CSC/ECE 506: Architecture of Parallel Computers
Problem Set 2
Initial Submission Due: Tuesday, March 5, 2024
Reflection Due: Tuesday, March 19, 2024

There will be two submission deadlines for this problem set. For the first submission deadline, you should turn in solutions to all the problems. Then the "official" solutions will be distributed. For the second submission deadline, you should turn in corrections of your work, and explanations of anything you got wrong. The second submission is the one that will be graded.

There are 80 points on this problem set.

Problem 1. *(20 points, 4, 4, 4, & 8 per part)* For parts (a) to (c), assume the address width is 16 bits.

(a) If the block offset is 8 bits and 8 bits are tag bits, then what kind of cache mapping (direct mapped, set associative or fully associative) would it be?

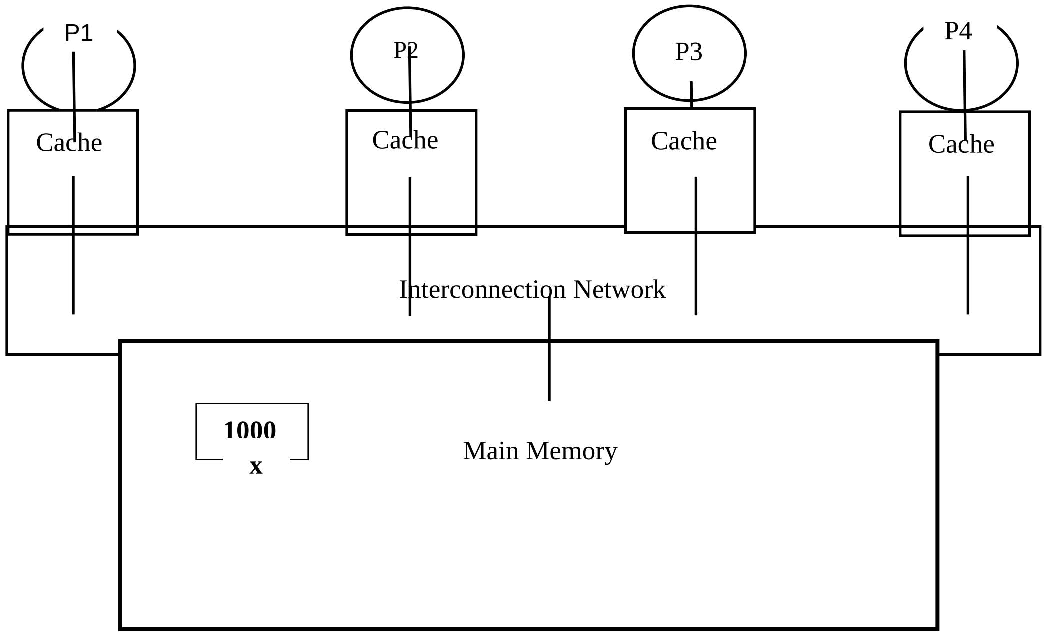
(b) If the cache size is 8 MB, and has an offset of 8 bits, with 5 index bits and the remaining bits being tag bits, then what kind of cache mapping (direct mapped, set associative or fully associative) would it be?

(c) Suppose the cache size is 8 MB, with an offset of 8 bits, 3 index bits and the remaining bits tag bits, then what kind of cache mapping (direct mapped, set associative or fully associative) would it be?

(d) To which set will a main-memory address of 9745_{16} be mapped, assuming that the cache contains 32 lines, with 16 bytes per cache block, assuming the following associativities?

(i) direct mapped
(ii) two-way set associative
(iii) four-way set associative
(iv) eight-way set associative?

Problem 2. *(20 points)* This problem relates to a system that has four processors (P1, P2, P3, and P4), each with a private L1 cache and a shared main memory. Memory location x is initialized to a decimal value of 1000. A logical arrangement of the system is shown below.



The system does not use a cache-coherence protocol.

The pseudo-assembly level code given below shows three threads run on processors P1, P2, P3 and P4 respectively.

Thread 1(P1)	Thread 2 (P2)	Thread 3 (P3)	Thread 4 (P4)
ld r2,x *// r2=x*			
ld r3,x *// r3=x*			
	ld r5, x *// r5=x*		
			ld r7, x *// r7=x*
sub r1,r2,r3 *// r1=r2-r3*			
st x, r1 *// x=r1*			
			k: **add r7,r7,100** *//r7=r7+100, k is the label.*
	ld r4, x *// r4 =x*		
		ld r6, x *// r6=x*	
			ld r8, x *// r8=x*
			bnz r8,k *//branch to k, if r8 is non zero*

(a) Fill in the following table for the above sequence of operations, assuming write-back caches are used.

Value of 'x' at different locations						Value of registers								
Action	**Cache**				**Memory**	**P1**			**P2**		**P3**	**P4**		
	P1	**P2**	**P3**	**P4**		**r1**	**r2**	**r3**	**r4**	**r5**	**r6**	**r7**	**r8**	
Initially	-	-	-	-	1000	-	-	-	-	-	-	-	-
ld r2,x													
ld r3,x													
ld r5, x													
ld r7, x													
sub r1,r2,r3													
st x, r1													
add r7,r7,100													
ld r4, x													
ld r6, x													
ld r8,x													

Control Flow Instruction	**Possible Outcome**	**Yes/No**
bnz r8,k	Will it branch to instruction 'k'	

(b) Fill in the following table for the above sequence of operations, assuming write-through caches were used.

Value of 'x' at different locations						Value of registers							
Action	Cache				Memory	P1			P2		P3	P4	
	P1	P2	P3	P4		r1	r2	r3	r4	r5	r6	r7	r8
Initially	-	-	-	-	1000	-	-	-	-	-	-	-	-
ld r2,x													
ld r3,x													
ld r5, x													
ld r7, x													
sub r1,r2,r3													
st x, r1													
add r7,r7,100													
ld r4, x													
ld r6, x													
ld r8,x													

Control Flow Instruction	Possible Outcome	Yes/No
bnz r8,k	Will it branch to instruction 'k'	

(c) Does a write-back cache provide a coherent view of the memory location x? Please explain your answer.

(d) Does using write through cache provide a coherent view of the memory location x? What improvement does it provide, when compared to write back caches? Please explain your answer.

Problem 3. *(15 points)* Here is the code for an algorithm commonly used in image-processing applications.

Consider an image *f* with width=ImageWidth and height=ImageHeight. *f* is a 2D grid of pixels. *k* is a kernel of width=2*w*+1 and height=2*h*+1 where (2*w*+1) < ImageWidth and (2*h*+1) < ImageHeight. The image *f* is processed using the kernel *k* to produce a new image *g* as shown:

```
for y = 0 to ImageHeight do
    for x = 0 to ImageWidth do
        sum = 0
        for i= -h to h do
            for j = –w to w do
                sum = sum + k[j,i] * f[x – j, y – i]
            end for
        end for
        g[x y] = sum
    end for
end for
```

(a). Identify the read-only, R/W non-conflicting and R/W conflicting variables, if the **for** *y* loop is parallelized.

Read only	R/W non-conflicting	R/W conflicting

(b). Identify the read-only, R/W non-conflicting and R/W conflicting variables, if (only) the **for** *i* loop is parallelized. Assume that the **for** *i* tasks for the previous value of *x* must complete before the **for** *i* tasks of the current value of *x* are started.

Read only	R/W non-conflicting	R/W conflicting

(c). Identify the read-only, R/W non-conflicting and R/W conflicting variables, if the **for** *i* loop is parallelized. Assume that the **for** *i* tasks for the previous value of *x* do not have to complete before the **for** *i* tasks of the current value of *x* are started.

Read only	R/W non-conflicting	R/W conflicting

Problem 4. *(25 points)* Consider a sequence of integers. A subsequence of these integers is a "zerosum" if the sum of all integers in the subsequence is zero. The task is to find the total number of zerosums for some given sequence.

For example, given the sequence

2, 1, 0, –3, 1

an example of a zerosum would be the sequence from the 1st element to the 4th element, since

2 + 1 + 0 + –3 = 0

Consider the parallel approach to finding zerosums in which task *t* tries to find zerosum subsequences starting at the *t*th element of the given sequence. The following questions relate to assigning these tasks to different processing elements.

(a) Would block assignment be a good choice for this problem? Explain.

(b) Consider cyclic assignment where there are *n*/2 processing elements to count zerosums in a sequence of *n* integers. What is the minimum number of integers a processor will need to sum? The maximum?

(c) Briefly describe one advantage and disadvantage of dynamic assignment.

(d) Why would 2D block partitioning be unsuitable for this task?