Department of informatics
Computer Science degree
Computer Architecture

June 25th, 2015

Rules:

- Read the exam carefully before starting
- Mobile phones, electronic devices, books and notes are not allowed.
- Mobile phones have to be completely disconnected
- Write the exam using a pen (not a pencil)
- **Exam duration is 2 hours 30 minutes**
- **Use a different page for each exercise. If you don't answer one question, use a white page with the exercise number and your name.**

NAME:
SURNAME:
NIA:

Exercise 1 [10 points]:

Given a high resolution image processing application that is partially parallel (a part is parallel and another part is sequential). We assume that there is no limit in the number of processes that can be executed.

The objective is to obtain and overall speedup value of 10 when running a given number of processes.

You are asked to obtain the relationship between the value of the percentage of code that is parallel and the parallelism degree (number of processes used) when this speedup is reached.

Exercise 2 [20 points]:

Given a MIPS processor with a pipeline that has two separate register files (one for integers and the other one for floating point numbers). The integer register bank has 32 registers. The floating-point register bank has 16 double-precision registers ($f0, $f2, …, $f30).

We assume that we have enough fetch and decode bandwidth to execute one instruction per cycle without stalls (with the exception of stalls associated to data dependencies).

The following table shows the extra latencies related to some types of instructions. These latencies have to be considered when there are data dependencies. When there are no dependencies, these extra latencies are not applicable.

Instruction	Extra latency	Operation
ldc1	+2	Load a 64 bits value in a floating point register
sdc1	+2	Stored a 64 bits value in main memory
add.d	+4	Add floating point double precision registers
mul.d	+6	Multiply floating point double precision registers
addi	+0	Add a value to a integer registry
subi	+0	Subtract a value to a integer registry
bnez	+1	Branch if the register value is not zero.

The instruction bnez uses a delayed branch with one delay slot.

We intend to execute the following code in the previous architecture:

```
loop:       ldc1 $f0, ($t0)
            ldc1 $f2, ($t1)
            add.d $f4, $f0, $f2
            mul.d $f4, $f4, $f6
            sdc1 $f4, ($t2)
            addi $t0, $t0, 8
            addi $t1, $t1, 8
            subi $t3, $t3, 1
            bnez $t3, bucle
            addi $t2, $t2, 8
```

The initial register values are:

- $t0: 0x00100000
- $t1: 0x00140000
- $t2: 0x00180000
- $t3: 0x00000100

Complete the following tasks:

a) Enumerate the RAW dependencies related to the previous code.
b) Show all the stalls that are produced when one single code iteration is being executed. Show the overall number of cycles per iteration.
c) Schedule the loop instructions in order to reduce the number of stalls.

d) Unroll the loop in the following way: each unrolled iteration processes four array positions. Obtain the resulting speedup. Note: use real register names ($f0, $f2, …, $f30).

IMPORTANT: The solutions that do not use real existing registers (eg: $f2' or $f2'') will **not** be considered valid.

Exercise 3 [10 points]

Let's assume that we have a computer with 1 clock cycle per instruction (CPI) when all the memory accesses are cache hits. The only data accesses are load and store instructions that represent 25% of the overall number of instructions. The cache miss penalty is 50 clock cycles and the miss rate is 5%.

Obtain the speedup when comparing the following two scenarios: (1) no cache misses (all hits) and (2) there are cache misses (with the previous miss rate).

Exercise 4 [15 points]:

Given the following definition of a lock-free stack:

```
template<typename T>
class stack {
private:
        struct node {
                std::shared_ptr<T> data;
                node* next;
                node(T const& data_):data(new T(data_)){}
        };
        std::atomic<node*> head;
public:
        void push(T const& data);
        std::shared_ptr<T> pop()
};
```

a) Provide a lock-free implementation of the push and pop functions. Note: ignore the memory leak problem.
b) Briefly explain how could the memory leak problem (related to previous point) be avoided.

Exercise 5 [15 points]:

Given the following OpenMP-based program:

```
double calcula_pi(double step) {
  int i;
  double x, sum = 0.0;
  #pragma omp parallel for reduction(+: sum) private(x)
  for (i=0;i<1000000;++i) {
    x = (i-0.5) * step;
    sum += 2.0 / (1.0 + x*x);
  }
  return step * sum;
}
```

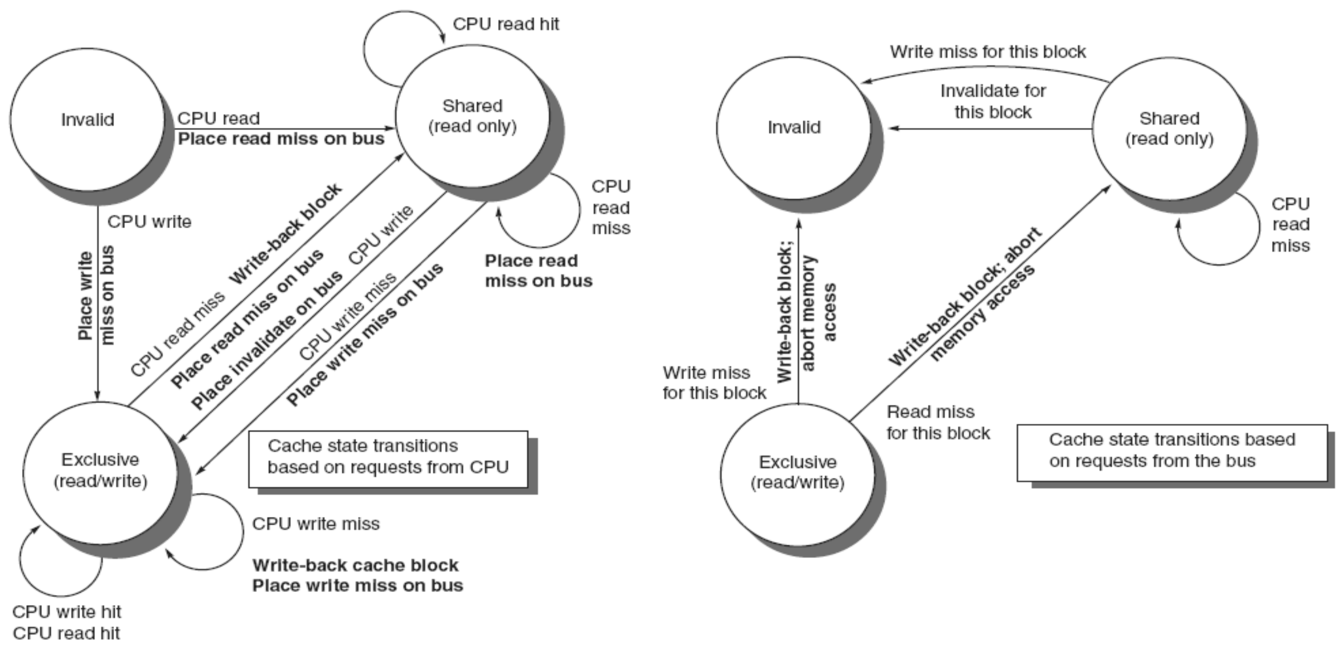
Write an equivalent OpenMP program that does not perform reductions.

Exercise 6 [10 points]:

Given a computer with two processors, each one with its private cache memory using a write-back policy on the main memory. Initially, all cache entries are invalid. The memory address related to variable **A** is initially set to 255.

This computer executes the following operation sequence:

Time	Processor 1	Processor 2
1	lw $t0, A	
2		lw $t1, A
3	sw $zero, A	
4		lw $t2, A



Show for each one of these operations the state and value of the memory and cache memories related to the memory address A.

Time	Cache 1 state	Cache 1 value	Cache 2 state	Cache 2 value	Memory value
0	I	--	I	--	255
1					
2					
3					
4					