PERFORMANCE

Lecturer: Hui Annie Guo

h.guo@unsw.edu.au

K17-501F

Lecture overview

- **Topics**
  - **Concepts**
    - **latency and throughput**
  - **Processor performance**
    - **instruction count, CPI, clock cycle time**
  - **Improving performance**
    - **Amdahl's Law**
  - **Benchmarks**

- **Suggested reading**
  - **H&P Chapter 1.6**

How to quantify performance?

- **Common performance metrics:**
  - **Latency:** response time, execution time, elapsed time
    - **A good metric for fixed amount of work**
      - E.g. How long does it take to execute a task?
        How long does it take to return a query?
  - **Throughput:** work per unit time
    - **A good metric for fixed amount of time**
      - E.g. What is the average execution rate?
        How much work can be done per unit time?

Good performance

- Ideally, for good performance we want
  - Latency (time) minimised
  - Throughput (work) maximised
  - Both may not be achieved at the same time

In-class exercise

- **Assume we have two machines: M1 and M2. M1 can complete an encryption in 1*ns*. M2 needs 5*ns* to run the encryption but can finish 30 encryptions in 15ns. Which machine is better?**

Performance comparison

- **When comparing the performance of two designs for a given task, we often use their execution time.**
  - **Given two designs, X and Y, if X is *n* times faster than Y, that means**

    exec_time(Y)/exec_time(X) = n

Processor performance

- **Evaluated based on the CPU time**
  - **Time in seconds the CPU spends on executing a program**

$$\text{cpu\_time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

  - **Comprised of three components**
    - **instruction count = instructions/program**
    - **CPI = average cycles/instruction**
    - **clock period (or clock cycle time) = seconds/cycle**
      - clock frequency = 1/(clock period)

CPI – average cycles per instruction

- **For a processor with varied instruction execution time, the CPI can be determined by**

$$CPI = cycles/instr\_count$$
$$= \sum_{i=1}^{n} CPI_i * \frac{I_i}{instr\_count}$$

Instruction frequency

n = total number of different instructions available in ISA
I_i = instruction count for instruction type i
$CPIi$ = number of clock cycles per instruction type i

In-class exercise

(1) If two machines have the same instruction set architecture (ISA), which of the following will always be identical for a given assembly program?

a) **clock rate**

b) **CPI**

c) **execution time**

d) **# of instructions**

(2) With the two machines of the same ISA, suppose for some program,

machine A has a clock cycle time of 250ps and a CPI of 2.0, and machine B has a clock cycle time of 500ps and a CPI of 1.2 .

Which machine is faster for this program, and by how much?

Impact on processor performance

- **The various design levels of computer system affect performance differently**
  - **See some examples in the next slides**

Design level	Instr. count	CPI	Clock period
program	X	X	
compiler	X	X	
ISA	X	X	X
organization		X	X
technology			X

Example

$(a-4)^2$

program (a-4)*(a-4) a^2- 8a +16

compiler mul shift ((a+a)+(a+a))+((a+a)+(a+a))

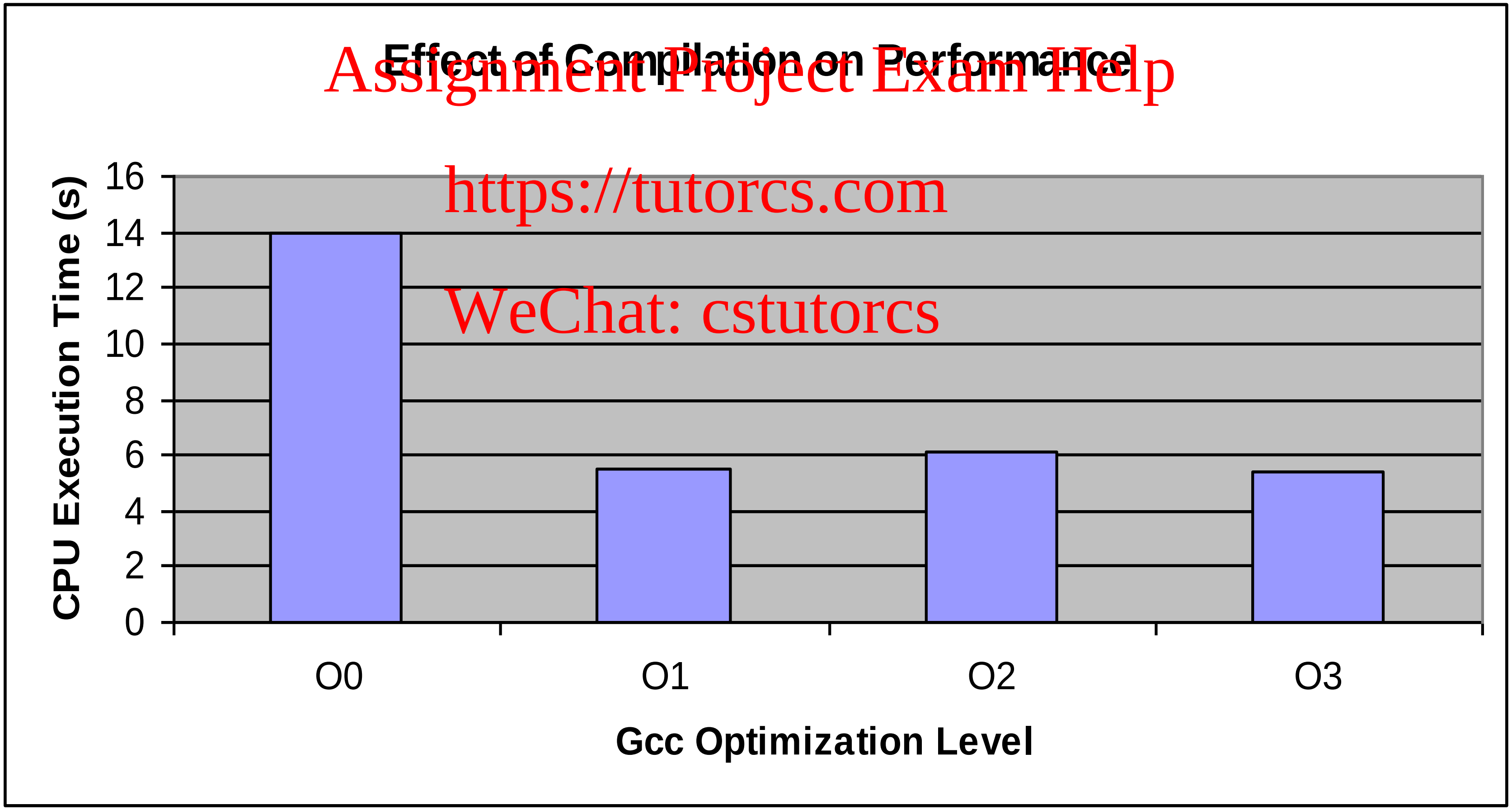
ISA shift3 shift1 shift1 shift1 Tclk(shift3)>Tclk(shift1)

organization 1 cc (70~80) cc

technology 65nm slow 5nm faster

Effect of compilation on performance

- **Experiment results**



In-class exercise

A compiler writer must decide between two code sequences for a high-level program targeted on a machine, given the following CPI information:

Instruction class	CPI for this inst. class
A	1
B	2
C	3

And the two code sequences have the following instruction counts:

Code sequence	A	B	C	total
1	2	1	2	5
2	4	1	1	6

(a) Which code sequence is faster?

In-class exercise (cont.)

(b) What is the CPI for each sequence?

Improving performance

- **Performance can be improved in many ways.**
  - **For example**
    - **Enhancing processor organization to lower CPI and clock cycle time**

    - **Enhancing compiler to reduce instruction count and CPI**

- **Improvements in one aspect may affect other aspects**

  - **For example**
    - **The compiler that replaces sequence 2 with 1 in the previous example reduces the instruction count only to increase CPI**

Speedup

- **We often use speedup**
  - **to measure the effectiveness of a design enhancement (improvement)**
- **Speedup due to enhancement *E*:**

$$\text{speedup(E)} = \frac{\text{exec\_time(\ \ )}}{\text{exec\_time(E)}} = \frac{\text{performance(E)}}{\text{performance(\ \ )}}$$

Gene Amdahl

Amdahl's law

- **Suppose that enhancement (*E*) accelerates a fraction (*F*) of a task by a factor (*S*) and the remainder of the task is unaffected.**

$$\text{exec\_time}(E) = ((1 - F) + F/S) \times \text{exec\_time}(\quad)$$

$$speedup(E) = \frac{1}{(1-F)+F/S}$$

- **Speedup is limited by the amount that the enhancement is used.**

In-class exercise

- **Suppose a program runs in 100 seconds on a machine, with multiply operations responsible for 80% of the execution. How much does the speed of the multiplication have to be improved for the program to run five times faster?**

$$\frac{100}{5} = \frac{80}{x} + 20$$

$$x \to \infty$$

Benchmarks

- **A computer's performance can be best determined by running real applications**
  - **using programs of typical expected workload, or expected class of applications**
    - **e.g., compilers/editors, scientific applications, graphics, etc.**
- **Small benchmarks**
  - **+ nice for architects and designers**
  - **+ easy to standardize**
  - **- can be abused**

Benchmarks (cont.)

- **SPEC (System Performance Evaluation Cooperative)**
  - **a set of real programs and inputs used by industry**
  - **valuable indicator of performance**
  - **can still be abused**

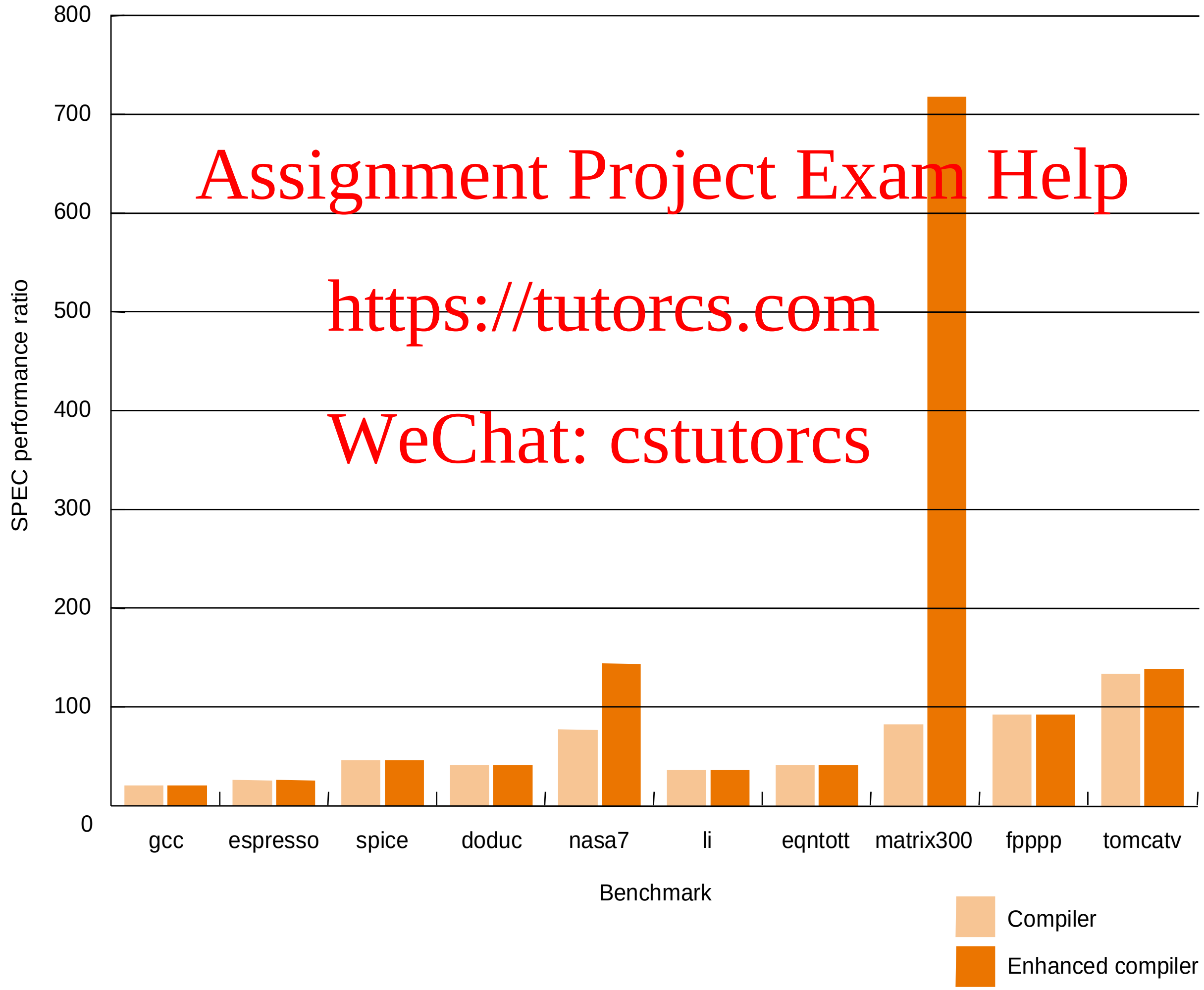
SPEC CPU2000

Integer benchmarks		FP benchmarks	
Name	**Description**	**Name**	**Type**
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

FIGURE 4.5 The SPEC CPU2000 benchmarks. The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

Example

Performance of two compilers



MiBench*

Auto./Industrial	Consumer	Office	Network	Security	Telecomm.
basicmath	jpeg	ghostscript	dijkstra	blowfish enc.	CRC32
bitcount	lame	ispell	patricia	blowfish dec.	FFT
qsort	mad	rsynth	(CRC32)	pgp sign	IFFT
susan (edges)	tiff2bw	sphinx	(sha)	pgp verify	ADPCM enc.
susan (corners)	tiff2rgba	stringsearch	(blowfish)	rijndael enc.	ADPCM dec.
susan (smoothing)	tiffdither			rijndael dec.	GSM enc.
	tiffmedian			sha	GSM dec.
	typeset				

About lab and testbench