CS 382 Fall 2024 | Computer Architecture and Organization | Homework 4

Breona Pizzuta
"I pledge my honor that I have abided by the Stevens Honor System."

Requirement (READ FIRST!) You have to type the solutions. Handwritten homework will not be graded and will receive zero credit. You can annotate on this document directly (you might need to know how to insert a picture into a PDF file); or you can submit a separate PDF, but with solutions clearly marked with question numbers.

1 (20 points)

Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3.2 GHz clock rate and a CPI of 1.5. P2 has a 2.0 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.3.

1.1 (7 points)

Calculate the performance of each processor expressed in instructions per second.

CPI= clock cycles per instruction

$$IPS = \frac{instructions}{sec} = \frac{instructions}{clock\ cycle} * \frac{clock\ cycle}{second} = \frac{clock\ rate}{CPI}$$

P1= $\frac{3.2x10^9 Hz}{1.5} = 2.13 * 10^9 \frac{instructions}{sec}$

P2= $\frac{2x10^9 Hz}{1} = 2 * 10^9 \frac{instructions}{sec}$

P3= $\frac{4x10^9 Hz}{2.3} = 1.739 * 10^9 \frac{instructions}{sec}$

1.2 (7 points)

If each of these processors executes a program in 10 seconds, calculate the number of cycles and the number of instructions.

# of instructions in 10 secs = IPS* 10

P1 instructions= $2.13 * 10^9 * 10 = 2.13 * 10^{10}\ instructions$

P2 instructions= $2 * 10^9 * 10 = 2 * 10^{10}\ instructions$

P3 instructions= $1.739 * 10^9 * 10 = 1.739 * 10^{10}\ instructions$

CPI= $\frac{cycle}{instruction}$

cycles = $\frac{cycle}{instruction} * instructions$

P1 cycles= $1.5 * 2.13 * 10^{10} = 3.195 * 10^{10}\ cycles$

P2 cycles= $1 * 2 * 10^{10} = 2 * 10^{10}\ cycles$

P3 cycles= $2.3 * 1.739 * 10^{10} = 4 * 10^{10}\ cycles$

1.3 (6 points)

We are trying to reduce the execution time of P2 by 30%, but this leads to an increase of 20% in the CPI. What should the clock rate be to obtain this reduction in execution time? P2 has a 2.0 GHz clock rate and a CPI of 1.0.

Old clock rate= 2 GHz

New clock rate=?

Old CPI= 1

New CPI= increase 20% = 1.2

P2 instructions= $2 * 10^9 * 10 = 2 * 10^{10}\ instructions$

P2's execution time is 10 secs

Clock rate= $\frac{instructions * old\ CPI * new\ CPI}{reduced\ time * execution\ time}$

$\frac{2*10^{10} * 1 * 1.2}{.7 * 10} = 3.4286\ GHz$

2 (20 points)

Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3 for the corresponding classes, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2. Given a program with a dynamic instruction count of 1.0e6 instructions divided into classes as follows: 30% class A, 20% class B, 30% class C, and 20% class D:

2.1 (7 points)

What is the total CPI for each implementation?

Using Weighted Average CPI: $$\text{Clock cycles} = \sum_{k=1}^{K} I_k \cdot CPI_k$$

P1: CPI= (1* 0.3) + (2* 0.2)+ (3* 0.3) + (3* 0.2)= 2.2
P2: CPI= (2* 0.3) + (2* 0.2)+ (2* 0.3) + (2* 0.2)=2

2.2 (7 points)

Calculate the clock cycles required in both cases.

CPI= $\frac{cycle}{instruction}$

cycles = $\frac{cycle}{instruction}$ * $instructions$

Instructions:
Class A: 1000000*.03= 30000 instructions
Class B: 1000000*.02= 20000 instructions
Class C: 1000000*.03= 30000 instructions
Class D: 1000000*.02= 20000 instructions

Clock cycles:
P1= (1*30000) + (2*20000) + (3*30000) + (3*20000)= 2200000
P2= (2*30000) + (2*20000) + (2*30000) + (2*20000)= 2000000

2.3 (6 points)

Which is faster: P1 or P2?

Execution time = $\frac{clock\ cycles}{clock\ rate}$

Given clock rates: P1= 2.5 GHz and P2= 3 GHz

P1: $\frac{2200000}{2.5*10^9} = 8.8 * 10^{-4} seconds$

P2: $\frac{2000000}{3*10^9} = 6.67 * 10^{-4} seconds$

P2 is faster because it has a shorter execution time than P1.

3 (20 points)

In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row of a matrix are stored contiguously. You can assume that undefined symbols are primitive types, i.e., int, double etc

```
for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {
        Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
        err = max(err, fabs(Anew[j][i] - A[j][i]);
    }
}
```

3.1 (7 points)

Which variables exhibit temporal locality?

Variables i, j, n, m and err exhibit temporal locality. Anew[j][i] would also exhibit temporal locality since the same element is accessed twice in a row in each iteration. Temporal locality means recently referenced data are likely to be referenced again in the near future.

3.2 (7 points)

Which variables exhibit spatial locality? Locality is affected by both the reference order and data layout.

Variable Anew and A would exhibit spatial locality. This is since elements are iterated through in the same row. We know that the rows are stored contiguous because the program is in row major order. This means the values in memory are stored next to each other since spatial locality means that data with nearby addresses tend to be used (referenced) close together in time. In this program, A also exhibits spatial locality in the cases of A[j][i-1] and A[j][i-1] since the values in memory will be next to each other in the same row j, but different columns i. On the other hand, we can not say that A[j-1][i] and A[j+1][i] exhibit spatial locality since we would be moving rows which would not be next to each other in memory.

3.3 (6 points)

Would the program be slower or faster if the outer loop iterated over i and the inner loop iterated over j? Why?

Here we know that arrays are stored in row-major order, meaning rows are laid out contiguously in memory. If the outer loop iterated over i, the program would process elements column by column (column-major order) instead of row by row. As a result, the elements of Anew would lose the advantage of spatial locality, leading to a slower program.

4 (20points)

Consider a byte addressing architecture with 64-bit memory addresses.

4.1 (5 points)

Which bits of the address would be used in the tag, index and offset in a direct-mapped cache with 512 1-word blocks?

Direct mapped= E (# lines per set)= 1
word= 4 bytes= 32 bits
m= # bits= 64
B= # of bytes is the block size B (bytes per line) = 4 bytes per line
S= # sets (rows)= 512 sets
C= S*E*B = 512*1*4= 2048 bytes
b=$\log_2$B bits block offset = $\log_2$4= 2
s=$\log_2$S bits for set index = $\log_2$512= 9
t = m-s-b = 64-9-2=53

tag	set index	block offset
53	9	2
All other most significant 53 bits	9 least significant bits after the block offset	Least significant 2 bits

4.2 (5 points)

Which bits of the address would be used in the tag, index and offset in a direct-mapped cache with 64 8-word blocks?

Direct mapped= E (# lines per set)= 1
word= 4 bytes= 32 bits
m= # bits= 64
B= # of bytes is the block size B (bytes per line) = 4 bytes * 8 = 32 bytes per line
S= # sets (rows)= 64 sets

C= S*E*B = 64*1*32= 2048 bytes
b=$\log_2$B bits block offset = $\log_2$32= 5
s=$\log_2$S bits for set index = $\log_2$64= 6
t = m-s-b = 64-6-5=53

tag	set index	block offset
53	6	5
All other most significant 53 bits	6 least significant bits after the block offset	Least significant 5 bits

4.3 (5 points)

What is the ratio of bits used for storing data to total bits stored in the cache in each of the above cases?

$$\frac{block\ size\ (bits)}{block\ size(bits) + tag + valid}$$

4.1:

$$\frac{32}{32+53+1} = 0.3721$$

4.2:

$$\frac{256}{256+53+1} = 0.8258$$

4.4 (5 points)

Which bits of the address would be used in the tag, index and offset in a two-way set associative cache with 1-word blocks and a total capacity of 512 words

Two way set= E (# lines per set)= 2
word= 4 bytes= 32 bits
m= # bits= 64
B= # of bytes is the block size B (bytes per line) = 4 bytes per line
S= # sets (rows)= 512/2= 256 sets

C= S*E*B = 512*2*4= 4096 bytes
b=$\log_2$B bits block offset = $\log_2$4= 2
s=$\log_2$S bits for set index = $\log_2$256= 8
t = m-s-b = 64-8-2=54

tag	set index	block offset
54	8	2
All other most significant 54 bits	8 least significant bits after the block offset	Least significant 2 bits

5 (20 points)

Given a byte-addressed memory with 12-bit addresses, and an attached four-way set associate cache with a total of 16 one-word blocks, make a table showing the final state of the cache after accessing the following memory addresses. The table should include the tags and data that will be stored in cache at the end of all insertions. Data should be shown using the corresponding addresses in main memory. Use the least-recently-used rule to evict from cache as needed. The sequence of addresses accessed is: 0x143, 0xc4a, 0x22b, 0x42f, 0x492, 0x2a2, 0x3ba, 0xb2d

Four way set= E (# lines per set)= 4
word= 4 bytes
m= # bits= 12
B= # of bytes is the block size B (bytes per line) = 4 bytes per line
S= # sets (rows)= $\frac{16}{4} = 4$ sets

C= S*E*B = 4*4*4 = 64 bytes
b=$\log_2$B bits block offset = $\log_2$4= 2
s=$\log_2$S bits for set index = $\log_2$4= 2
t = m-s-b = 12- 2-2= 8

tag	set index	block offset
8	2	2
All other most significant 8 bits	2 least significant bits after the block offset	Least significant 2 bits

0x143 → 000101000011
0xc4a → 110001001010
0x22b → 001000101011
0x42f → 010000101111
0x492 → 010010010010
0x2a2 → 001010100010
0x3ba → 001110111010
0xb2d → 101100101101

	Line 0					Line 01					Line 2					Line 3				
	Tag	Byte 00	Byte 01	Byte 10	Byte 11	Tag	Byte 00	Byte 01	Byte 10	Byte 11	Tag	Byte 00	Byte 01	Byte 10	Byte 11	Tag	Byte 00	Byte 01	Byte 10	Byte 11
Set 00	00010100	0x140	0x141	0x142	0x143	01001001	0x490	0x491	0x492	0x493	00101010	0x2a0	0x2a1	0x2a2	0x2a3					
Set 01																				
Set 10	11000100	0xc48	0xc49	0xc4a	0xc4b	00100010	0x228	0x229	0x22a	0x22b	00111011	0x3b8	0x3b9	0x3ba	0x3bb					
Set 11	01000010	0x42c	0x42d	0x42e	0x42f	10110010	0xb2c	0xb2d	0xb2e	0xb2f										