ELEC S333 Computer And PC Designs (Free Courseware)

香港公開大學
THE OPEN UNIVERSITY OF HONG KONG

Contents

Chapter 1 Measuring computer performance

1.1 About this module

Welcome to this free courseware module ‘Measuring computer performance’!

This module is taken from the OUHK course *ELEC S333 Computer designs and performance (http://www.ouhk.edu.hk/wcsprd/Satellite?pagename=OUHK/tcGenericPage2010&c=C_ETPU&cid=191154102600&lang=eng)*, a ten-credit, Higher level course that is an optional course for the BSc and BSc (Hons) in Computer Engineering offered by the School of Science and Technology (http://www.ouhk.edu.hk/wcsprd/Satellite?pagename=OUHK/tcSubWeb&l=C_ST&lid=191133000200&lang=eng) of the OUHK. This course aims to provide students with in-depth knowledge in computer architecture, PC design and operating systems. Students will also be exposed to a range of design techniques and performance measurement.

ELEC S333 is mainly presented in printed format and comprises eight study units. Each unit contains study content, activities, self-tests, assigned readings, etc for students’ self-learning, and multimedia elements have been added where appropriate. This module (The materials for this module, taken from the print-based course ELEC S333, have been specially adapted to make them more suitable for studying online. In addition to this topic on ‘Measuring computer performance’, which is an extract from Unit 1 of the course, the original Unit 1 also includes the topic on determining computer performance.) retains most of these elements, so you can have a taste of what an OUHK course is like. Please note that no credits can be earned on completion of this module. If you would like to pursue it further, you are welcome to enrol in ELEC S333 Computer designs and performance (http://www.ouhk.edu.hk/wcsprd/Satellite?pagename=OUHK/tcGenericPage2010&c=C_ETPU&cid=191154102600&lang=eng).

This module will take you about **six hours** to complete, including the time for completing the activities and self-tests (but not including the time for assigned readings). Owing to copyright issues, textbook and assigned readings are not included in the free courseware.

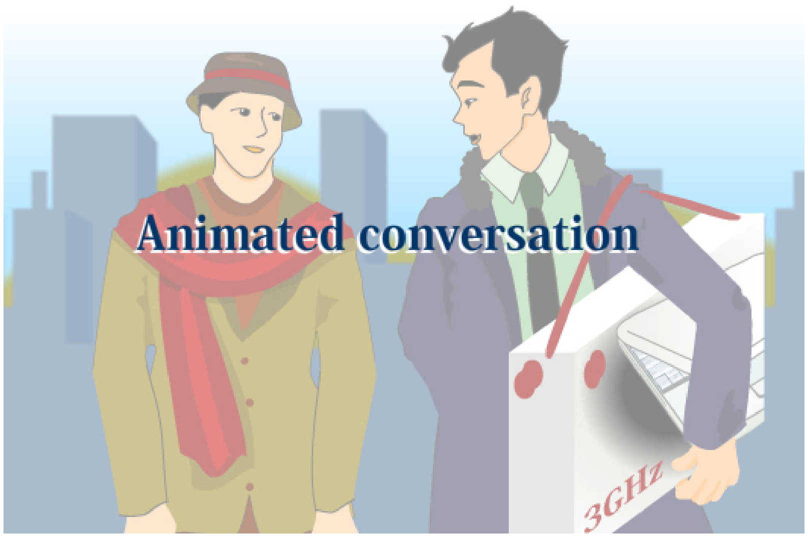
Good luck, and enjoy your study!

1.2 Introduction

Before we go into details on various computer designs concepts, we need to understand how to interpret the performance of a computer system. To start with, we will look at some basic metric and benchmarks for the measurement of a computer's performance.

1.3 Welcome to the world of computer system designs

Let's begin with a little drama. Mr ProudOfIt is telling Mr NoClue about his new computer.



Click this link to watch the video: http://www.opentextbooks.org.hk/system/files/resource/10/10075/10079/media/Animated%20Conversation.mp4

I have heard this type of conversation many times over the years. It is true that a faster processor improves the performance of a computer system. However, can we estimate how much better the performance will be? Will *Mr ProudOfIt* really experience a 6X difference in performance with his new system? Should *Mr NoClue* buy a new and faster computer if all he wants to do is to download his uncle's photo album faster? It is possible, however, that a 6X faster processor might not be able to improve the surfing experiences of *Mr ProudOfIt* and *Mr NoClue* dramatically. However, what would be the reasons behind this conclusion?

This course is all about computer system designs — not to design a completely new and different computer system *per se*, but an in-depth study of various design techniques used in today's modern computer systems. No book or course can cover the entire design space of computer technologies, and certainly no instructor can teach you how to design a completely new and technologically innovative computer system. After all, inventions and innovations are something no one can teach. The aim of this course is to teach and show you the various design techniques employed in today's computers and their individual components — the fundamental concepts, technical issues and tradeoffs, so that you can effectively and scientifically analyse and compare the performance of various computer systems that are built today (and in the future).

Precisely what do we mean when we say computer *X* performs better than computer Y? Before we can effectively and scientifically analyse and compare the performance of computer systems, first we need to learn how to measure the performance of a computer system, which is the focus of the rest of this unit. In this unit, you will study various scientific tools that are used in the field of computer science to measure the performance of a computer system. You will also study many different metrics that are used to indicate different types of performance. And at the end of the unit, a few fundamental concepts related to computer performance will be introduced to you. These concepts are important for understanding and analysing the performance of all computers and their components' designs, and will be used throughout the rest of the course. In the remainder of this section, we will briefly review the evolution of computer technologies first, followed by revisiting the *Mr ProudOfIt* and *Mr NoClue* dilemma. An overview of the course will be presented at the end of the section.

1.3.1 Computer technologies on the fast track

The computer industry has changed dramatically in the past half century -- today, for just a few thousand Hong Kong dollars, we can buy a computer system with capabilities that far exceed the mainframe systems constructed a few decades ago that cost tens of millions of dollars. Computers have evolved from a roomful of refrigerator-size specialized calculating machines such as the *ENIAC* (Electronic Numerical Integrator and Calculator) built by Eckert and Mauchly at the University of Pennsylvania during World War II (you can find a good history of ENIAC, with photos, here (http://www.library.upenn.edu/exhibits/rbm/mauchly/jwmintro.html)), to general-purpose systems that are compact enough to be placed on your desktops. Computer systems today not only perform mathematical calculations, but also a variety of tasks that even the early pioneers could not imagine. The evolution is amazing. Here is an example (For example, a group of researchers at University of Illinois Computer Science Department recently put together 70 SONY PlayStation 2 game consoles and created a powerful scientific computation cluster*. *http://arrakis.ncsa.uiuc.edu/ps2). These changes are credited to three significant advances in computer technologies:

1. The introduction of integrated circuit logic technology
2. The use of high-level programming languages
3. The operating systems themselves.

Instead of using hot and bulky vacuum tubes and miles of electronic cables, *integrated circuit logic technology* allows us to build electronic components that are far more efficient and cost effective. The transistor density of the integrated circuit logic technology improves at an average of 50% per year, or quadruples every three years, which has paved the way for replacing the large and bulky mainframe and minicomputers that were once fashionable with the compact microprocessors that are packed with astonishing capabilities today.

The demise of assembly language in favour of *high-level programming languages* such as FORTRAN, and C++ languages eliminates the need for extensive hardware compatibilities between computer systems. The use of high-level programming languages allows programmers to develop applications that can run on many different systems as long as the language compiler is available for that particular architecture. Together with the protection and common programming interfaces provided by the operating systems, such as virtual memory and POSIX interface, these technological revolutions give way to the rapid development and reproduction of software for commercial uses.

Traditionally, the term *personal computer*, or *PC*, stands for a small but general-purpose computer system that can be placed on our desktop to perform various limited computational tasks and for entertainment purposes. Today, many refer to PCs, as computer systems that contain the x86 processors (these processors are produced by Intel Corp. and AMD Corp.). This course, however, follows the traditional definition of the term PC, and discusses the architectural and implementation issues that cover the design spaces of both *x86* and *non-x86* general-purpose processors and their systems. It is also important for you to know that many techniques employed in traditional mainframe and super-computer systems for enhancing the system performance are now leveraged and implemented in the personal computers that sit in front of you and me. In fact, the tide has turned so much that many of today's high-end super-computer systems are constructed by inter-networking many general purpose processors together — the same processor that you use on your desktop or laptop day-in and day-out.

1.3.2 Clock rate does not equal performance!

Computer vendors often use performance numbers to indicate the advantages of their systems and to compare against their rivals. Those performance numbers are usually obtained from highly-tuned programs with tests that focus on highly-isolated criteria, such as the raw ability of a processor to perform a specific set of calculations.

However, the fact is that *the performance improvement of a computer system for general applications is not necessarily proportionate* to the increase in speed of the processor. Let's take the systems built with Intel Pentium 4 processors as an example (http://www.intel.com/content/www/us/en/library/benchmarks.html) -- the MPEG digital video encoding performance only improved by 23% (79 seconds down to 61 seconds) with a 42% increase (2.4 GHz to 3.4 GHz) in the processor speed on the Intel Pentium 4 systems.

Computer systems today are built with tens of thousands of individual components that are interconnected together. Each component performs its duty independently when a request is received from its interconnected neighbours. Unless the performance of every single component in the computer system improves at the same rate, the overall improvement experienced by the applications will be limited.

Let's revisit the story of Mr ProudOfIt and Mr NoClue.

Click this link to watch the video:
http://www.opentextbooks.org.hk/system/files/resource/10/10075/10081/media/Clock%20rate%20does%20not%20equal%20performance.mp4

Since networking and disk I/O activities are individual components that operate independently of the computer's processor, an increase in processor speed will have little or no effect on the performance of the networking and disk I/O activities. Unless the performance of the networking and I/O components are improved at the same rate as the system processor, we cannot expect a 6X improvement in the surfing experience for *Mr ProudOfIt* and *Mr NoClue*. This phenomenon can be observed through *Amdahl's Law* , a concept that we will formalize later in this unit.

Amdahl's Law

Gene Amdahl, a pioneer in our field of study, stated the following:

> *If F is the fraction of a calculation that is sequential, and (1-F) is the fraction that can be parallelized, then the maximum speedup that can be achieved by using P processors is 1/(F+(1-F)/P). (Amdahl, 1967)*

This is called *Amdahl's Law*, one of the fundamental laws in studying computer system performance. It states that the performance improvement to be gained from using a faster mode of execution is fundamentally limited by the fraction of the time the faster mode can be used.]

1.3.2.1 Activity 1

1. Do you have any practical suggestions for *Mr NoClue* in terms of speeding up his system? Suggest two or three other factors that might be slowing his system's performance.
2. Like *Mr NoClue*, imagine your computer games are running slowly. What might you do to improve your game-playing experience? (Think beyond buying a new computer system.)

1.3.2.1.1 Activity 1 Feedback

1. Given that Mr NoClue owns a 1 GHz system, the processor itself should be powerful enough for the job of processing the Internet operations. Therefore, Mr NoClue should investigate other possible reasons for the slowdown. There are many possibilities: lack of system memory, a faulty Internet connection equipment or setup, disk fragmentation, or maybe his friend's photo albums are stored on a Internet server that has a slow or congested uplink. Without proper investigation of the cause for the slowdown, Mr NoClue should not upgrade his computer system. (You might have come up with other reasons that are likely as valid as mine.)

2. If I want to improve my gaming experience, I should examine which part of my system is the performance bottleneck. A 3D graphic accelerator might help with the performance but only if the main processor spends the majority of processing time on 3D generations.

1.3.2.2 The Pentium 4 example

The early Pentium 4 made use of a feature called the 'instruction trace cache' and the clock rate of the earliest model was 1.3 GHz. The fastest model was the Pentium 4 670, with a clock rate of 3.8 GHz. Intel later redesigned the processor based on the 'Core' architecture, which can achieve the same performance at a lower clock rate. The second generation of the Core architecture, called the Core-2 architecture, first started with a clock rate of 1.8 GHz and its performance far exceeded the Pentium 4 670 at 3.8 GHz. This is a classic example that demonstrates how other architectural features can determine the performance of a processor, and of an overall computer system.

1.3.3 Course Overview

ELEC S333 is an advanced level course focusing on the designs and implementations of modern computer systems. Materials presented in this course rely heavily on your knowledge of two middle-level advisory pre-requisite courses — *ELEC S224 Microprocessor-Based Computers and COMP S260 Computer Architecture and Operating Systems*. In these courses, you have learned fundamental concepts such as computer arithmetic and organization, basic execution of machine instructions and instruction set architecture, data representation in the computer and basic memory management, the role of operating systems such as resources interface and scheduler, and so on. As much as we plan to structure *ELEC S333* into a self-contained course using brief reviews of related materials, you are strongly recommended to review the materials presented in *ELEC S224* and *COMP S260*, and use them as references as you proceed through this course.

Armed with the knowledge of computer system organization you acquired in *ELEC S224* and *COMP S260*, it is a natural step for you to explore these techniques more deeply, the designs and the implementations of today’s computer systems. The aim of this course is to provide an avenue for you to explore and to understand the hardware organization of computer systems. The in-depth understanding of the system hardware organization helps you to gain the necessary knowledge to analyse and criticize existing systems, as well as paving the road for you to become an excellent computer system designer.

The primary responsibility of a computer system designer is to *construct a system that performs well with minimal costs*. This requires a thorough understanding of the hardware organization as well as the mechanisms that interface the system hardware with the user programs. A typical computer system today consists of many individual components tightly interconnected together, such as:

- a processor core for executing program instructions
- a cache unit to store frequently used data
- a memory system where program and data reside
- an external disk storage system for storing programs and permanent data
- a set of external connectors (buses) that connects the computer to networks and other electronic devices
- a chipset that orchestrates the communications between these components.

As you might have noticed already, the key terminology here is the word *performance*. Therefore, the rest of the *Unit 1* material will focus on discussing the meaning of *performing well* and on the various techniques we use to measure the performance of computer systems.

Specifically, this unit:

- explains the role of a computer system designer;
- explains various ways to classify computer system performance including system throughput, response time and execution time;
- illustrates how to calculate cycles per instruction (CPI);
- relates instruction count, cycle per instruction and clock rate;
- explains the differences between theoretical and effective performance;
- distinguishes among and between various performance metrics including MIPS, MOPS and FLOPS;
- explains the differences between various popular performance benchmarks, in particular, the SPEC, TPC, INPACK and LAPACK, and ScienceMark;
- relates application performance with benchmarks results;
- describes the concepts of parallelism and critical path; and
- explains and estimates application performance using Amdahl's Law.

In addition to hardware organization, it is important for you to have a comprehensive knowledge of the instruction set architecture — the interface between the user programs and the computer hardware. Unit 2 will include a brief review of instruction set architecture, with an in-depth discussion of how instruction set architecture affects modern processor designs.

In *Units 3* and *4*, we will explore and examine various techniques that processor designers use today to improve the performance of the microprocessor. These include latency hiding and exploring parallelism at various levels.

A processor alone cannot perform computations — it requires memory to store the computation instructions and the data. In *Unit 5*, you will learn various memory designs and implementation techniques, and find out more importantly *why* they are crucial to the performance of a computer system.

As computer systems are used in different capacities, the connectivity of the system to the outside world is a very important topic. We devote *Units 6* and *Unit 7* to computer system external connectivity issues. Virtually all computers today are externally connected in one form or another. In Unit 6, you will be presented with materials that discuss and examine the various ways computers are connected through networking. In *Unit 7*, you will learn how computer systems are connected with external devices, the protocols they use, and examine the organization and performance of various peripheral buses.

Last but not least, in *Unit 8*, we conclude the course with an in-depth investigation of today's computer systems, and apply the knowledge learned throughout the course to analyse the performance results obtained from the example systems.

1.4 Measuring performance

'Sweetheart, how deep is your love?' Ms Snow asked.

'I love you more than you can imagine'. Mr Sunshine answered.

'Why do you say so?' Ms Snow asked again.

'I love you a lot because I buy you a jacket when you are cold; I cook when you are hungry; I will do anything for you because I love you'. Mr Sunshine answered.

'Well then, I guess I love you more because I let you to yell at me all the time!' Ms Snow replied.

The only reason to buy a newer, faster computer system is to improve the execution performance of software programs (unless you have a broken computer). My questions to you are: could you tell me precisely how to determine whether a computer is fast? How can you determine that computer *X* performs better than computer *Y*? *Mr ProudOfIt* and *Mr NoClue* used the speed of the processor to quantify their computer's performance; even *Mr Sunshine* used some countable but lame excuses to justify his love (uncountable) for *Ms Snow*.

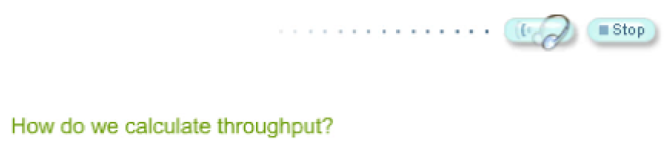
In the world of science, we use standardized mechanisms to measure and compare. For example, we use a ruler to measure the distance between objects in metres; we use a stopwatch with a given travel distance to measure the speed of the automobiles in kilometres per hour. We use these tools (such as ruler and stopwatch) combined with standardized mechanisms (such as running the ruler from myself to the object in a straight line and measuring the time for the car to travel a fixed distance) to compare entities using countable quantities (such as distance in metres and speed in kilometres per hour).

The single most important aim of this course is to make sure that you understand and are able to apply the underlying concepts of today's computer designs to compare and analyse the performance of present and future computer systems. To achieve this aim, you must first have a thorough understanding of the tools and means of measuring computer performance. In the remainder of this section, you will learn the common means to compare a computer's performance (metrics), and the tools that we use to obtain those means (benchmarks). First, let's define a couple of terms:

- **Execution time**: the time between the start and the completion of a job or an event. It is also commonly referred to as Response time.
- **Throughput**: the total amount of work completed in a given time.

Let's consider the following example:

Example



Click this link to watch the video: http://www.opentextbooks.org.hk/system/files/resource/10/10075/10086/media/Measuring%20performance.mp4

Despite many ways of measuring computer performance, you can see that *time* is the key element in all measurements -- the system that can complete the execution of a specific task in the minimum *time* is the faster.

Over the years, the use of execution time as a way to compare the performance of computers has diminished. Computer systems are now more often compared using *throughput*: how many simulations can be completed per hour? How many instructions can be executed per second? How much data can be transferred per second? We will examine some common metrics used to compare system performance later in this section.

It's time to state clearly the definition of performance we will use throughout the module:

Performance = Number of jobs or tasks completed in a given unit of time

=

If a computer system can execute an instruction every 10 milliseconds, we can say the performance of the computer system is:

Performance =

=

$= 100 \textit{ instructions per second}$

Now you have a basic understanding of these terms, it's time for you to acquire a deeper understanding of the metrics we use in the science of computers for comparing performance.

1.4.1 Self-test 1.1

If a processor can execute 1,200 instructions in 250 milliseconds:

1. What is the throughput of the processor per second?

$$\textit{Throughput} = \frac{\boxed{}}{\boxed{}} \times \boxed{} = \boxed{}$$

2. Calculate the execution time of program Z if Z contains 35,000 instructions.

$$\text{Execution time} = \frac{\boxed{}}{\boxed{}} = \boxed{}$$

1.4.1.1 Self-test 1.1 — suggested answer

1. Since 1000 milliseconds = 1 second, therefore the throughput of the processor is:

$$\textit{Throughput} = \frac{1000}{250} \times 1200 = 4800$$

2. The execution time of program Z is:

$$\text{Execution time} = \frac{35000}{4800} = 7.29$$

1.4.2 Basic performance metric

A computer system is a state machine that is composed of many individual circuitries driven by one or more internal clocks. These internal clocks are usually described by their frequency, or clock rate (e.g. 500 MHz). The term MHz is a combination of two acronyms, *M* and *Hz*:

M, or *mega* -- a quantity equals to a million, or 10^6.

Hz, or *Hertz* -- the number of cycles per second.

Therefore, a 500 MHz clock is equal to 500 million cycles per second, or 5×10^8 cycles per second. Over the years, much confusion has been cast over the acronyms for the quantities such as: kilo, mega, giga, tera, peta, and so on. Depending on the context, the acronyms may refer to two different quantities: a kilohertz (KHz) usually means 1,000 Hertz (1 × 10^3 Hertz), and a megahertz (MHz) usually means 1,000,000 Hertz (1 × 10^6 Hertz).

Kilo (K)	$= 1 \times 10^3$ or 1,000
Mega (M)	$= 1 \times 10^6$ or 1,000,000
Giga (G)	$= 1 \times 10^9$ or 1,000,000,000
Tera (T)	$= 1 \times 10^{12}$ or 1,000,000,000,000
Peta (P)	$= 1 \times 10^{15}$ or 1,000,000,000,000,000

When we compile a software program written in high-level languages like C++ or FORTRAN, the compiler translates the software program into an ordered collection of machine-understandable instructions. The processor will then execute these instructions through a number of cycles. Let's consider the following example.

Example

How long does it take to execute one instruction?

$$\text{Length of each cycle} = \frac{1}{500 \text{ X } 10^6}$$

$$= 2 \text{ X } 10^{-9} \text{ seconds}$$

Click this link to watch the video:
http://www.opentextbooks.org.hk/system/files/resource/10/10075/10089/media/Basic%20performance%20metric.mp4

The example you just seen is an over simplification of what is really going on under the hood of a computer system. In reality, virtually every computer system today is installed with a multi-programming-enabled operating system. Therefore, the execution time of a user program may include other activities such as waiting for an I/O event or running other jobs. Sometimes, we refer to the execution time of a program as the *wall-clock time*, which represents the total time elapsed since the start of the program. The time period that the CPU spends on executing user code is called *user time* or *CPU time*; and the time spent by the processor on other activities is called *system time*. Hence:

$$\text{Execution time} \begin{aligned} &= \text{Wall-clock time} \\ &= \text{User time} + \text{system time} \\ &= \text{CPU time} + \text{system time} \end{aligned}$$

We are only interested in the performance results measured from an *unloaded system* -- that means the operating system is not busy multiplexing other jobs onto the processor (i.e. minimize the system time). There are three major components in the previous example: the number of instructions in a program, the number of CPU cycles per instruction, and the length of each CPU cycle.

We can summarize these variables into the following equation, called the CPU performance equation:

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cyles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Cycles}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instructions} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{Cycles per instructions}}{\text{Clock rate}}$$

Note that *instruction count (IC)* refers to the total number of instructions executed by the processor, and that *cycles per instruction (CPI)* is the *average* number of cycles per instruction.

Definitions

Instruction count (IC): total number of instructions executed by the processor.

Cycles per instruction (CPI): the average number of cycles per instruction.

The CPU performance equation you've just learned gives us some clues about how we could improve the performance of a computer system:

1. Reduce the number of instructions executed by a program; or
2. Reduce the number of cycles per instruction; or
3. Reduce the time of each CPU cycle; or
4. Any combination of the above.

However, it is not easy to improve the system performance by reducing one variable without affecting others:

- The number of instructions to be executed by a program is determined by the instruction set architecture as well as the compiler technology. (This may or may not have anything to do with the program size. Sometimes, a small program size may have a large instruction count (e.g. a loop). On the other hand, expanding a loop many times will lead to a large program, but the number of instructions executed will decrease because of the reduction in the loop overhead);
- The number of cycles per instruction is dependent on the instruction set architecture and the machine organization; and
- The CPU cycle time depends on the hardware technology and the machine organization.

For example, it is very easy to create an instruction set that can reduce the total number of instructions in a program (e.g. the x86 instruction set versus the RISC instruction set). However, this change is likely to cause the total number of cycles per instruction to increase, off-setting the enhancement from the reduced instruction count.

In addition to the CPU performance equation, there are other metrics and standards that have been used for decades to compare the performance of computer systems. We'll review several of them in this section.

MOPS -- Millions of Operations Per Second

Sometimes we interchange *MOPS* with the term *MIPS -- Millions of Instructions Per Second*. MOPS refers to the total number of operations (instructions) executed, or that a processor can execute in a second. For example, a processor with a 500 MHz clock, with a CPI of 1, is capable of delivering 500 MOPS. This metric is somewhat misleading because the instruction count depends on the instruction set architecture, which is processor dependent. A FORTRAN program compiled on machine *X* may have a totally different instruction count on machine *Y*, which leads to inaccurate performance conclusions if MIPS is the only metric used in the comparison.

MFLOPS -- Millions of FLoating-point Operations Per Second

The majority of scientific applications run on computers today are packed with many floating-point calculations. MFLOPS measures the performance of the computer system to execute floating-point operations such as add, subtract, multiply and so on. MFLOPS refers to the total number of floating-point operations executed in millions per second. With today's high-powered processors, we often pronounce MFLOPS as *Mega-FLOPS*. Thus, *GFLOPS*, or *Giga-FLOPS* is 1,000 times more than MFLOPS; and *TFLOPS*, or *Tera-FLOPS* is 1,000,000 times more than a MFLOPS.

So, you might ask, who needs TFLOPS? Applications such as weather prediction will not be useful if the time required to generate the prediction is longer than the prediction period. For example, if it takes one hour of computing time in order to predict what will happen to the temperature in the next 10 minutes, why don't I just wait for 10 minutes and measure it for myself? Weather prediction involves complex fluid dynamics modelling of six or more variables with coordinates in a threedimensional space. The formulas and data are complex and large enough that only a TFLOPS system can effectively produce an accurate answer in a timely manner.

It is true that FLOPS is a more accurate measurement of performance since it counts the number of floating-point operations. However, caution has to be taken with the measurement since not all floating-point processors perform the same set of functions -- some processors, such as x86, include supports for complex mathematical operations (e.g. square-root) while other implementations do not. It is important that we normalize the floating-point operation count so that we can effectively compare the performance of the processors.

Example

Machine A	Machine B

Click this link to watch the video: http://www.opentextbooks.org.hk/system/files/resource/10/10075/10089/media/Example%202.mp4

In addition to these metrics, we also distinguish the difference between theoretical and actual performance obtainable from a computer system.

Theoretical peak performance

This is the absolute performance limit of a given computer system or device. For example, a computer system equipped with a 500 MHz floating-point co-processor that is capable of delivering a floating-point result per clock cycle is said to have a theoretical peak performance of 500 MFLOPS (500 millions cycles per second × 1 floating-point result per cycle). Likewise, a computer system equipped with two 500 MHz floating-point co-processors will have a theoretical peak performance of 1 GFLOPS.

Effective, or delivered performance

This is the *actual* performance obtained from the computer system. Delivered performance must be less than -- and seldom equal to -- the theoretical peak performance. There are many factors that affect the delivered performance of a computer system including the instruction mix and the memory latency. For example, a program consisting of 40% floating-point operations will have a maximum of 200 MFLOPS if the program is run on the computer system described above (500 MFLOPS × 40%). If the entire program is slower by a factor of two because of the memory latency, the delivered performance of the computer system will be 100 MFLOPS (500 MFLOPS × 40%/2).

For example, let's consider three n × n row-major matrices *$A_{(i, j)}$, $B_{(i, j)}$,* and where (**image**). The general form of any matrix A is defined as follows:

(**image**)

A *n* × *n* matrix multiplication operation multiplies the values in matrix *$A_{(i, j)}$* with values in matrix *$B_{(i, j)}$*, and the resulting matrix *$C_{(i, j)}$* is governed by the following calculations:

(**image**)

Question 1: how many floating-point add operations are in a matrix multiplication operation?

Answer: There are *n - 1* floating point add operations in each

$$\sum$$

operation, which are used for calculating the value of each $C_{(i,j)}$. Since there are *n x n* entries in $C_{(i,j)}$, therefore the total number of floating-point operations is $n^2(n-1)$

Question 2: how many floating-point multiply operations are in a *n x n* matrix multiplication operation?

Answer: there are *n* floating-point multiply operations in each

$$\sum$$

operation, which are used for calculating the value of each $C_{(i,j)}$. Since there are *n x n* entries in $C_{(i,j)}$, therefore the toal number of floating-point operations is n^3.

Question 3: State the floating-point operation performace in terms of *n* and *t,* where *t* is the execution time (in minutes) of the *n x n* matrix multiplication operation.

Answer: since the total number of floating-point operations in a matrix multiplication operation is equal to the sum of the number of floating-point add and floating-point multiply operations, therefore the total number of floating-point operations is $n^3+n^2(n-1)$. Therefore the floating-point performance is

$$\frac{n^3 + n^2\,(n-1)}{60 \times t \times 10^6}$$

MFLOPS.

Question 4: if machine *A* has a theoretical peak floating-point performance of 800 MFLOPS, how long will it take, in theory, to execute a 1024 x 1024 matrix multiplication operation?

Answer: by substituting performance = 800 MFLOPS and *n* = 1024 into the equation developed in Question 3, we have:

$$\frac{1024^3 + 1024^2\,(1024-1)}{60 \times 10^6} = 800MHz$$
$$2,146,435,072 = 48,000 \times t$$
$$t = 44,717.40\ minutes$$

Question 5: how long will it take to execute the same matrix multiplication operation if machine *A* has an average of 85% delivered performance?

Answer: if machine A has an average of 85% delivered performance, the performance delivered by the machine is 800 MHz×85%. By substituting the values into the equation developed in Question 3, we have:

$$\frac{1024^3 + 1024^2\,(1024 - 1)}{60 \times t \times 10^6} = 800MHz \times 85\%$$
$$2,146,435,072 = 48,000 \times t$$
$$t = 52,608.70\ minutes$$

1.4.2.1 Self-test 1.2

A program consisting of 1.6 million instructions is run on an unloaded 200 MHz system with a wall-clock time of 10 milliseconds. What is the average CPI of the system?

1.4.2.1.1 Self-test 1.2 — suggested answer

The average CPI of the system = 1.25

1.4.2.2 Self-test 1.3

Program X has the following characteristics:

- 20% of the instructions are memory operations;
- 50% of the instructions are performing integer calculations;
- 30% of the instructions are performing floating-point calculations.

Program X is compiled and run on machine A with the following characteristics:

- CPI of memory operations = 2;
- CPI of integer operations = 1;
- CPI of floating-point operations = 4.

1. Calculate the average CPI of the program X.
2. Calculate the CPU time of program X if X has 150 million instructions and machine A has a 200 MHz processor.

Now assume that a floating-point co-processor has been added to the system, and the CPI of floating-point operations is reduced to 2. Next:

3. Calculate the performance improvement (in percentages) of the floating-point operations.
4. Calculate the performance improvement (in percentages) of program X.
5. What is the clock rate required in order for the system to execute program X without the float-point co-processor and achieve the same performance?

1.4.2.2.1 Self-test 1.3 — suggested answer

1. $$Average\ CPI = (2 \times 0.2) + (1 \times 0.5) + (4 \times 0.3) = 2.1$$

2. $$CPU\ time = \frac{IC \times CPI}{Clockrate} = \frac{150 millions \times 2.1}{200 MHz} = 1.575$$

3. $$Since\ performance = \frac{1}{Executiontime},\ therefore:$$
$$Performance\ improvement = \frac{\frac{Clock\ rate}{IC \times 2} - \frac{Clock\ rate}{IC \times 4}}{\frac{Clock\ rate}{IC \times 4}} \times 100\% = 100\%$$

4. $$The\ average\ CPI\ of\ the\ new\ system$$
$$= (2 \times 0.2)\,(1 \times 0.5)\,(4 \times 0.3) = 1.5$$
$$The\ CPU\ time\ is\ therefore = \frac{150 millions \times 1.5}{200 MHz} = 1.125\ seconds$$
$$Thus,\ the\ performance\ improvement$$
$$= \frac{1.575 - 1.125}{1.575} \times 100\% = 28.57\%$$

5. $$Since\ CPU\ Time = \frac{IC \times CPI}{Clock\ rate},\ therefore:$$
$$\frac{150 million \times 2.1}{Clock\ rate} = 1.125\ seconds$$
$$Clock\ rate = 280 MHz$$

1.4.2.3 Self-test 1.4

Is the conclusion 'machine A has a higher floating-point operation performance' absolutely correct? Explain why the statement might be false.

1.4.2.3.1 Self-test 1.4 — suggested answer

The statement can be false since program X and program Y are different. Program Y might contain non-floating-point operations, which did not count into the MFLOPS metric but are still reflected in the total execution time. In order to determine the floating-point operation performance of the machines, the same program should be used to perform the tests.

1.4.3 Performance benchmarks

It would be ideal for performance if a computer system were tailored to a specific application. However, computers today are designed for running a wide range of applications. Therefore, it is important for computer designers and users to develop a set of tools that can be used to evaluate and predict performance of applications on various systems. You just learned the most common metrics we use in the science of computers for comparing performance. It is the time for you to learn the tools we use to obtain those metrics. We called these tools the performance benchmarks.

A *benchmark* is a software program that is used for obtaining the execution performance of a computer system based on a particular *workload*. The most effective workloads are the real programs that users will run to solve their problems. However, it is not easy to use the performance results of real programs to predict the performance of other applications. Therefore, many different types of performance benchmarks have been developed. Some benchmarks stress computational operations, while others may only focus on measuring the data transfer performance of computer systems.

Much effort has been put into extracting a few key operations that are the lowest common denominators among popular applications to evaluate system performance. *Micro-benchmarks* (also known as *kernels*) are small software programs that are being used to measure the performance of individual components or features of computer systems. Results obtained from these micro-benchmarks are often used to estimate the performance of real programs, and to explain the performance differences of real programs run on various computer systems.

Personal computers and servers are used for a variety of applications today. These applications put pressure on various components of the system. Therefore, it is rare that a single micro-benchmark result can be used for estimating the overall performance of systems. A *benchmarks suite* is a collection of micro-benchmarks that attempt to capture a particular set of workloads. NAS Parallel Benchmarks Suite (http://www.nas.nasa.gov/publications/npb.html) is one such benchmark suite developed by NASA Advanced Supercomputing division (NAS). The suite contains a set of eight computational fluid dynamics (CFD) kernels, which are used to evaluate the performance of parallel supercomputers. These CFD kernels are representative routines that are used frequently in various fluid dynamics simulations and modelling applications, such as simulating the physical fluid phenomena in weather systems and hypersonic aerospace vehicles.

In addition, we often use weighted average such as weighted arithmetic mean to indicate the relative importance of each application. For example, more weight would be given to graphic rendering benchmarks on systems that are intentionally designed for home entertainment use. Weighted arithmetic means of benchmark results are calculated using the following formula:

$$\sum_{i=1}^{n} Weight_i \times Executiontime_i$$

where *n* is the total number of benchmarks in the suite.

Example

The following table is the execution time of a benchmarks suite, which contains the program X and Y executed on computers A and B. Calculate the weighted arithmetic mean if the frequency of program X is 20% and program Y is 80%. Which system performs better?

	Machine A	Machine B
Program X	20 seconds	40 seconds
Program Y	65 seconds	10 seconds

Answer: the weighted arithmetic mean of machine *A* is (0.2 x 20) + (0.8 x 65) = 56 seconds. The weighted arithmetic mean of machine *B* is (0.2 x 40) + (0.8 x 10) = 16 seconds. Therefore, machine *B* performs better overall.

You probably have read about performance results many times in computer magazines and wondered how editors review the performance of newly available systems. The following are some common benchmarks that are used by system vendors and users today to compare the performance of various systems.

Standard Performance Evaluation Corporation (SPEC)

SPEC (http://www.spec.org/) is a non-profit organization aimed at establishing standards for measuring the performance of high-performance computer systems. *SPECint (integer operations performance)* and *SPECfp (floating-point operations performance)* are the two most frequently produced performance results by system vendors. Numerous revisions have been made to the SPEC benchmarks suite and SPEC CPU2000 is a benchmarks suite that is designed to measure the performance of CPU intensive tasks. This includes SPEC CINT2000 (measures the performance of integer intensive operations) and SPEC CFP2000 (measures the performance of floating-point intensive operations) (measures the performance of floating-point intensive operations).

Linear Algebra Package (LINPACK and LAPACK)

LINPACK (http://www.netlib.org/linpack/) and LAPACK (http://www.netlib.org/lapack/) are libraries of subroutines that are frequently used by scientific applications to solve systems of linear equations. They are real programs (subroutines) that are often linked to scientific applications to perform various linear algebra functions. They were originally developed to help scientific application writers develop programs that are portable to different computer systems. These subroutines are computation intensive, and some require good memory communication bandwidth. Many system vendors today provide the performance results of these subroutines to compete for scientific application customers. The *TOP 500 Supercomputer Sites (http://www.top500.org/)* listing uses the results produced by LINPACK as the basis for performance comparison.

Transaction Processing Performance Council (TPC)

TPC (http://www.tpc.org/) is a non-profit organization that was founded to standardize the performance benchmarks, and the review and monitoring process of performance results for the database transaction processing community. There are four major benchmarks that are in use today, and each of these benchmarks represents a different class of usage of the database system. Since databases perform large quantities of data disk read and write operations, TPC benchmarks are good performance indicators for the I/O subsystem.

Online Resources

Here are some online resources where you can find more information on popular benchmarks used by industrial vendors and reviewers to evaluate system performance:

- BenchWeb (http://www.netlib.org/benchweb)
- Science Mark (http://www.sciencemark.org)
- Top 500 Supercomputer Sites (http://www.top500.org)

Additional resources for the benchmarks described in this section:

- SPEC (http://www.spec.org)
- LINPACK (http://www.netlib.org/linpack)
- LAPACK (http://www.netlib.org/lapack)
- TPC (http://www.tpc.org)
- NAS Parallel Benchmarks (http://www.nas.nasa.gov/Software/NPB)

1.4.3.1 Activity 1.2

What is/are the potential issue(s) with weighted arithmetic mean to evaluate the performance of computer systems? Discuss.

1.4.3.1.1 Activity 1.2 — Feedback

The weighted arithmetic mean calculation gives a performance index, which does not show the distribution of the performance among the micro-benchmarks used. Consider the following example:

A benchmarks suite consists of two micro-benchmarks (X and Y) and the distribution of weights is: 80% of weight is given to benchmark X, and 20% of the weight is given to benchmark Y. Assume that the benchmarks suite is used to compare machine A and B with the following statistics:

	Machine A	Machine B
Benchmark X	10 seconds	30 seconds
Benchmark Y	100 seconds	30 seconds
Weighted arithmetic mean	28	26

From the weighted arithmetic mean, we can conclude that machine B performs better. But is the mean a fair representation of machine A's performance with benchmark X? The problem arises when the workload mix selected by the benchmarks suite and the weights distribution is only a close estimate to the actual usage. Suppose user Z wants to use the performance results for his workload mix, which comprise 95% of programs similar to benchmark X, and only 5% of the programs similar to benchmark Y. The following table shows the problem with the weighted arithmetic mean calculation:

	Results with old weights		**Results with new weights**	
	Machine A	Machine B	Machine A	Machine B
Benchmark X	10 seconds	30 seconds	10 seconds	30 seconds
Benchmark Y	100 seconds	10 seconds	100 seconds	10 seconds
Weighted arithmetic mean	28	26	14.5	29

Since the benchmarks suite uses a weight distribution close to user Z's usage (95:5 verses 80:20), user Z will select machine B based on the results. However, machine A actually performs better for his/her workload, which cannot be concluded from the weighted arithmetic mean results published by the benchmarks suite.

1.4.3.2 Activity 1.3

Try to find performance results concerning your computer system online. What are the benchmarks used? What are the characteristics of those benchmarks? How does your computer perform when compared with other systems in the market?

1.4.3.2.1 Activity 1.3 — Feedback

Explore the Internet sites listed in the unit. It is okay if you cannot find any results for your computer, but try to understand the benchmarks, their characteristics, and why they are useful.

1.4.3.3 Activity 1.4

Download a copy of ScienceMark 2.0 from the ScienceMark website and examine your computer's performance. What are the micro-benchmarks used by the suite? What are their characteristics? Are the benchmark results consistent with your expectations? How do the benchmark results give you insight to your computer’s configuration?

1.4.3.3.1 Activity 1.4 — Feedback

ScienceMark benchmarks suite comprises processor and memory performance benchmarks. See the descriptions on the site for more details. Discuss the performance results obtained from your system with your colleagues.

1.5 References

Below are the resources referred to or cited by the developer(s) of the original unit:

Amdahl, G (1967) 'Validity of the single processor approach to achieving large-scale computing capabilities', *AFIPS Conference Proceedings*, 30: 483–5.

http://arrakis.ncsa.uiuc.edu/ps2 (http://arrakis.ncsa.uiuc.edu/ps2%20http://arrakis.ncsa.uiuc.edu/ps2)

http://www.intel.com/performance/desktop/consumer/consumer_digital_video2.htm (http://www.intel.com/performance/desktop/consumer/consumer_digital_video2.htm%20http://www.intel.com/performance/desktop/consumer/consumer_digital_video2.htm)

1.6 Conclusion

In this module, you have learned the tools and means, and the fundamental principles that prepare you to measure and compare the performance of various computer systems. In particular, you have learned different types of software benchmarks and their respective metrics to measure the performance of computer systems and their individual components quantitatively.

If you would like to learn more on this subject, you are welcome to enrol in *ELEC S333 Computer designs and performance (http://www.ouhk.edu.hk/wcsprd/Satellite?pagename=OUHK/tcGenericPage2010&c=C_ETPU&cid=191154102600&lang=eng)* offered by the School of Science and Technology (http://www.ouhk.edu.hk/wcsprd/Satellite?pagename=OUHK/tcSubWeb&l=C_ST&lid=191133000200&lang=eng) of the OUHK.