



計算機結構 作業一

學號：111062107

姓名：鄧弘利

用 3.2 舊版

(a) (10 points) *Effects of algorithms on performance*

Press “Profile”  in the toolbar and select Profile as “Application Program”. Press the button “Resume”  in the debug window, record the CycC and InsC for the four functions listed in the table below and complete the table. Based on the characteristics of the programs, briefly compare and explain the differences between the naïve and fast power algorithms in their profiles.

Function	Source	CycC	InsC
naive_power_iter()	naive_power_iter.c	89	50
naive_power_recur()	naive_power_recur.c	218	135
fast_power_iter()	fast_power_iter.c	42	27
fast_power_recur()	fast_power_recur.c	118	73

We can see that fast_power_iter and fast_power_recur have less CycC and InsC than naïve version. It's because that they use faster algorithm and therefore they can have less instructions and clock cycles. They use like bitwise operations and that makes the calculations less.

(b) (10 points) *Effects of programming on performance*

From the table above and based on the characteristics of the programs, briefly compare and explain the differences between the iterative and recursive implementations of the fast power algorithm in their profiles. Suppose that they are executed in a processor with a clock rate of 3 GHz, what are the average CPI and CPU execution time for the fast_power_iter() and fast_power_recur() functions?

Function	Average CPI	Average Execution Time
fast_power_iter()	1.555	14×10^{-9}
fast_power_recur()	1.616	39.3×10^{-9}

The main difference between fast_power_iter and fast_power_recur is that recursion version generates more function calls and thus make the call stack bigger. That's why recursive version has more cycle counts.

Cycle per instruction = total cycle/instruction count

Fast_power_iter CPI : $42 / 27 = 1.555...$

Fast_power_recur CPI : $118 / 73 = 1.616...$

CPU time = clock cycle count / clock rate

Fast_power_iter CPU time : $42 / (3 \times 10^9) = 14 \times 10^{-9}$ (sec)

Fast_power_recur CPU time : $118 / (3 \times 10^9) = 39.3 \times 10^{-9}$ (sec)

(c) (10 points) **Effects of compilers on performance**

Compile fast_power_iter.c and fast_power_recur.c with two optimizations levels, -O0 and -O1. Record the CycC and InsC and compute their corresponding CPI for the two different optimization levels. Furthermore, briefly compare and explain the differences in their profiles.

Function	Optimization level -O0			Optimization level -O1		
	CycC	InsC	CPI	CycC	InsC	CPI
fast_power_iter()	191	87	2.19	42	27	1.55
fast_power_recur()	267	167	1.59	118	73	1.61

We can see CycC and InsC of O0 are higher than those of O1.

It's because optimization level -O0 doesn't optimize so the CycC and InsC are higher than those of optimization level -O1, since for O1, it optimizes for speed.

O0

Fast_power_iter CPI = $191/87 = 2.1954...$

Fast_power_recur CPI = $267/167 = 1.5988...$

O1

Fast_power_iter CPI = $42/27 = 1.5555...$

Fast_power_recur CPI = $1.6164...$

(d) (10 points) **Compilers versus hardware implementations**

If we want to run the -O0 codes compiled in (c) on a faster processor to achieve the same speedup as running the -O1 codes on the original processor in fast_power_iter() and fast_power_recur(), what will the clock rates of the faster processor be for fast_power_iter.c and fast_power_recur.c respectively?

CPU time for O1

Fast_power_iter : $42 / (3 \times 10^9) = 1.4 \times 10^{-8}$ (sec)

Fast_power_recur : $118 / (3 \times 10^9) = 3.93 \times 10^{-8}$ (sec)

$1.4 \cdot 10^{-8} = 191$ / new clock rate for fast_power_iter

=> new clock rate for fast_power_iter = $136.4 \cdot 10^8 = 13.64 \cdot 10^9 = 13.64\text{GHz}$

$3.93 \cdot 10^{-8} = 267$ / new clock rate for fast_power_recur

=> new clock rate for fast_power_recur = $67.9 \cdot 10^8 = 6.79\text{GHz}$

(a) (5 points) Follow the link <https://nanoreview.net/en/soc-list/rating> and fill in the table below.

Core name	Peak frequency of the most performant block of cores (MHz)			
Snapdragon 8 Gen 3	One core	Three cores	Two cores	Two cores
	Cortex-X4	Cortex-A720	Cortex-A720	Cortex-A520
	3300	3150	2960	2260
Apple A17 Pro	Two cores		Four cores	
	Everest		Sawtooth	
	3780		2110	
Google Tensor G3 Pro	One core	Four cores		Four cores
	Cortex-X3	Cortex-A715		Cortex-A510
	2910	2370		1700

(b) (10 points) Suppose that we run three computer graphics and multimedia programs on all three smartphones:

Program A: Renders 114,000,000 pixels when viewing HW1.pdf.

Program B: Blurs the background of 2,000 images in the image gallery.

Program C: Processes 4,000 images in the image gallery.

For simplicity, we assume that the program only runs on a single core. The Samsung Galaxy S24 Ultra uses Cortex-X4, the Apple iPhone 15 Pro Max uses Everest, and the Google Pixel 8 Pro uses Cortex-X3. Furthermore, there is no other overhead. We are interested in the execution time (in seconds) and the clock cycles (in millions) of each smartphone. Use the provided information in the table and your answer in (a) to complete the table below.

Smartphone	Program A		Program B		Program C	
	Seconds	Clock Cycles	Seconds	Clock Cycles	Seconds	Clock Cycles
Samsung (Cortex-X4)	0.501	1653.3	74.9	247170	61.6	203280
Apple (Everest)	0.639	2415.42	71.7	271026	50.6	191268
Google (Cortex-X3)	0.745	2167.95	133.3	387903	85.1	247641

Samsung

ProgramA

Sec: $114000000 / 227.4M = 0.501$

Clock cycles: $3300 * 0.501 = 1653.3$ (millions)

ProgramB

Sec: $2000 / 26.7 = 74.9$

Clock cycles: $3300 * 74.9 = 247170$ (millions)

ProgramC

Sec: $4000 / 64.9 = 61.6$

Clock cycles: $3300 * 61.6 = 203280$ (millions)

Apple

ProgramA

Sec: $114000000 / 178.5\text{M} = 0.639$

Clock cycles: $3780 * 0.639 = 2415.42$ (millions)

ProgramB

Sec: $2000 / 27.9 = 71.7$

Clock cycles: $3780 * 71.7 = 271026$ (millions)

ProgramC

Sec: $4000 / 79.1 = 50.6$

Clock cycles: $3780 * 50.6 = 191268$ (millions)

Google

ProgramA

Sec: $114000000 / 153\text{M} = 0.745$

Clock cycles: $2910 * 0.745 = 2167.95$ (millions)

ProgramB

Sec: $2000 / 15 = 133.3$

Clock cycles: $2910 * 133.3 = 387903$ (millions)

ProgramC

Sec: $4000 / 47 = 85.1$

Clock cycles: $2910 * 85.1 = 247641$ (millions)

(c) (10 points) We are interested in comparing the performances of the three smartphones. Calculate the relative performance of the three smartphones, with each phone as the reference for comparison. Use your answer in (b) to complete the table below and summarize the performance results by calculating the **geometric mean of the performance ratio** of the three benchmark programs (Program A, Program B, and Program C). **Hint:** you might only need to compute some of the six values from scratch.

Reference	Performance Ratio		
	Samsung Galaxy S24 Ultra	Apple iPhone 15 Pro Max	Google Pixel 8 Pro
Samsung Galaxy S24 Ultra	1	0.999	0.649
Apple iPhone 15 Pro Max	1.001	1	0.650
Google Pixel 8 Pro	1.541	1.540	1

我們可以看出 Apple 和 Samsung 的效能差不多，而 Google 的效能比較差。

Samsung as reference

$$\text{Apple} \Rightarrow \sqrt[3]{\frac{0.501}{0.639} \times \frac{74.9}{71.7} \times \frac{61.6}{50.6}} = 0.999$$

$$\text{Google} \Rightarrow \sqrt[3]{\frac{0.501}{0.745} \times \frac{74.9}{133.3} \times \frac{61.6}{85.1}} = 0.649$$

Apple as reference

$$\text{Samsung} \Rightarrow 1/0.999(\text{Apple with Samsung as reference}) = 1.001$$

$$\text{Google} \Rightarrow \sqrt[3]{\frac{0.639}{0.745} \times \frac{71.7}{133.3} \times \frac{50.6}{85.1}}$$

Google as reference

$$\text{Samsung} \Rightarrow 1/0.649(\text{Google with Samsung as reference}) = 1.541$$

$$\text{Apple} \Rightarrow 1/0.650(\text{Google with Apple as reference}) = 1.540$$

3. (10 points) **Performance and speedup**

Assume that a program requires the execution of 100×10^6 FP instructions, 140×10^6 INT instructions, 110×10^6 L/S instructions, and 55×10^6 branch instructions. The CPI for each type of instruction is 3, 2, 5, and 3, respectively. Assume that the processor has a 5 GHz clock rate.

(a) (5 points) By how much must we improve the CPI of INT instructions if we want the program to run two times faster? Please show the calculation procedure.

(b) (5 points) By how much is the execution time of the program improved if the CPI of FP instructions is reduced by 28%, the CPI of INT instructions is reduced by 32% and the CPI of L/S instructions is reduced by 61% and the CPI of branch instructions is reduced by 64%? Please show the calculation procedure.

(a)

CPU time = Clock cycles / clock rate

$$= (100 \times 10^6 \times 3 + 140 \times 10^6 \times 2 + 110 \times 10^6 \times 5 + 55 \times 10^6 \times 3) / (5 \times 10^9)$$

$$= 2 \times \text{improved CPU time}$$

$$= 2 \times (100 \times 10^6 \times 3 + 140 \times 10^6 \times \text{new CPI} + 110 \times 10^6 \times 5 + 55 \times 10^6 \times 3) / (5 \times 10^9)$$

$$\text{New CPI} = -2.625$$

So it can't be down if we just improve the CPI of INT instructions for the purpose of making the program run two times faster.

(b)

New CPU time = New Clock cycles / clock rate

$$= (100 \times 10^6 \times 3 \times 0.72 + 140 \times 10^6 \times 2 \times 0.68 + 110 \times 10^6 \times 5 \times 0.39 + 55 \times 10^6 \times 3 \times 0.36) / (5 \times 10^9) = 0.13606 \text{ (sec)}$$

$$\text{CPU time} = (100 \times 10^6 \times 3 + 140 \times 10^6 \times 2 + 110 \times 10^6 \times 5 + 55 \times 10^6 \times 3) / (5 \times 10^9) = 0.259 \text{ (sec)}$$

$$\text{CPU time} - \text{New CPU time} = 0.259 - 0.13606 = 0.12294 = \text{improved time}$$

4. (15 points) **Amdahl's law and the eight great ideas of computer architecture**

One of the great ideas of computer architecture is parallelization. Amdahl's law can be used to calculate the overall speedup of parallel executions. Amdahl's Law is defined as follow:

$$S_{latency} = \frac{1}{(1 - p) + \frac{p}{s}}$$

where

$S_{latency}$: the theoretical speedup of the execution of the whole task,

s : the speedup of the part of the task that benefits from improved system resources,

p : the proportion of the execution time that the part benefiting from improved resources originally occupied.

The ideal speedup of a parallelized program is the number of processors used. However, the theoretical speedups have limitations by the percentage of the application that cannot be parallelized, which includes the communication costs. The problem is that the communication costs are not fixed but often vary based on the number of processors used. In the following, let us consider the communication costs separately from the non-parallelizable execution of the program.

- (5 points) Suppose we have a method to parallelize the `fast_power_iter()` function in (1) using an arbitrary number of processors. Moreover, the execution time T on one processor is the result obtained in (1)(b). Compute the parallel execution time of `fast_power_iter()` on 2, 4, 8 processors assuming 75% of the function is parallelizable and there is no communication cost.
- (5 points) Assuming the communication costs are 4% of the original execution time regardless of the number of cores, what is the speedup with 8 cores when 75% of the program is parallelizable?
- (5 points) Assuming the communication costs are increased by 2% of the original execution time every time the number of processors is doubled, what is the speedup with n cores when 75% of the program is parallelizable? Furthermore, what is the specific speedup value when $n = 8$ in this scenario?

(a)

2 processors

$$1.4 * 10^{-8} / \left(\frac{1}{(1-0.75)+0.75/2} \right) = 8.75 * 10^{-9} \text{ (sec)}$$

4 processors

$$1.4 * 10^{-8} / \left(\frac{1}{(1-0.75)+0.75/4} \right) = 6.125 * 10^{-9} \text{ (sec)}$$

8 processors

$$1.4 * 10^{-8} / \left(\frac{1}{(1-0.75)+0.75/8} \right) = 4.8125 * 10^{-9} \text{ (sec)}$$

(b)

$$S(\text{latency}) = \left(\frac{1}{(1-0.75) + \frac{0.75}{8} + 4\%} \right) = 2.61$$

(c)

$$S(\text{latency}) = \left(\frac{1}{(1-0.75) + \frac{0.75}{n} + 2\% \cdot \log_2 n} \right) \quad \text{**} \underline{n \text{ stands for } n \text{ cores}}$$

When $n = 8$

$$S(\text{latency}) = \left(\frac{1}{(1-0.75) + \frac{0.75}{8} + 2\% \cdot \log_2 8} \right) = 2.48$$

5. (10 points) ***Integrated circuit cost and manufacturing***

Assume that a 50mm diameter-wafer has a cost of \$9 and contains 95 dies. The yield for this wafer is 90%.

- (a) (4 points) Find the defects per area for this wafer using the Equation on Page 28 of the textbook (or Page 43 of the slide of Introduction).
- (b) (2 points) Find the cost per die for this wafer.
- (c) (4 points) If the number of dies per wafer is increased by 10% and the defects per area unit increases by 25%, find the new die area and new yield.

(a)

Dies per wafer = wafer area / die area

$$\Rightarrow 95 = 3.14 \cdot (25 \cdot 10^{-3})^2 / \text{die area}$$

$$\Rightarrow \text{Die area} = 3.14 \cdot (25 \cdot 10^{-3})^2 / 95 = 0.0000206 \text{ (m}^2\text{)} = 20.6 \text{ (mm}^2\text{)}$$

$$\text{Yield} = 1 / (1 + (\text{defects per area} \cdot \text{die area} / 2))^2$$

$$\Rightarrow 0.9 = 1 / (1 + (\text{defects per area} \cdot 20.6 / 2))^2$$

$$\Rightarrow \text{defects per area} = 5.25 \cdot 10^{-3} \text{ (defects/mm}^2\text{)}$$

(b)

Cost per die for this wafer = cost per wafer / (dies per wafer * yield)

$$= 9 / (95 \cdot 0.9)$$

$$= 0.105$$

(c)

New Die area = wafer area / new dies per wafer

$$= 3.14 \cdot (25)^2 / (95 \cdot 1.1)$$

$$= 20.6/1.1$$

$$= 18.73 \text{ (mm}^2\text{)}$$

$$\text{New Yield} = 1 / (1 + (\text{defects per area} * \text{die area} / 2))^2$$

$$= 1 / (1 + 5.25 * 10^{-3} * 1.25 * 18.73 / 2)^2$$

$$\Rightarrow \text{new Yield} = 0.88755$$