University of Southern California

Viterbi School of Engineering

EE352
Computer Organization and Architecture

Performance Modeling

References:

1) Textbook
2) Mark Redekopp's slide series

Shahin Nazarian

Spring 2010

An Opening Question

- An Intel and a Sun/SPARC computer measure their respective rates of instruction execution on the same application written in C
  - Computer A achieves 160 MIPS (Million Instructions Per Second)
  - Computer B achieves 200 MIPS
- Which computer executes the program faster?
  - It depends on the instruction set and compiler (and ultimately, the instruction count). However computer B and its compiler may use many more simpler (faster) instructions to implement the program thereby increasing its instruction execution rate but saying nothing of the overall execution time

Another Question

- A Pentium 3 has a clock rate of 1GHz while a Pentium 4 has a clock rate of 2GHz
  - They implement the same instruction set
  - They are tested on the same executable program
- Is the Pentium 4 twice as fast as the Pentium 3?
  - Since they both use the same instructions and the same instruction count (same executable), we may think that the Pentium 4 would be twice as fast
  - However, the microarchitectural implementation of the processor may mean that the Pentium 3 executes instructions in 2 clocks on average while the Pentium 4 executes instruction in 4 clocks on average thus making the execution time exactly the same

Execution Time

- Execution time (and not the clock rate and # of instructions per sec by themselves) is the valid metric for comparing performance
- Two possible performance goals
  - Execution time: Measured for a single program's execution
  - Throughput: Total jobs performed per unit time

Wall Clock Time vs. CPU Time

- Even execution time can be hard to measure accurately because the OS may allocate a percentage of compute cycles to other programs (also, part of a program's execution time is spent in OS calls for I/O, etc.)
  - Wall Clock Time: Real time it took from when the user submitted the job until it was completed
  - CPU Time: Actual time the program took to execute when it was running

Performance

- **Performance is defined as the inverse of execution time**

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

- **Often want to compare relative performance or speedup (how many times faster is a new system than an old one)**

$$\text{Speedup} = \frac{\text{Performance}_{\text{New}}}{\text{Performance}_{\text{Old}}} = \frac{\text{Execution}_{\text{Old}}}{\text{Execution}_{\text{New}}}$$

Performance Equation

- Execution time can be modeled using three components
  - **Instruction Count**: Total instructions executed by the program
  - **Clocks Per Instruction (CPI)**: Average number of clock cycles to execute each instruction
  - **Cycle Time**: Clock period (1 / Freq.)

$$\begin{aligned}\text{Exec. Time} &= \text{Instruc. Count} * \frac{\text{Clocks}}{\text{Instruction}} * \frac{\text{Time}}{\text{Clock}} \\ &= \text{Instruc. Count} * \text{CPI} * \text{Cycle Time}\end{aligned}$$

Example

- Processor A runs at 200 MHz and executes a 40 million instruction program at a sustained 50 MIPS
- Processor B runs at 400 MHz and executes the same program (w/ a different compiler) which yields a count of 60 million instructions and a CPI of 6
- What is the CPI of the program on Proc. A?

$$CPI_A = \frac{200 * 10^6 \, cycles}{second} * \frac{second}{50 * 10^6 \, instrucs}$$

- Which processor executes the program faster & by what factor?

$$ExecTime_A = 40 * 10^6 \, instrucs . * \frac{second}{50 * 10^6 \, instrucs .} = 0.8 \sec$$

$$ExecTime_B = 60 * 10^6 \, instrucs . * \frac{6 cycles}{instruc .} * \frac{second}{400 * 10^6 \, cycles} = 0.9 \sec$$

$$Speedup = \frac{ExecTime_B}{ExecTime_A} = \frac{0.9}{0.8} = 1.125$$

- What is the MIPS rate of Proc. B? $MIPS_B = \frac{60 * 10^6 \, instrucs}{0.9 seconds} = 66.67 MIPS$

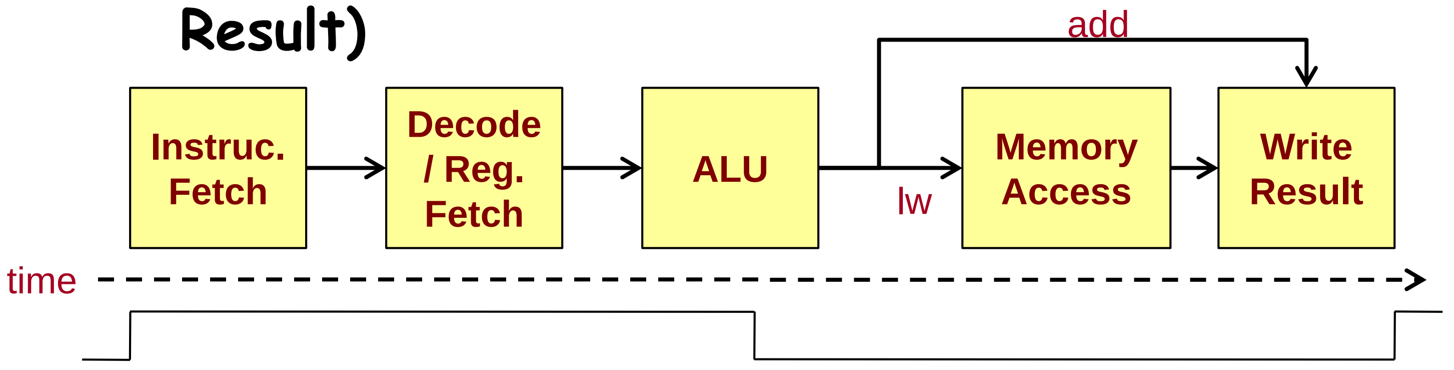
What Affects Performance

Component	SW/HW	Affects	Description
Algorithm	SW	Instruc. Count & CPI	Determines how many instructions & which kind are executed
Programming Language	SW	Instruc. Count & CPI	Determines constructs that need to be translated and the kind of instructions
Compiler	SW	Instruc. Count & CPI	Efficiency of translation affects how many and which instructions are used
Instruction Set	HW	Instruc. Count, CPI, & Clock Cycle	Determines what instructions are available and what work each instruction performs
Microarchitecture	HW	CPI, & Clock Cycle	Determines how each instruction is executed (CPI, clock period)

Source: H&P, Computer Organization & Design, 3rd Ed.

Microarchitecture's Effect

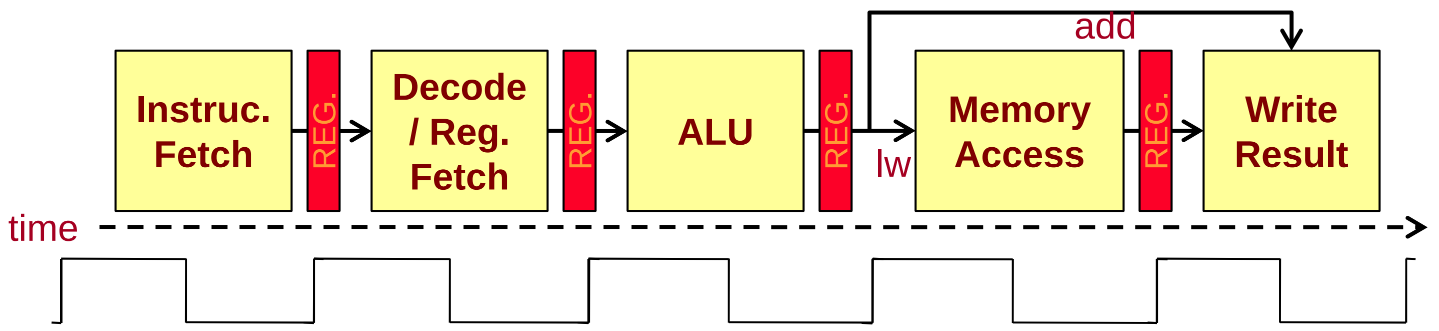
- Micro-architecture affects performance via the CPI and clock period
  - Consider LW and ADD instruction
    - LW (Fetch, Decode & Fetch Base Reg., Add Base+Offset, Read Mem., Write Result
    - ADD (Fetch, Decode & Fetch Src. Regs., Add, Write Result)



CPI_{add}=1

CPI_{lw}=1

T_{CLK}=50ns



CPI_{add}=4

CPI_{lw}=5

T_{CLK}=10ns

Calculating CPI

- CPI can be found by taking the expected value (weighted average) of each instruction type's CPI
  - i.e. CPI for each type * probability (frequency of occurrence) of that type of instruction

$$CPI = \sum_i CPI_{Type_i} * P(InstructionType_i)$$

- In practice, CPI is often hard to find analytically because in modern processors, instruction execution is dependent on earlier instructions
  - Instead we run benchmark applications on simulators to measure average CPI

Example

- Calculate CPI of this snippet of code using the following CPIs for each instruction type

```
        add     $s0,$zero,$zero
        addi    $t1,$zero,4
loop:   lw      $t2,0($t0)
        add     $t2,$t2,$t1
        addi    $t0,$t0,4
        addi    $t1,$t1,-1
        bne     $t1,$zero,loop
        sw      $t2,0($t2)
```

Instruction Type	CPI
add	1
lw / sw	4
bne	2

Dynamic Instruction Count = 4*5 + 3 = 23

Instruction Type	Dynamic Count
add	14
lw / sw	5
bne	4

$$CPI = \sum_i CPI_{Type_i} * P(InstructionType_i)$$

$$CPI = \frac{1}{23}\sum(1*14)+(4*5)+(2*4) = \frac{42}{23} = 1.826$$

Example

- Two different processors implement the same instruction set (such as Intel and AMD processors)
- There are four instruction classes (types) with the given CPIs
- P1 has a clock rate of 2GHz and P2 has a clock rate of 3GHz
- A certain program has an instruction mix in which classes A and B are executed twice as often as C and D
- Which computer is faster and by how much?

Average CPI (P1): 11/6

Average CPI (P2): 14/6

Instruction Type	CPI P1	CPI P2
A	1	2
B	2	2
C	2	2
D	3	4

Exec. Time (P1): IC * 11/6 * 0.5 ns

Exec. Time (P2): IC * 14/6 * 0.333 ns

Speedup = 11/12 / 14/18 = 1.17

CPI vs. IPC

- The reciprocal of CPI is IPC (Instructions per Cycle)
- Modern processors have the ability to execute more than one instruction simultaneously (superscalar)
- In the case of a 2-way superscalar, the maximum performance would be 2 instructions per clock cycle yielding a CPI of 0.5
- Thus, CPI is often inverted to IPC (e.g., max IPC is 2 instructions per cycle for the 2-way superscalar)

$$
\begin{aligned}
\text{Exec. Time} &= \text{Instruc. Count} * \text{CPI} * \text{Cycle Time} \\
&= \text{Instruc. Count} * \frac{1}{\text{IPC}} * \text{Cycle Time}
\end{aligned}
$$

Other Performance Measures

- FLOPS is the number of floating-Point Operations per Sec (similar definition to OPS [operations per sec] or IPS [instructions per sec])
  - Maximum number of arithmetic operations per second the processor can achieve
  - Example: 4 FP ALUs on a processor running @ 2 GHz => 8 GFLOPS
- Memory Bandwidth (Bytes/Sec.)
  - Maximum bytes of memory per second that can be read/written

Programs are either memory bound or computationally bound

Amdahl's Law

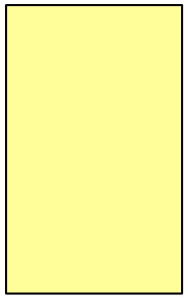
- Where should we put our effort when trying to enhance performance of a program
- Amdahl's Law is about how much performance gain do we get by improving only a part of the whole

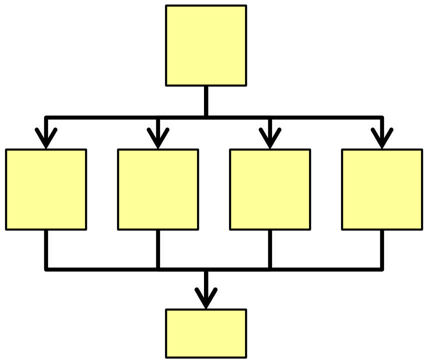
$$ExecTimeNew = ExecTimeUnaffected + \frac{ExecTimeAffected}{ImprovementFactor}$$

$$Speedup = \frac{ExecTimeOld}{ExecTimeNew} = \frac{1}{Percent_{Unaffected} + \frac{Percent_{Affected}}{ImprovementFactor}}$$

Amdahl's Law

- Holds for both HW and SW
  - HW: Which instructions should we make fast? The mostly used (executed) ones
  - SW: Which portions of our program should we work to optimize
- Holds for parallelization of algorithms (converting code to run multiple processors)


Original Sequential Program


Parallelized Program

Amdahl's Law Example

- A program consists of a single function with a loop. The loop body executes 10 times and consists of 5 instructions. The rest of the function consists of 50 instructions. Assume all instructions take the same amount of time to execute
- If we could somehow remove the 50 sequential instructions altogether, how much faster will our program run

$$Speedup = \frac{1}{Percent_{Unaffected} + \frac{Percent_{Affected}}{ImprovementFactor}}$$

$$Speedup = \frac{1}{0.5+0} = 2$$

Amdahl's Law Example

- A program consists of a single function with a loop. The loop body executes 10 times and consists of 5 instructions. The rest of the function consists of 50 instructions. Assume all instructions take the same amount of time to execute
- If we could reduce the loop by two instructions, how much faster will our program run

$$ExecTimeNew = ExecTimeUnaffected + \frac{ExecTimeAffected}{ImprovementFactor}$$

$$= 50 + \frac{50}{5/3}$$

$$= 80$$

$$Speedup = \frac{ExecTimeOld}{ExecTimeNew} = \frac{100}{80} = 1.25$$

Parallelization Example

- A programmer is porting his code to run on a Cell processor with its 1 main core (PPE, i.e., Power Processor Element) and 8 simple cores (SPEs, i.e., Synergistic Processing Elements)
  - 40% of the original program will still need to be executed sequentially on the PPE
  - Another 40% of the code can be parallelized into only 4 independent threads (thread = execution stream of a core)
  - The remaining 20% of the code can be fully parallelized to use all 8 SPEs
- What speedup will be achieved assuming all other factors are equal (clock speed, etc.)?

$$Speedup = \frac{1}{0.4 + \frac{0.4}{4} + \frac{0.2}{8}} = \frac{1}{0.525} = 1.905$$

[Optional] Power Processor Element (PPE)

- The *PPE* is the Power Architecture based, two-way multithreaded core acting as the controller for the eight SPEs, which handle most of the computational workload
- The PPE works with conventional operating systems due to its similarity to other 64-bit PowerPC processors, while the SPEs are designed for vectorized floating point code execution. The PPE contains a 32KB instruction and a 32KB data Level 1 cache and a 512KB Level 2 cache. The size of a cache line is 128 bytes
- Additionally, IBM has included an AltiVec unit which is fully pipelined for single precision floating point. (Altivec does not support double precision floating-point vectors.) Each PPU can complete two double precision operations per clock cycle using a scalar-fused multiply-add instruction, which translates to 6.4GFLOPS at 3.2GHz; or eight single precision operations per clock cycle with a vector fused-multiply-add instruction, which translates to 25.6GFLOPS at 3.2GHz

[Optional] Synergetic Processing Element (SPE)

- Each SPE is composed of a SPU (Synergistic Processing Unit) and a MFC (Memory Flow Controller,) i.e., DMA, MMU and bus interface)
- An SPE is a RISC processor with 128bit SIMD organization for single and double precision instructions. With the current generation of the Cell, each SPE contains a 256KB embedded SRAM for instruction and data, called "Local Storage" (not to be mistaken for "Local Memory" in Sony's documents that refer to the VRAM) which is visible to the PPE and can be addressed directly by software. Each SPE can support up to 4GB of local store memory. The local store does not operate like a conventional CPU cache since it is neither transparent to software nor does it contain hardware structures that predict which data to load

[Optional] SPE (Cont.)

- The SPEs contain a 128-bit, 128-entry register file and measures 14.5 mm^2 on a 90 nm process. An SPE can operate on sixteen 8-bit integers, eight 16-bit integers, four 32-bit integers, or four single-precision floating-point numbers in a single clock cycle, as well as a memory operation. Note that the SPU cannot directly access system memory; the 64-bit virtual memory addresses formed by the SPU must be passed from the SPU to the SPE MFC to set up a DMA operation within the system address space
- In one typical usage scenario, the system will load the SPEs with small programs (similar to threads), chaining the SPEs together to handle each step in a complex operation. For instance, a set-top box might load programs for reading a DVD, video and audio decoding, and display, and the data would be passed off from SPE to SPE until finally ending up on the TV

[Optional] SPE (Cont.)

- Another possibility is to partition the input data set and have several SPEs performing the same kind of operation in parallel. At 3.2 GHz, each SPE gives a theoretical 25.6 GFLOPS of single precision performance
- Compared to a modern PC, the relatively high overall floating point performance of a Cell processor seemingly dwarfs the abilities of the SIMD unit in desktop CPUs like the Pentium 4 and Athlon 64
- However, comparing only floating point abilities of a system is a one-dimensional and application-specific metric. Unlike a Cell processor, such desktop CPUs are more suited to the general purpose software usually run on personal computers. In addition to executing multiple instructions per clock, processors from Intel and AMD feature branch predictor

[Optional] SPE (Cont.)

- The Cell is designed to compensate for this with compiler assistance, in which prepare-to-branch instructions are created. For double-precision floating point operations, as sometimes used in personal computers and often used in scientific computing, Cell performance drops by an order of magnitude, but still reaches 12.8 GFLOPS (the PowerXCell 8i variant, which was specifically designed for double-precision, reaches 102.4 GFLOPS in double-precision calculations
- Recent tests by IBM show that the SPEs can reach 98% of their theoretical peak performance using optimized parallel Matrix Multiplication
- Toshiba has developed a co-processor powered by four SPEs, but no PPE, called the SpursEngine designed to accelerate 3D and movie effects in consumer electronics