Imperial College London

Computer Architecture: Year 2

4: Performance

Xin Wang

November 17, 2020

Contents

1 Introduction

Assessing the performance of computers can be quite challenging. The scale and intricacy of modern software systems have made performance assessment much more difficult.

Measuring the performance of a computer is a very relative aspect. Many different forms of performance metrics are available that measure different aspects that are of concern to the computer engineers.

2 Definition of performance

The term **performance** can be defined in many different ways. The meaning of the word often depends on the users of the computer systems. For example, users would be interested in the **response time** but data centers would be interested in **throughout**.

Response time or **execution time**: The total time required for the computer to complete a given task.
Throughout or **bandwidth**: Number of tasks completed per unit time.

3 Measuring performance

Time is a measure of performance and, within computer performance, execution time is measured in *seconds per program*. Time can be attached onto various aspects concerned with computer performance e.g. **response time** and **elapsed time**.

With the advance of CPU technology, CPU began to be able to be shared by multiple tasks thus measuring performance with "elapsed time" is not detailed enough. Instead, **CPU execution time** is used to show the time spent on a specific task.

CPU execution time: Actual time CPU spent on computing the specific task.

User CPU time: CPU time spent in a program itself.

System CPU time: CPU time spent in the operating system performing tasks on behalf of the program.

3.1 CPU performance and factors affecting it

Users and designers often examine performance using different metrics. If it is possible to relate these different metrics, it is possible to determine the effect of a design change on the performance as experienced by the user.

The most common measure of performance is execution time, the following formula relates the most basic metric to CPU time:

$$\text{CPU time for program} = \text{CPU clock cycles} \times \text{Clock cycle time}$$

Alternatively:

$$\text{CPU time for program} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle. But there is often a trade-off between the number of clock cycles needed for a program and the length of each cycle. Many techniques that decrease the number of clock cycles may also increase the clock cycle time.

3.2 Instruction performance

Note that the compiler generated instructions to execute, and the computer had to execute the instructions to run the program. This means that the execution time must depend on the number of instructions in a program.

One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction.

CPU clock cycles = Program instructions × Average clock cycles per instruction

Clock cycles per instructions (CPI): Average number of clock cycles per instruction for a program.

Different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program. CPI provides one way of comparing **two different implementations of the same instruction set architecture**, since the number of instructions executed for a program will, of course, be the same.

3.3 Classical CPU performance equation

This performance equation is written in terms of instruction count i.e. the number of instructions executed by the program, CPI, and clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

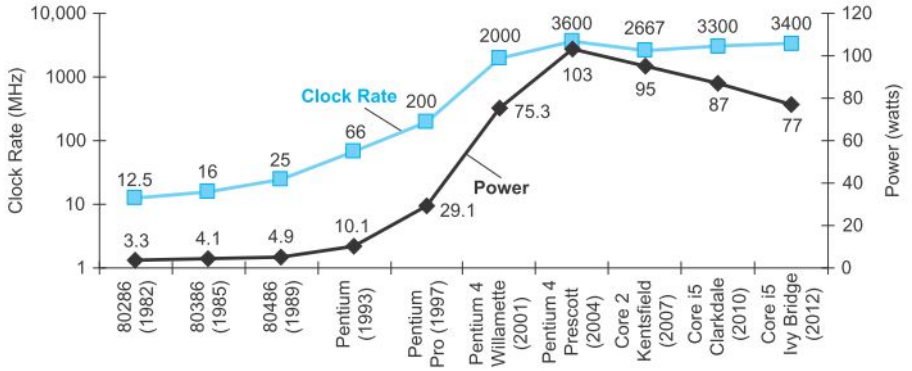
Figure 1: Summary of basic components of performance

The performance of a program depends on:

- The algorithm
- The language
- The compiler
- The architecture
- The actual hardware

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

4 Power wall



Note both clock rate and power increased rapidly for decades, and then flattened off recently. The reason for the growth together is that it is correlated, and the reason for the recent slowing is due to the practical power limit for cooling commodity microprocessors.

Energy is a critical resource. Battery life is more important than performance in the personal mobile device, and the architects of warehouse scale computers try to reduce the costs of powering and cooling 100000 servers as the costs are high at this scale.

Just like measuring time in seconds is a safer measure of program performance than a rate like MIPS (see Section 1.10), the energy metric joules is a better measure than a power rate like watts, which is just joules/second.

The defining equation on CMOS integration technology is:

$$\text{Power} = \text{Capacitative load} \times \text{Voltage}^2 \times \text{Frequency}$$

Example 1: Given a new CPU that has

- 85% the capacitative load of the old CPU
- Reduction of 15% on the voltage and frequency

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

5 Uni-processors to multi-processors

The power limit has forced a dramatic change in the design of microprocessors. Rather than continuing to decrease the response time of a single program running on the single processor, microprocessors, with multiple processors per chip, have the benefit of more on throughput rather than on response time.s

In the past, programmers could rely on innovations in hardware, architecture, and compilers to double performance of the programs. Today, for programmers to get significant improvement in program response time, programs need to be written to take advantage of multiple processors.

Parallelism has always been critical to performance in computing. A common example being pipelining, an technique that runs programs faster by overlapping the execution of instructions. This example is a form of instruction-level parallelism, where the parallel nature of the hardware is abstracted away so the programmer and compiler can think of the hardware as executing instructions sequentially.

Parallelism has always been less emphasised to programmers due to the following reasons:

- Parallel programming is by definition performance programming, which increases the difficulty of programming.
- To be fast in parallel hardware, the programmer must divide an application so that each processor has roughly the same amount to do at the same time and that the overhead of scheduling and coordination reduce the potential performance benefits of parallelism.

6 Performance and ISAs

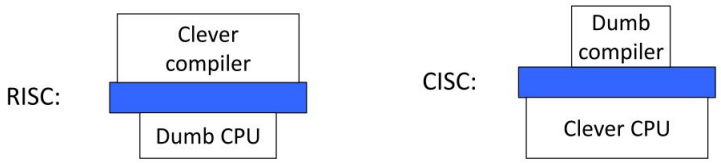
ISAs have a big influence on performance, good and bad.

- Good: ISAs may expose optimisation opportunities with compilers targeting specific instructions and ISA flexibility making it easier to tune micro-architecture.
- Bad: ISAs can act as a performance constraint by making it difficult to compile code efficiently and introduce complex critical path.

As ISAs are meant to be stable and last for many years, it is a careful balancing act between:

- Consistency and regularity for compilers
- Flexibility and opportunities for micro-architecture

There are two general approaches to ISAs:



- CISC:
  - Dense code, simple compiler
  - Powerful instruction set, variable format
- RISC:
  - Simple instructions, fixed format, optimising compiler
  - Speed, low development cost, adapt to new technology

7 Memory

- Register-register: ALU operations require 0 memory access
  - Pros: Simple, fixed-length insts taking similar cycle time, simple code generation model for compilers
  - Cons: High inst. count, wasted bits for simple insts
- Register-memory: ALU operations require 1 memory access
  - Pros: Data access without load; easy inst. encoding, dense code
  - Cons: Number of cycles per inst. depends on operand, not enough bits for register/memory address
- Memory-memory: ALU operands in memory
  - Pros: Low inst. count, don't waste registers for temporaries
  - Cons: Variation in inst. size and work per inst., slow due to memory accesses