Computer Architecture MSBs: Definitions, Formulas, Rules of Thumb, and Key Topics Précis, v1.1

George B. Adams III

October 31, 2017

The following version 1.1 synopsis of the content of CS250 contains some material that is from or paraphrased from the writings of John Hennessy and David Patterson from the fronts pieces of their computer architecture textbooks familiarly known as COD, *Computer Organization and Design: The Hardware/Software Interface*, and CAAQA, *Computer Architecture A Quantitative Approach*.

1 Definitions

Abstraction: The removal of detail from the description of a system to improve the clarity of presentation for a subset of information

Antidependence: A artificial dependence resulting from the reuse of a name where correct program execution demands that the processor read from the name before it writes to that same name.

Big Endian: Storage order in which the byte in the most significant position (end) of a multi-byte data item is found in the lowest numbered memory address, and bytes of successively lesser significance are found at successively higher numbered addresses.

Clock rate: Inverse of clock cycle time, usually measured in units of MHz or GHz.

CPI: (average) Clock cycles per instruction.

Data dependence: A true, or essential, data dependence that occurs when a program instruction uses (reads) a data value that is produced (written) by an instruction appearing earlier in the program.

FLOP/s or *flops*: FLoating-point OPerations / second. It is important to state the number of bits in the operands when reporting FLOP/s; 64 bits is typical.

Hit rate: Fraction of memory references found in cache; equal to 1 − Miss rate.

Hit time: Memory access time for a cache hit, including the time to determine if the access is a hit or a miss.

IEEE Standard for Floating Point Arithmetic (IEEE 754): A technical standard for floating-point computation established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE) to make software using floating point arithmetic easier to write, to use, and to port between computers.

Instruction count: Number of machine instructions executed while running a program.

Little Endian: Storage order in which the byte in the least significant position (end) of a multi-byte data item is found in the lowest numbered memory address, and bytes of successively more significance are found at successively higher numbered addresses.

LSB: Least Significant Bit refers to the bit in a binary number representation that has the least power of two weighting.

MIMD: A multiprocessor or multicomputer that has Multiple Instruction streams each processing one of Multiple Data streams.

MISD: A multiprocessor or multicomputer that has Multiple Instruction streams each sequentially processing successive data items from a Single Data stream.

Miss penalty: The time to replace a block in the top level of a cache system with the corresponding block from the next lower level.

Miss rate: The fraction of memory references not found in the cache; equal to $1 -$ Hit rate.

MSB: Most Significant Bit refers to the bit in a binary number representation that has the greatest power of two weighting.

$N_{1/2}$: The minimum vector length sufficient to achieve FLOPS of one-half R_∞.

N_v: The minimum vector length sufficient for vector assembly language instructions to be faster than scalar assembly language instructions.

Output dependence: A artificial dependence that occurs because two program instructions write to the same name.

R_∞ or R_{peak}: The theoretically achievable floating point operations per second per second rate for an operation on infinite length vector operands.

Read after read (RAR): A type of operand sharing between machine instructions that cannot cause a data hazard.

Read after write (RAW) data hazard: Due to overlapped execution of instructions in a pipeline, an instruction may try to read an operand before an instruction appearing earlier in the program writes its result into the location of this operand, and so the instruction would get the old value.

SIMD: A multiprocessor or multicomputer that has a Single Instruction stream that processes Multiple Data streams.

SISD: A single processor or uniprocessor, which has a Single Instruction stream processing a Single Data stream.

Spatial locality (locality in space): A program that references a location in memory is likely to soon reference nearby locations in memory.

Temporal locality (locality in time): A program that references a location in memory is likely to soon reference that location again.

Write after read (WAR) data hazard: Due to overlapped execution of instructions in a pipeline, an instruction tries to write its result to a destination before that location is read by an instruction appearing earlier in the program, so the prior instruction would incorrectly get the new value. This hazard can only occur when the pipeline design allows some instructions to write results early in the pipeline and allows other instructions to read an operand late in the pipeline.

Write after write (WAW) data hazard: Due to overlapped execution of instructions in a pipeline, an instruction tries to write a result to a destination before that location is written to by a prior instruction, so after both instructions were complete the location would incorrectly hold the old value. This hazard is present only in pipeline designs that write in more than one pipe stage or allow an instruction to proceed even when a previous instruction is stalled.

2 Formulas

1. *CPU time* = Instruction count × Clock cycles per instruction × Clock cycle time
2. X is n times faster than Y: $n =$ Execution time$_\text{Y}$/Execution time$_\text{X}$
3. *Amdahl's Law*:

$$\text{Speedup}_\text{overall} = \frac{\text{Execution time}_\text{old}}{\text{Execution time}_\text{new}} = \frac{1}{1 - \text{Fraction}_\text{enhanced} + \frac{\text{Fraction}_\text{enhanced}}{\text{Speedup}_\text{enhanced}}}$$

4. *Average memory access time* = Hit time + Miss rate × Miss penalty

5. *Average misses per instruction* = Miss rate × Memory accesses per instruction

6. *Cache index size* : 2^{index} = Cache size/(Block size × Set associativity)

3 Rules of Thumb

1. *90/10 Locality Rule:* A program executes about 90% of its instructions using just 10% of its code.

2. *85/60 Branch-taken Rule:* About 85% of backward-going branches are taken, while about 60% of forward going branches are taken.

3. *2:1 Cache Rule:* The miss rate of a direct-mapped cache of size N is about the same as a two-way set-associate cache of size $N/2$.

4. *Watt-Year Rule:* The fully burdened cost of a Watt per year in a Warehouse Scale Computer in North America in 2011, including the cost of amortizing the power and cooling infrastructure, is about $2.

4 Key Topics Précis

4.1 Computer Abstractions

The five classic components of a computer are *input*, *output*, *memory*, *data path*, and *control*. The last two are sometimes combined and referred to as the *processor*. This organization is independent of hardware technology. Every piece of computer hardware can be placed into one of these five categories.

Hardware and software consist of a hierarchy of layers, with each lower layer hiding details from the layer above. This *Principle of Abstraction* is the way hardware and software designers cope with the complexity of computer systems. At the bottom of the hierarchy are the physical components of the computer; at the top are the user-facing applications. One key interface in the hierarchy of abstraction is the *instruction set architecture*: the interface between the hardware and the low-level software. This interface enables hardware implementations of varying cost and performance to run identical software.

4.2 Program Instructions

Today's computers are build on two key principles:

1. Program instructions are represented as strings of bits.

2. Programs can be stored in memory and accessed the same as any other information in memory. This is the *stored program* concept.

4.3 Computer Arithmetic

Bit strings have no inherent meaning. They can be used to represent CPU instructions, unsigned integers, floating-point numbers, and many more kinds of information. What a given bit string represents depends on the machine instruction that operates on the bits in the string. The key difference between numbers in a computer and numbers used by humans is that computer number representations have finite size and, hence, limited precision or limited range.

4.4 Assembly Language

Assembly language programs are inherently machine-specific and must be totally rewritten to run on a different computer architecture. Assembly language programs do not specify the type of data stored at a memory location, rather the program implies the type of the data by the operation the programmer chooses to use to process that data. All control flow in assembly programs is implemented with *go to* instructions. The primary reason to program in assembly language is when the speed and/or size of a program is *critically* important.

4.5 Performance

The only complete and reliable measure of computer performance is time. *CPU execution time* for a given program is the product of three relatively easy to make measurements of CPU activity:

1. the **instruction count**, which is number of instructions executed by the program,
2. the **CPI** (Clock cycles Per Instruction), which, note carefully, is the *average* over the instructions in the program, and
3. the **clock cycle time**, which is the duration of a clock cycle in seconds.

Thus, $CPU\ execution\ time\ in\ seconds = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ cycle}$

Before spending time and money to enhance the performance of hardware or software, first apply Amdahl's Law to the scenario to learn how much overall speedup to expect from the proposed effort. This knowledge may help make the time and money expenditure more proportional to the achievable return on that investment.

4.6 Pipelining

Pipelining increases the number of simultaneously executing instructions and the rate at which instructions are completed. Pipelining does not reduce the time it takes to complete an individual instruction. Pipelining improves *throughput* rather than the *latency* of individual instructions (the time from start of instruction fetch until the instruction is completed).

Although the hardware may not rely on the compiler to resolve hazard dependences to ensure correct execution, the compiler must understand the pipeline to achieve the best performance. Otherwise, unexpected stalls will reduce the performance of the compiled code.

Longer pipelines put more pressure on the compiler to deliver on the performance potential of the hardware. But data and control dependences in programs, together with instruction latencies, place an upper limit on delivered performance because the processor must sometimes wait for a dependence to be resolved.

4.7 Making Memory Fast, Large, and Affordable

The processor connects to the highest level of the memory hierarchy, which is implemented using the fastest but also most expensive memory technology to be found in the hierarchy. So the highest level is also necessarily the smallest level. Accesses that do hit in the highest level of the hierarchy can be processed quickly. Accesses that miss go to lower levels of the hierarchy, which are slower, use cheaper technology, and are, thus, larger. If the hit rate is high enough, the memory hierarchy has an average access time close to that of the highest and fastest level and a size equal to the slowest, cheapest, and largest level.

While *caches*, *translation lookaside buffers*, and *virtual memory* may initially look very different, they can be understood by looking at how they each deal with the same four questions:

1. Where can a block be placed?
2. How is a block found?
3. What block is replaced on a miss?
4. How are writes handled?

The challenge in designing memory hierarchies is that every change that potentially improves the *miss rate* can also negatively affect overall performance: Increasing size of a level decreases *capacity misses* but may increase access time, increasing *associativity* decreases *conflict misses* but may increase access time, and increasing *block size* may decrease miss rate yet increase *miss penalty*.

4.8 I/O

Different bus characteristics allow the creation of buses optimized for widely different demands.

I/O system performance depends on all the elements in the path between the peripheral device and computer memory, including the operating system the generates the I/O commands and executes the device driver.