

Performance and Top Level View

Joannah Nanjekye

July 04, 2024

Reading Assignment

- ▶ Summarize the major events before the ENIAC machine
- ▶ Summarize the key highlights in the evolution of the Motorola 68000
- ▶ Embedded System organization and their operating systems

Performance Improvements

- ▶ Speedup the processor hardware by reducing the logic gate size
- ▶ Increase the size and speed of caches
- ▶ Modify the processor organization and architecture e.g by parallelizing some parts

Performance Mechanisms

- ▶ **Pipelining:** processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously
- ▶ **Branch prediction:** processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next
- ▶ **Superscalar execution:** This is the ability to issue more than one instruction in every processor clock cycle. In effect, multiple parallel pipelines are used

Performance Mechanisms

- ▶ **Data flow analysis:** the processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
- ▶ **Speculative execution:** using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations

Performance

- ▶ Concerns from the Architectural perspective:
 - ▶ System performance
 - ▶ Relative importance
 - ▶ Cost
 - ▶ Cost/Performance trade-off
- ▶ Performance comparison:
 - ▶ Criteria
 - ▶ Metric

Question: *What factors in the architecture affect system performance and the cost of these factors?*

Latency and Throughput

- ▶ **Latency is execution time**
 - ▶ How long it takes to finish a task
 - ▶ Computed by tracking the start and end time
- ▶ Throughput is the number of instructions per unit time
 - ▶ The unit should be defined e.g. hour, second etc
- ▶ Depending on the application latency and throughput can be improved in different ways

What is Performance

$$\text{Performance } (P) = 1 / \text{Latency } (L)$$

How to compare two programs:

- ▶ If X takes 50s and Y takes 25s
- ▶ Performance = $L_X / L_Y = 50/25 = 2$

Therefore Y is $2 \times$ *faster than X*

Performance Metrics

- ▶ CPU Time
 - ▶ Time taken to finish a task without regard to I/O time and task sharing
 - ▶ Includes time spent in the program as well as OS tasks
- ▶ CPU performance is different from system performance since the later accounts for the total response time inclusive of all aspects
- ▶ The rate of pulses is called *clock speed* or *clock rate*
- ▶ One pulse is a *clock cycle* or *clock tick*

The Performance Equation

$$\text{CPU Time} = \text{Clock Cycles} \times \text{Clock Cycle Time}$$

$$\text{CPU Time} = \frac{\text{Clock Cycles}}{\text{Clock Rate}}$$

Clock cycles per instruction (CPI): the average number of clock cycles each instruction takes to execute. It is used to compare different implementations of the same ISA

$$\text{Clock Cycles} = \text{instruction_count} \times \text{CPI}$$

$$\text{CPU Time} = \text{instruction_count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$\text{CPU Time} = \frac{\text{instruction_count} \times \text{CPI}}{\text{Clock Rate}}$$

Example

Consider three processors L , M , and N with clock rates of 5GHZ, 3GHZ and 8GHZ respectively. And CPI values 1.5, 2.8, and 2.0. Calculate the number of cycles if execution time is 20s.

$$\text{CPU Time} = \frac{\text{Clock Cycles}}{\text{Clock Rate}}$$

$$\text{Cycles for } L = 20 \times 5 \times 10^9$$

$$\text{Cycles for } M = 20 \times 3 \times 10^9$$

$$\text{Cycles for } N = 20 \times 8 \times 10^9$$

Example

Consider three processors L , M , and N with clock rates of 5GHZ, 3GHZ and 8GHZ respectively. And CPI values 1.5, 2.8, and 2.0. Calculate the performance in instructions per second.

$$CPU\ Time = \frac{instruction_count \times CPI}{Clock\ Rate}$$

$$Performance\ for\ L = \frac{5 \times 10^9}{1.5}$$

$$Performance\ for\ M = \frac{3 \times 10^9}{2.8}$$

$$Performance\ for\ N = \frac{8 \times 10^9}{2.0}$$

CPI

- ▶ We can have different instruction classes with different number of cycles
- ▶ Consider n classes

$$\text{Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

- ▶ For example, given classes X, Y, Z with CPI 4, 5 and 6.
- ▶ And instruction counts of 1, 2, 3 respectively
- ▶ Then the Clock Cycles will be:

$$\begin{aligned} &= (1 \times 4) + (2 \times 5) + (3 \times 6) \\ &= 32 \end{aligned}$$

Note: You can be given frequency instead of instruction count, the same computation applies by replacing frequency with instruction count

Key Performance Factors

- ▶ **CPI:** depends on the instruction type and the ISA details
- ▶ **Instruction Count:** measured using a profiler or simulator and is independent of implementation details
- ▶ **Clock Rate:** is usually given

$$CPU\ Time = instruction_count \times CPI \times clock_cycle$$

$$CPU\ Time = \frac{instruction_count \times CPI}{clock_rate}$$

Summary of the Performance Equation

$$CPU\ Time = \left(\frac{instructions}{program} \right) \times \left(\frac{cycles}{instruction} \right) \times \left(\frac{seconds}{cycle} \right)$$

- ▶ Instructions / program: dynamic instruction count
 - ▶ Function of program, compiler, instruction set architecture (ISA)
- ▶ Cycles / instruction: CPI
 - ▶ Function of program, compiler, ISA, micro-architecture
- ▶ Seconds / cycle: clock period
 - ▶ Function of the micro-architecture

The MIPS Rate

- ▶ Alternate measure of processor performance
- ▶ Refers to the rate at which instructions are executed
- ▶ **MIPS = millions of instructions per second**

$$MIPS = \frac{instruction_count}{ExecutionTime \times 10^6}$$

$$MIPS = \frac{frequency}{CPI \times 10^6}$$

- ▶ Limitations of MIPS include not accounting for:
 - ▶ Differences in ISAs between computers
 - ▶ Differences in complexity between instructions

Example

Compilers have profound impact on performance of an application. Consider a compiler X with instruction count $2.2E9$ and execution time 5 seconds. What is the MIPS rate:

$$MIPS = \frac{instruction_count}{ExecutionTime \times 10^6}$$

$$MIPS\ Rate = \frac{2.2 \times 10^9}{5 \times 10^6}$$

Benchmarking

- ▶ The instruction rate is not a useful measure of performance
- ▶ Instead a set of programs of interest should be used
- ▶ **Workload:** a set of tasks run on a computer that is either a representative of the actual program or the actual user application
- ▶ **Benchmarks:** are standard workloads chosen to compare performance across machines

Benchmark Principles

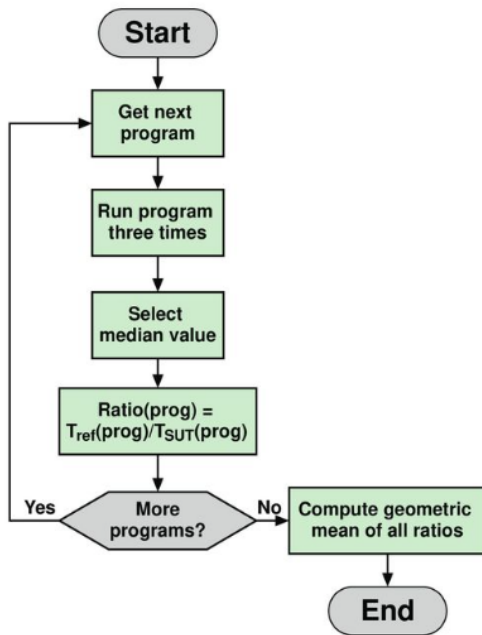
Desired characteristics of benchmark programs:

- ▶ Written in a **high-level language** for portability across different machines
- ▶ **Representative** of a particular kind of programming domain or paradigm, e.g. numerical, systems, or commercial programming
- ▶ Can be **measured easily**
- ▶ Have **wide distribution**

Examples include:

- ▶ SPEC
- ▶ GeekBench (others)
- ▶ PyPerformance (others)

SPEC Benchmarking Example



SPEC Benchmarking Results

Benchmark	Execution time	Execution time	Execution time	Reference time	Ratio
400.perlbench	3077	3076	3080	9770	3.18
401.bzip2	3260	3263	3260	9650	2.96
403.gcc	2711	2701	2702	8050	2.98
429.mcf	2356	2331	2301	9120	3.91
445.gobmk	3319	3310	3308	10490	3.17
456.hammer	2586	2587	2601	9330	3.61
458.sjeng	3452	3449	3449	12100	3.51
462.libquantum	10318	10319	10273	20720	2.01
464.h264ref	5246	5290	5259	22130	4.21
471.omnetpp	2565	2572	2582	6250	2.43
473.astar	2522	2554	2565	7020	2.75
483.xalancbmk	2014	2018	2018	6900	3.42

Reporting or Summarizing Performance Results

- ▶ **Arithmetic Mean:** is an appropriate measure if the sum of all the measurements is a meaningful and interesting value
 - ▶ For units that are proportional to time (latency)

$$AM = \frac{x_1 + \cdots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i$$

- ▶ **Harmonic Mean:** when a system's execution rate may be viewed as a more useful measure of the value of the system
 - ▶ For units that are inversely proportional to time (throughput)

$$HM = \frac{n}{\left(\frac{1}{x_1}\right) + \cdots + \left(\frac{1}{x_n}\right)} = \frac{n}{\sum_{i=1}^n \left(\frac{1}{x_i}\right)} \quad x_i > 0$$

- ▶ **Geometric Mean:** provides consistent results when measuring the relative performance of machines
 - ▶ For unitless quantities (speedup ratios)

$$GM = \sqrt[n]{x_1 \times \cdots \times x_n} = \left(\prod_{i=1}^n x_i\right)^{1/n} = e^{\left(\frac{1}{n} \sum_{i=1}^n \ln(x_i)\right)}$$

SPEC Benchmarking Results

Benchmark	Execution time	Execution time	Execution time	Reference time	Ratio
400.perlbench	3077	3076	3080	9770	3.18
401.bzip2	3260	3263	3260	9650	2.96
403.gcc	2711	2701	2702	8050	2.98
429.mcf	2356	2331	2301	9120	3.91
445.gobmk	3319	3310	3308	10490	3.17
456.hammer	2586	2587	2601	9330	3.61
458.sjeng	3452	3449	3449	12100	3.51
462.libquantum	10318	10319	10273	20720	2.01
464.h264ref	5246	5290	5259	22130	4.21
471.omnetpp	2565	2572	2582	6250	2.43
473.astar	2522	2554	2565	7020	2.75
483.xalancbmk	2014	2018	2018	6900	3.42

Performance Laws

- ▶ There are basically two equations on performance
- ▶ Provide insight in the performance of parallel and multicore systems:
 - ▶ Amdahl's law
 - ▶ Little's law

Amdahl's Law

Expresses a limitation in optimization: a speedup in one aspect does not result in a corresponding improvement in general performance

$$\frac{1}{(1 - f) + \frac{f}{N}}$$

$(1 - f)$ = *fraction of time spent executing sequential code*

(f) = *fraction of time spent executing parallelized code*

(N) = *number of processors*

Example

Consider that 45% of the execution time is spent in parallel code. What is the speed up with an infinite number of processors?

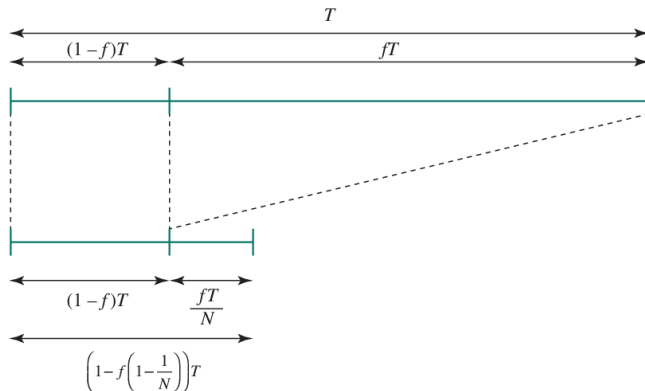
$$T = \frac{1}{1 - 0.45}$$

What is the speed up with $2 \times$ cores?

$$T = \frac{1}{(1 - 0.45) + \frac{0.45}{2}}$$

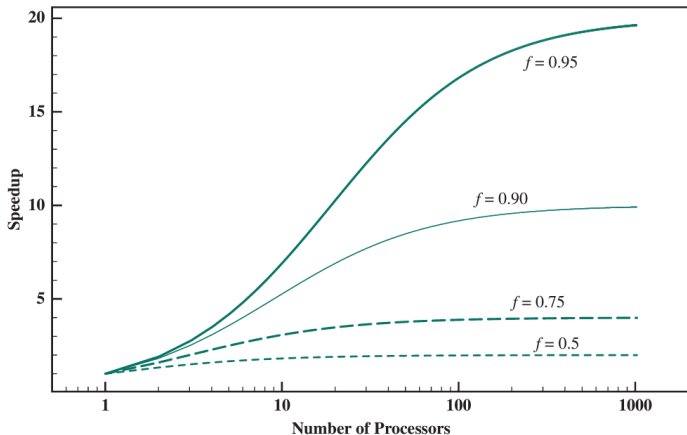
Amdahl's Law

Parallel processors have little impact when less time is spent on parallel code



Amdahl's Law

There are diminishing returns for using more processors



Little's Law

Assumes that the system is in steady state. i.e, average arrival time = average departure time

$$L = \lambda W$$

$L =$ *items in the system*

$\lambda =$ *average arrival rate*

$W =$ *average wait time*

Works on most queue-based systems

It works because it requires very few assumptions

We only need to measure two of the parameters

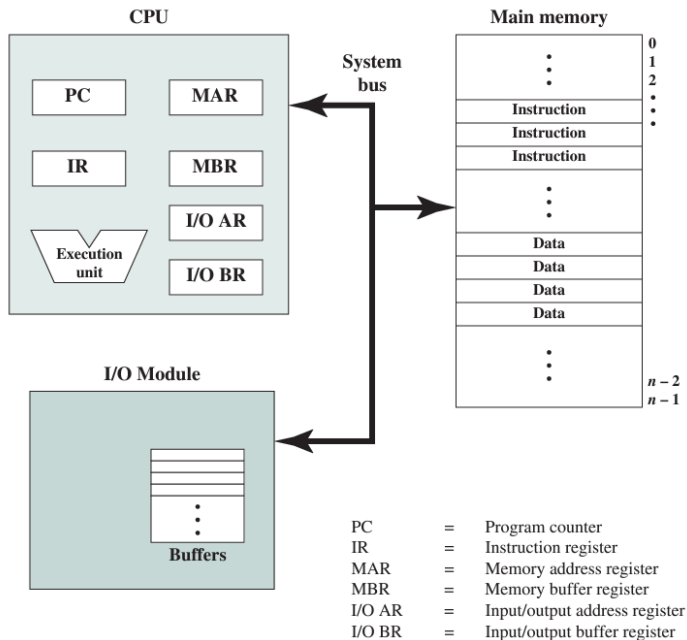
Example

Consider check-in of an airline where 385 passengers check in at 5:00 AM. The average number of passengers waiting for check-in is about 60. What is the average wait time for this airline.

$$W = \frac{L}{\lambda}$$

$$W = \frac{60}{385}$$

Top Level View



Overview of Top Level Components

- ▶ **Main Memory:** is made of a set of locations:
 - ▶ Locations are defined with linearly numbered addresses
 - ▶ Each location has a binary number that represents either an instruction or data
- ▶ **I/O Module:** moves data from the external environment to the CPU and memory
 - ▶ Data may not be sent immediately
 - ▶ Temporary internal buffers hold this data until it is ready to be sent
- ▶ **CPU:** executes program instructions
 - ▶ Has internal registers for exchanging data between memory and IO devices
 - ▶ The PC (points to the next instruction) and IR (contains a fetched instruction) registers are used while executing a program

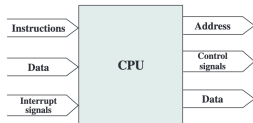
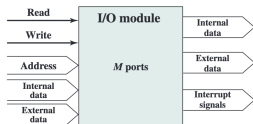
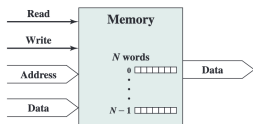
CPU Registers

- ▶ **Memory Address Register (MAR):** specifies the address in memory for the next read or write
- ▶ **Memory Buffer Register (MBR):** contains the data to be written into memory or receives the data read from memory
- ▶ **IO Address Register (I/O AR):** specifies a particular I/O device
- ▶ **IO Buffer Register (I/O BR):** exchange of data between the I/O module and the CPU

Interconnection for Components

Interconnection Structures: collection of paths connecting the various modules in a computer

- ▶ **Memory:** receives/sends data, receives addresses and control signals (read, write, timing)
- ▶ **CPU:** reads instructions and data, writes out data after processing, sends signals to other units, receives and acts on interrupts



I/O Connection

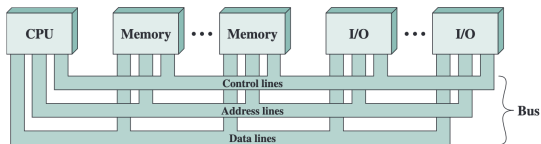
- ▶ Treated as memory
- ▶ Input
 - ▶ Receives data from peripheral
 - ▶ Sends data to computer
- ▶ Output
 - ▶ Receives data from computer
 - ▶ Sends data to peripheral
- ▶ Receives control signals from computer
- ▶ Sends control signals to peripherals
- ▶ Receives peripheral addresses from computer
- ▶ Sends interrupt signals

Buses

A bus is a communication pathway among devices. It has a shared transmission medium allowing one device transmission at a time

A bus can have many lines or pathways categorized as:

- ▶ **Data bus:** data lines that move data among system modules
- ▶ **Address bus:** designate the source or destination of the data on the data bus
- ▶ **Control bus:** control the access to and the use of the data and address lines



Control Signals

Control signals transmit both command and timing information among system modules

Timing signals: indicate the validity of data and address information

Command signals: specify operations to be performed

Control Signals

Typical control signals:

- ▶ **Memory write:** causes data on the bus to be written into the addressed location
- ▶ **Memory read:** causes data from the addressed location to be placed on the bus
- ▶ **I/O write:** causes data on the bus to be output to the addressed I/O port
- ▶ **I/O read:** causes data from the addressed I/O port to be placed on the bus
- ▶ **Transfer ACK:** indicates that data have been accepted from or placed on the bus

Control Signals ..

- ▶ **Bus request:** indicates that a module needs to gain control of the bus
- ▶ **Bus grant:** indicates that a requesting module has been granted control of the bus
- ▶ **Interrupt request:** indicates that an interrupt is pending
- ▶ **Interrupt ACK:** acknowledges that the pending interrupt has been recognized
- ▶ **Clock:** is used to synchronize operations
- ▶ **Reset:** initializes all modules

Point-to-point Interconnect

Point-to-point interconnect has lower latency, higher data rate, and better scalability compared to the shared bus


Main characteristics of QPI¹ and other point-to-point interconnect schemes:

- ▶ **Multiple direct connections:** Multiple components within the system enjoy direct pairwise connections to other components. This eliminates the need for arbitration found in shared transmission systems
- ▶ **Layered protocol architecture:** use a layered protocol architecture as in network environments, rather than the simple use of control signals found in shared bus arrangements
- ▶ **Packetized data transfer:** Data is sent as a sequence of packets (not raw bits), each of which includes control headers and error control codes

¹ Intel's QuickPath Interconnect (QPI)

Example QPI Four-layer Protocol Architecture

- ▶ **Physical:** Consists of the actual wires carrying the signals, as well as circuitry and logic to support ancillary features
- ▶ **Link:** Responsible for reliable transmission and flow control
- ▶ **Routing:** Provides the framework for directing packets through the fabric
- ▶ **Protocol:** The high-level set of rules for exchanging packets² of data between devices

²A packet is comprised of an integral number of Flits 

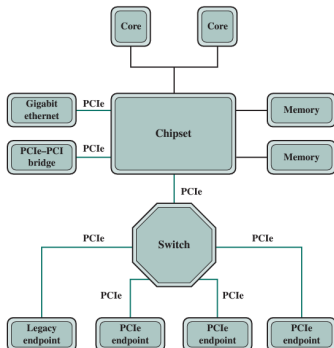
PCI Express

- ▶ The peripheral component interconnect (PCI) is a processor-independent bus
- ▶ It functions as a mezzanine or peripheral bus
- ▶ It delivers better system performance for high-speed I/O subsystems
- ▶ The bus-based PCI scheme does not keep pace with the data rate demands of attached devices
- ▶ An improved a point-to-point interconnect PCI Express (PCIe) was developed

PCI Express Architecture

A chipset connects the processor and memory subsystem to the PCI Express switch fabric comprising one or more PCIe and PCIe switch devices

- ▶ **Switch:** manages multiple PCIe streams
- ▶ **PCIe endpoint:** an I/O device or controller that implements PCIe
- ▶ **Legacy endpoint:** intended for existing designs that have been migrated to PCI Express
- ▶ **PCIe/PCI bridge:** allows older PCI devices to be connected to PCIe-based systems



PCI Protocol Architecture Layers

- ▶ **Physical:** consists of the actual wires carrying the signals, as well as circuitry and logic to support ancillary features
- ▶ **Data link:** Is responsible for reliable transmission and flow control. Data packets generated and consumed by the DLL are called Data Link Layer Packets (DLLPs)
- ▶ **Transaction:** generates and consumes data packets used to implement load/ store data transfer mechanisms and also manages the flow control of those packets between the two components on a link. Data packets generated and consumed by the TL are called Transaction Layer Packets (TLPs)