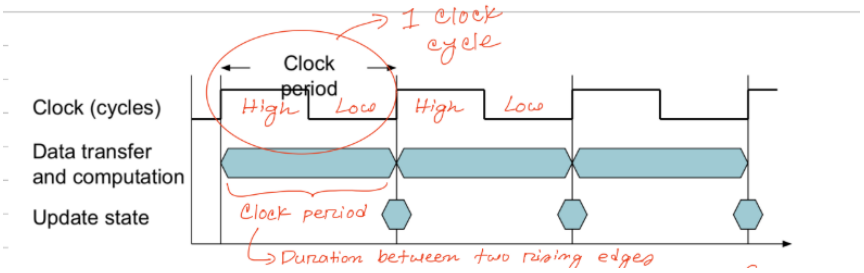
System CPU time:

Amount of time the processor worked on the operating system's functions connected to that specific program.
Example: A program spends 2 seconds waiting for data to be loaded from disk by the operating system.

Response time/ ***Execution*** time/ ***Elapsed*** time :

The total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead—everything

Clock cycle:



Clock Period / Clock Duration/ Clock Cycle Time:

Time taken to complete a clock cycle.

Clock Frequency/ Clock Rate:

The number of Clock cycles completed in one second.

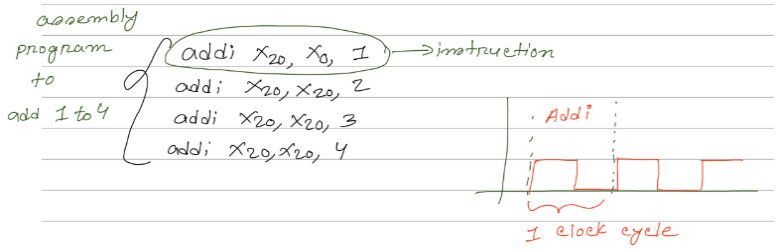
CPI:

The number of clock cycles needed to execute one instruction.

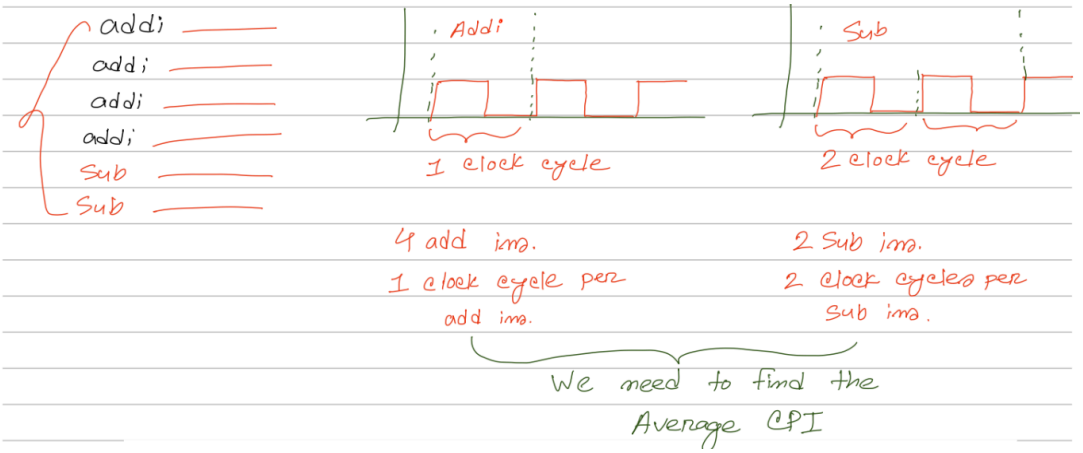
ISA (Instruction Set Architecture):

ISA refers to the set of instructions that a computer processor can understand and execute. These instructions define the operations that the processor can perform, such as arithmetic, logical operations, and data movement. The ISA acts as an interface between the hardware and software, allowing programmers to write code that can be executed by the processor.

Instruction:



When do you need the average CPI?



Reducing Power:

Microarchitecture Enhancement:

- Cache optimization – increase the size & efficiency of cache
- Specialized execution units – GPUs for specific task.

Software optimization :

Energy efficient Computing:

- power gating – turn off the unused component of CPU

Heterogeneous Computing:

- Combine different types of cores. (high perf. + energy eff.)

Advanced cooling and packaging.

Benchmarking

SPEC CPU Benchmark:

SPEC stands for Standard Performance Evaluation Corporation.

SPEC CPU Benchmark is a standardized set of tests used to measure and compare the performance of computer processors.

$$SPEC\ Ratio\ =\ \frac{Reference\ Time}{CPU\ Time}$$

Here, For a particular program/task,
Reference Time means the time it takes to run that program/task on a standard reference computer.

CPU Time means the time it takes to run that program/task on your computer.

Geometric Mean:

$$Geometric\ mean,\ x\ =\ \sqrt[n]{\prod_{i=1}^{n} Spec\ Ratio_i} = \sqrt[n]{S.R_1 \times\ S.R_2 \times\ S.R_3 \times\ \ldots\ldots\ \times S.R_n}$$

The value of the geometric mean indicates that, on average, the CPU is about x times faster than the reference system across the tested workloads/tasks/programs.

Amdahl's Law:

This law helps us to understand the overall performance improvement gained by optimizing a single part of a system.

$$Time_{improved}\ =\ \frac{Time_{affected}}{Improvement\ Factor} + Time_{unaffected}$$

MIPS:

MIPS stands for Millions of instructions per second.
1 MIPS = 1 million instructions executed per second.

Formulas

1. $Performance = \frac{1}{Execution\ Time}$

2. $CPU\ Time = CPU\ clock\ cycles \times Clock\ Cycle\ Time$

3. $Clock\ Cycle\ Time = \frac{1}{Clock\ Rate}$

4. $CPU\ Time = \frac{CPU\ clock\ cycles}{Clock\ Rate}$

5. $CPU\ Clock\ Cycles = Instruction\ Count \times Average\ CPI$

$$= \sum_{i=1}^{n} (Instruction\ Count_i \times CPI_i)$$

6. $CPU\ Time = Instruction\ Count \times Average\ CPI \times Clock\ Cycle\ Time$

7. $CPU\ Time = \frac{Instruction\ Count \times Average\ CPI}{Clock\ Rate}$

8. $Average\ CPI = \frac{\Sigma(CPI\ of\ each\ instruction\ type \times Number\ of\ instructions\ of\ that\ type)}{Total\ number\ of\ instructions}$

9. $SPEC\ Ratio = \frac{Reference\ Time}{CPU\ Time}$

10. $Geometric\ mean = \sqrt[n]{\prod_{i=1}^{n} Spec\ Ratio_i}$

$$= \sqrt[n]{S.R_1 \times S.R_2 \times S.R_3 \times \ldots\ldots \times S.R_n}$$

11. $Time_{improved} = \frac{Time_{affected}}{Improvement\ Factor} + Time_{unaffected}$ [Amdahl's Law]

12. $Power = Capacitive\ Load \times Voltage^2 \times Frequency$

13. 1 MIPS = 1 Million instructions executed in 1 second.

Sample Maths with Explanation

Problem Statement: If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

Solution:

We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.

In the above example, we could also say that Computer B is 1.5 times slower than Computer A.

Problem Statement: Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

Solution:

We know that each computer executes the same number of instructions for the program; let's call this number I.

First, find the number of processor clock cycles for each computer:

$$CPU\ Clock\ Cycles_A = I \times 2.0$$

$$CPU\ Clock\ Cycles_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$CPU\ Time_A = CPU\ Clock\ Cycles_A \times Clock\ Cycle\ Time$$
$$= I \times 2.0 \times 250\,ps = I \times 500\,ps$$
$$CPU\ Time_B = CPU\ Clock\ Cycles_B \times Clock\ Cycle\ Time$$
$$= I \times 1.2 \times 500\,ps = I \times 600\,ps$$

Clearly, computer A is faster.

The amount faster is given by the ratio of the execution times:

$$\frac{CPU\ Performance_A}{CPU\ Performance_B} = \frac{Execution\ Time_B}{Execution\ Time_A} = \frac{I \times 600\,ps}{I \times 500\,ps} = 1.2$$

Computer A is 1.2 times as fast as computer B for this program.

Problem Statement: A compiler designer is trying to decide between two code sequences for a computer. The hardware designers have supplied the following facts:

CPI	Add	Sub	Mul
	1	2	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code Sequence	Instruction counts for each instruction class		
	Add	Sub	Mul
1st	2	1	2
2nd	4	1	1

a) Which code sequence executes the most instructions?
b) Which will be faster?
c) What is the CPI for each sequence?

Solution:

a) Sequence 1 executes 2 + 1 + 2 = 5 instructions.
Sequence 2 executes 4 + 1 + 1 = 6 instructions.

So, sequence 1 executes fewer instructions.

b) The instruction sequence that takes a lesser amount of time to execute will be faster.

CPU Time = Clock cycles X Clock cycle time
= Instruction count X CPI X Clock cycle time

Here we do not know the clock cycle time. But both of these instruction sequences will be running on the same computer. Hence, the clock cycle time will be the same for both of them.

So, we can ignore clock cycle time here and find only the CPU Clock Cycles.

$$CPU\ Clock\ Cycles = \sum_{i=1}^{n} (Instruction\ Count_i \times CPI_i)$$

$$CPU\ Clock\ Cycles_{1st} = (2 \times 1) + (1 \times 2) + (2 \times 3) = 10\ Cycles$$

$$CPU\ Clock\ Cycles_{2nd} = (4 \times 1) + (1 \times 2) + (1 \times 3) = 9\ Cycles$$

So, code sequence 2 is faster

c) $$Average\ CPI = \frac{\Sigma(CPI\ of\ each\ instruction\ type \times Number\ ojf\ instructions\ of\ that\ type)}{Total\ number\ of\ instructions}$$

$$CPI_{1st} = \frac{CPU\ Clock\ Cycles_{1st}}{Total\ number\ of\ instructions_{1st}} = \frac{10}{5} = 2.0$$

$$CPI_{2nd} = \frac{CPU\ Clock\ Cycles_{2nd}}{Total\ number\ of\ instructions_{2nd}} = \frac{9}{6} = 1.5$$