Advanced and parallel architectures

Prof. A. Massini

June 13, 2017

Part B

Student's Name

_____

***Matricola* number**

_____

Exercise 1a (3 points)	
Exercise 1b (3 points)	
Exercise 2 (8 points)	
Exercise 3 (4 points)	
Exercise 4 (3 points)	
Exercise 5 (3points)	
Exercise 6 (4 points)	
Exercise 7 (4 points)	
Total (32 points)	

Exercise 1a (3 points) – Interconnection Networks – CLOS

Design a Clos network of size 150 x 150, using in the first stage modules having 24 inputs. Consider both cases, strictly non-blocking and rearrangeable network.

Exercise 1b (3 points) – Interconnection Networks – Comparison Clos-Crossbar

Compare the cost of the crossbar 150 x 150 and the Clos network, strictly non-blocking and rearrangeable, designed in the previous point.

Exercise 2 (4+2+2 points) - GPU & CUDA

You need to write a kernel that operates on a 2D matrix of size **15000x4500**. You would like to assign one thread to each matrix element. You would like your thread blocks to use the maximum number of threads per block possible on your device, having compute capability 3.0.

a) How would you select the dimensions of a **2D grid** and **2D blocks** for your kernel? Consider the two cases of **rectangular** and **square** blocks

b) How would you select the dimensions of a **3D grid** and **2D blocks** for your kernel?

c) What is the best choice for grid and block dimensions with respect to the number of idle threads?

Technical specifications	**Compute capability (version)**									
	1.0	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**	**3.7**	**5.0**	**5.2**
Maximum dimensionality of grid of thread blocks	2				3					
Maximum x-dimension of a grid of thread blocks	65535					2^{31}-1				
Maximum y-, or z-dimension of a grid of thread blocks	65535									
Maximum dimensionality of thread block	3									
Maximum x- or y-dimension of a block	512				1024					
Maximum z-dimension of a block	64									
Maximum number of threads per block	512				1024					
Warp size	32									
Maximum number of resident blocks per multiprocessor	8					16			32	
Maximum number of resident warps per multiprocessor	24		32		48	64				
Maximum number of resident threads per multiprocessor	768		1024		1536	2048				
Technical specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**	**3.7**	**5.0**	**5.2**
	Compute capability (version)									

Exercise 3 (4 points) - Cache coherence

Consider a multicore multiprocessor implemented as a symmetric shared-memory architecture, as illustrated in the figure.

Each processor has a single, private cache with coherence maintained using the snooping coherence protocol. Each cache is direct-mapped, with four blocks each holding two words. The coherence states are denoted M, S, and I (Modified, Shared, and Invalid).

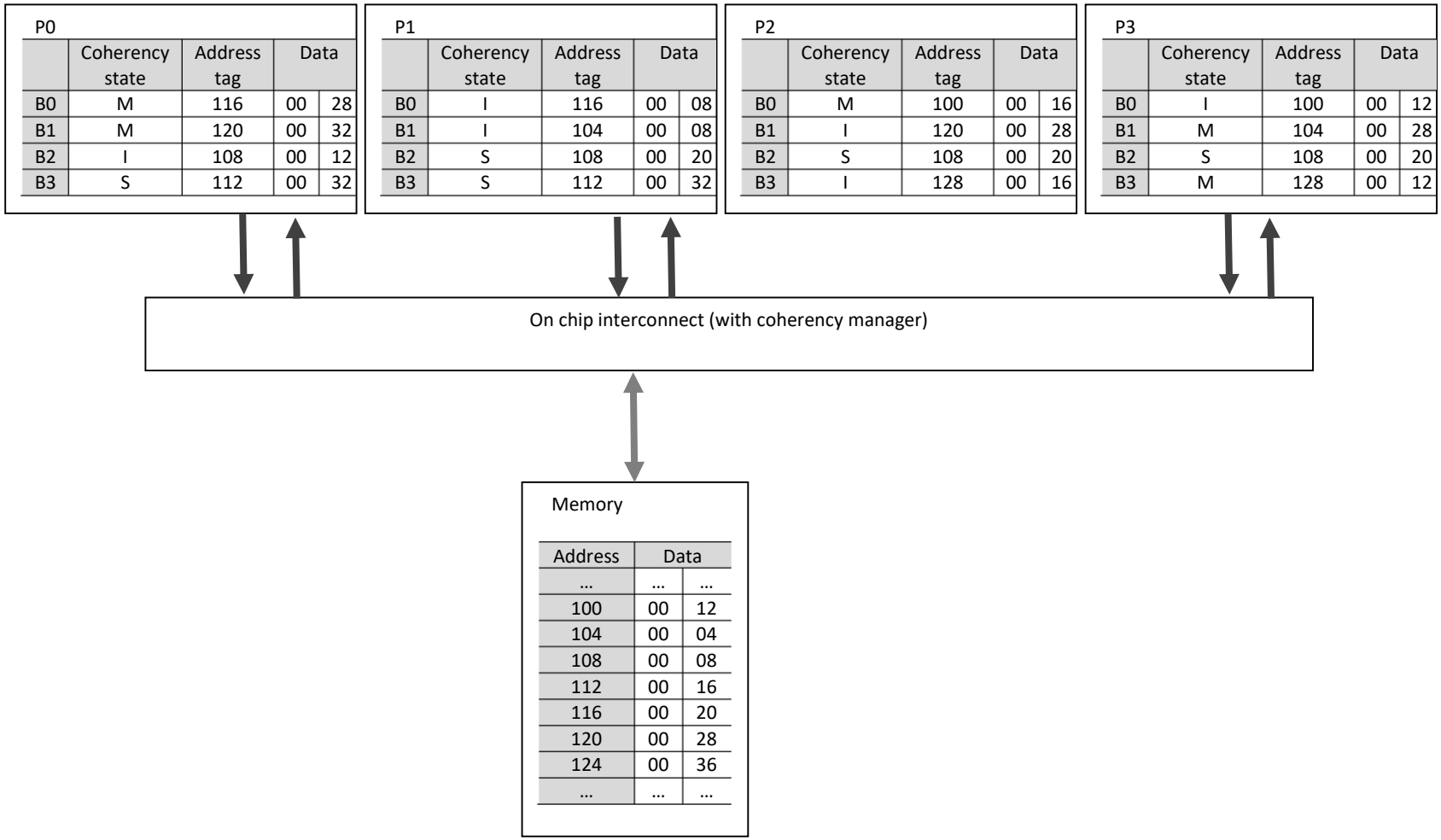
Each part of this exercise specifies a sequence of one or more CPU operations of the form:

P#: <op> <address> [<value>]

where P# designates the CPU (e.g., P0), <op> is the CPU operation (e.g., read or write), <address> denotes the memory address, and <value> indicates the new word to be assigned on a write operation.

For each part of this exercise, assume the initial cache and memory state as illustrated in the figure.

Show in the table for the results:
- miss/hit
- the coherence state before the action
- the CPU processor Pi and cache block Bj
- the changed state (i.e., coherence state, tags, and data) of the caches and memory after the given action.



P0	Coherency state	Address tag	Data	
B0	M	116	00	28
B1	M	120	00	32
B2	I	108	00	12
B3	S	112	00	32

P1	Coherency state	Address tag	Data	
B0	I	116	00	08
B1	I	104	00	08
B2	S	108	00	20
B3	S	112	00	32

P2	Coherency state	Address tag	Data	
B0	M	100	00	16
B1	I	120	00	28
B2	S	108	00	20
B3	I	128	00	16

P3	Coherency state	Address tag	Data	
B0	I	100	00	12
B1	M	104	00	28
B2	S	108	00	20
B3	M	128	00	12

Memory	Address	Data	
	...	...	...
	100	00	12
	104	00	04
	108	00	08
	112	00	16
	116	00	20
	120	00	28
	124	00	36
	...	...	...

Specify the value returned by a read operation.

a) P2: write 112 ← 28

Comments

hit/miss	state before	Pi.Bj (state, tag, datawords)

b) P1: write 124 ← 24

hit/miss	state before	Pi.Bj (state, tag, datawords)

c) P3: read 120

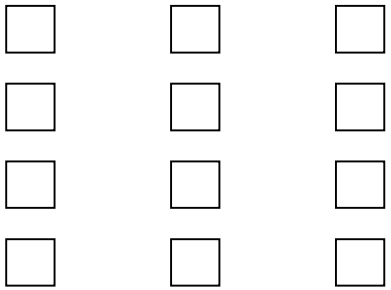
hit/miss	state before	Pi.Bj (state, tag, datawords)

d) P0: write 100 ← 32

hit/miss	state before	Pi.Bj (state, tag, datawords)

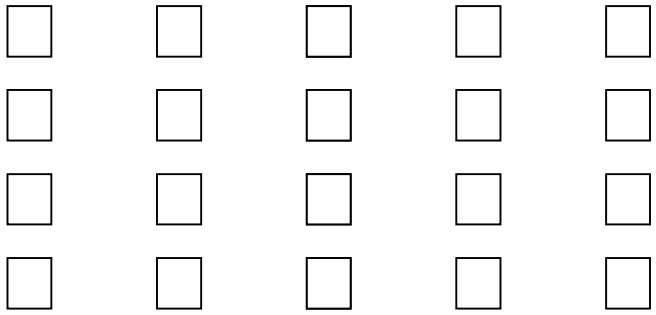
Exercise 4 (3 points) – Interconnection networks – log N MIN

Complete the scheme Baseline networks of size N=8. Show if it can realize the permutation $P = \begin{pmatrix} 01\ 23\ 45\ 67 \\ 40\ 36\ 71\ 52 \end{pmatrix}$ showing the switch setting obtained using the self-routing.



Exercise 5 (3 points) – Interconnection networks – (2 log N - 1) MIN

Complete the scheme of the Butterfly-Butterfly^{-1}. Show the switch setting to realize permutation $P = \begin{pmatrix} 01\ 23\ 45\ 67 \\ 40\ 36\ 71\ 52 \end{pmatrix}$ according to the looping algorithm.



Exercises 6 (4 points) Amdhal Law

The following measurements are recorded with respect to the different instruction classes for the instruction set running a given set of benchmark programs:

Instruction Type	Instruction Count (millions)	Cycles per Instruction
Arithmetic and logic	6	4
Load and store	8	2
Branch	4	4
Others	6	4

Assume that "*Arithmetic and logic*" instructions can be modified so that they take 3 cycles per instruction instead of 4, and "*Branch*" instructions can be modified so that they take 2 cycle per instruction instead of 4 as in the table. Compute the speedup obtained by introducing only one enhancement and both enhancements using the **Amdhal law**.

Exercises 7 (4 points) Performance equation

Suppose we have made the following measurements, where we are considering Arithmetic instructions and FP (Floating Point):

Frequency of Arithmetic operations = 35%
Average CPI of Arithmetic operations = 4.0
Average CPI of other instructions = 2.5

Frequency of FP operations = 15%
CPI of FP = 8.0

Assume that the two design alternatives are to decrease the CPI of FP to 3.0 or to decrease the average CPI of Arithmetic operations to 2.5. Compare these two design alternatives using the processor **performance equation**, and compute the speedup in both cases.