Department of Computer Science and Engineering
Bachelor in Computer Science and Engineering
Computer Architecture

Exam

ATTENTION:

- **Time limit is 150 minutes.**

NAME:
FAMILY NAME:
NIA:

Exercise 1 [3 points]:

A computer has an L1 data cache which is 2 ways set-associative and with a size of 32KB and an L1 instructions cache with the same characteristics. It also has an L2 unified cache which is 4-ways set-associative with a size of 256 KB. In both cases the line size is 64 bytes. It is assumed that hit in L1 cache requires 3 cycles and a hit in L2 cache requires 9 additional cycles and a penalty for bringing a block from main memory to L2 cache of 60 cycles. All caches have a write-back policy.

On this machine we have the following code fragment:

```
struct body {
  double x, y, z;
  double vx, vy, vz;
  double ax, ay, az;
  double m;
};
void f(std::vector<body> & v, double dt) {
  const size_t sz = v.size();
  for (int i=0; i< sz; ++i) {
    v[i].vx += v[i].ax * dt;
    v[i].vy += v[i].ay * dt;
    v[i].vz += v[i].az * dt;
    v[i].x += v[i].vx * dt;
    v[i].y += v[i].vy * dt;
    v[i].z += v[i].vz * dt;
  }
}
```

1.1 [0.5 points]: Determine which should be the average access time for data assuming the following miss rates:

- L1 data: 3%
- L2: 1%

1.2 [1.5 points]: Determine the number of misses in data access during the execution of the loop in function f() for cache L1 (initially empty) if vector size is 64 elements. Assume that variables I and sz are allocated to registers and do not generate data accesses. The data block from vector v is assumed to be aligned at 64 bytes limit.

1.3 [1 point]: Propose a modification in code that uses parallel arrays (i.e., eliminating the array merge and using an independent array for each field in the original structure) instead of array merge and determine which would be the new miss rate. Assume that all data blocks from arrays are stored consecutive in memory.

SOLUTION:

1.1:

Ta = th(L1) + m(L1) * (th(L2) + m(L2) * tp(L2))

Ta = 3 + 0.03 * (9 + 0.01 * 60) = 3 + 0.03 9.6 = 3,228

1.2:

Each element in the array has 10 values of 8 bytes, using in total 80 bytes. Each cache line uses 64 bytes. This makes that elements in the array do not perfectly match in cache lines. The least common multiple fo 80 and 64 is 320. Thus, a match happens for every 320 bytes, which is equivalent to 5 cache lines or 4 array locations.

L1 cache has 32 KB. The size of a way is 16 KB. As the size of a line is 64 B, the number of lines per way is $2^{14}/ 2^6 = 2^8$.

As very element in the vector needs 80 bytes, the number of elements that a way can hold is $2^{14} /(2^5*5) =2^9/5 > 102$. Consequently, the 64 elements in the vector can fit in a single way of L1 cache.

To analyze the number of misses and hits, we will analyze accesses to the first 4 locations in the array, as those results can be extrapolated to the rest. Its memory layout is:

L0: v[0].x, v[0].y, v[0].z, v[0].vx, v[0].vy, v[0].vz, v[0].ax, v[0].ay

L1: v[0].az, v[0].m, v[1].x, v[1].y, v[1].z, v[1].vx, v[1].vy, v[1].vz

L2: v[1].ax, v[1].ay, v[1].az, v[1].m, v[2].x, v[2].y, v[2].z, v[2].vx

L3: v[2].vy, v[2].vz, v[2].ax, v[2].ay, v[2].az, v[2].m, v[3].x, v[3].y

L4: v[3].z, v[3].vx, v[3].vy, v[3].vz, v[3].ax, v[3].ay, v[3].az, v[3].m

In each iteration there is a total of 18 memory accesses (two reads and one write per sentence, as operator += is used). In four iterations, there is a total of 4*18=72 memory accesses. From them, 4 are cache misses. In total, there are 5 * 64 / 4 = 80 misses.

Thus, the cache miss rate is 5/72 = 0,06944 -> 6,94%

1.3

Using parallel arrays we have:

```
void f(std::vector<double> & x, std::vector<double> & y, std::vector<double> & z,
   std::vector<double> & vx, std::vector<double> & vy, std::vector<double> & vz,
   std::vector<double> & ax, std::vector<double> & vy, std::vector<double> & vz,
   double dt) {
  const size_t sz =x.size();
  for (int i=0; i<sz; ++i) {
    vx[i] += ax[i] * dt;
    vy[i] += ay[i] * dt;
    vz[i] += az[i] * dt;
    x[i] += vx[i] * dt;
    y[i] += vy[i] * dt;
    z[i] += vz[i] * dt;
  }
}
```

Each of the nine arrays (array m is not used) need 8 * 64 bytes = 2^3*2^6 bytes = 2^9 bytes, equivalent to 8 consecutive cache lines. The total number of lines used by all arrays is 8*9 = 72, which is much lower thatn the 256 cache lines per way in L1

In this case, it is necessary to analyze what happens in 8 iterations, as 8 elements fit in one line.

Iteration 0: 9 misses (one per array) and 9 hits.

Iterations 1 to 7: 18 hits per iteration.

In total, for 8 iterations, we have 9 misses and 135 hits (9+18*7), and the miss rate is 9/144 = 0.0625 -> 6,25%

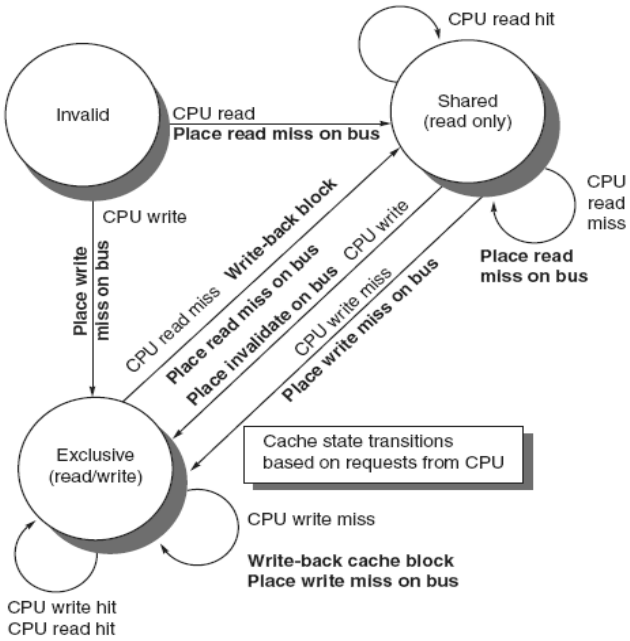
Universidad Carlos III de Madrid

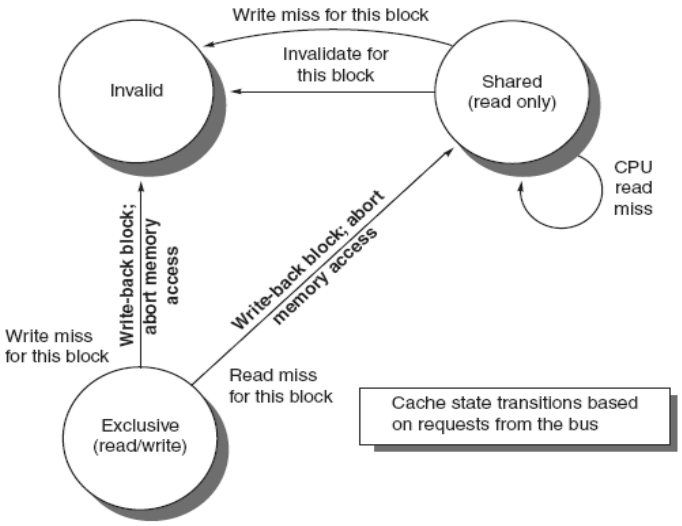
Department of Computer Science and Engineering
Bachelor in Computer Science and Engineering
Computer Architecture

ARCOS

Exam

Exercise 2 [2.5 points]: A processor with three cores and symmetric shared memory architecture uses a bus with snooping protocol. Each core has a single level of cache memory which is private for that core and whose coherence is kept by using a MSI protocol. Cache memories use direct mapping and each cache line stores two words.





The following scenarios show different execution situations of different threads in each core. Assume that:

- Each scenario is independent and for each one all cache memories are initially empty.
- Vectors a and b have two entries, are memory aligned storing a single block each one.
- Due to mapping policy, vectors a and b are stored in the same cache line.
- You are asked to specify for each scenario the contents of cache memories from each core across time following the format in the provided table (make one for each scenario). **Besides, you must justify your answers.**

Time	Core 1		Core 2		Core 3		Bus Traffic
	State	Content (a or b)	State	Content (a or b)	State	Content (a or b)	

Universidad Carlos III de Madrid

Department of Computer Science and Engineering
Bachelor in Computer Science and Engineering
Computer Architecture

Exam

- **Scenario 1:**

Time	Thread 1 (running in core 1)	Thread 2 (running in core 2)	Thread 3 (running in core 3)
0	Read a[0]		
1		Read a[0]	
2			Write a[0]

- **Scenario 2:**

Time	Thread 1 (running in core 1)	Thread 2 (running in core 2)	Thread 3 (running in core 3)
0		Write a[0]	
1			Write a[0]
2			Write a[0]
3			Read a[0]

- **Scenario 3:**

Time	Thread 1 (running in core 1)	Thread 2 (running in core 2)	Thread 3 (running in core 3)
0		Read a[0]	
1			Read a[0]
2	Write a[1]		
3			Read a[0]

- **Scenario 4:**

Time	Thread 1 (running in core 1)	Thread 2 (running in core 2)	Thread 3 (running in core 3)
0		Write a[0]	
1		Read b[0]	
2			Write a[0]
3		Write b[0]	
4		Write a[0]	

Solution:

Scenario 1

Time	Core 1		Core 2		Core 3		Bus traffic
	State	Content (a or b)	State	Content (a or b)	State	Content (a or b)	
	S	a					Read miss Send a
	S	a	S	a			Read miss Send a
	I	a	I	a	E	a	Write miss Send a

Universidad Carlos III de Madrid

Department of Computer Science and Engineering
Bachelor in Computer Science and Engineering
Computer Architecture

Exam

Scenario 2

Time	Core 1		Core 2		Core 3		Bus traffic
	State	Content (a or b)	State	Content (a or b)	State	Content (a or b)	
			E	a			Write miss Send a
			I	a	E	a	Write miss Write back a
			I	a	E	a	
			I	a	E	a	

Scenario 3

Time	Core 1		Core 2		Core 3		Bus traffic
	State	Content (a or b)	State	Content (a or b)	State	Content (a or b)	
			S	a			Read miss Send a
			S	a	S	a	Read miss Send a
	E	a	I	a	I	a	Write miss Send a
	S	a	I	a	S	a	Read miss Write-back a

Scenario 4

Time	Core 1		Core 2		Core 3		Bus traffic
	State	Content (a or b)	State	Content (a or b)	State	Content (a or b)	
0			E	a			Write miss a Send a
1			S	b			Read miss b Write-back a Send b
2			S	b	E	a	Write miss a Send a
3			E	b	E	a	Invalidate
4			E	a	I	a	Write back b Write miss a Write back a

Universidad Carlos III de Madrid

Department of Computer Science and Engineering
Bachelor in Computer Science and Engineering
Computer Architecture

ARCOS

Exam

Exercise 3 [1 point]: Given the following code, answer the questions below:

```
01  void exec(int a){
02    /*
03      * This code needs "a" seconds of CPU to be executed
04      */
05  }
06
07  int main(){
08
09    #pragma omp parallel for
10    for(int i = 0; i < 100; i++){
11              exec(i);
12    }
13
14    return 0;
15  }
```

3.1 [0.5 points]: What is done in line 9? Consider that no environment variable related to OPenMP has be set in the session where the program is running.

3.2 [0.5 points]: Which is the best OpenMP scheduler for directive in line 9 using a chunk size of 10? Justify your answer comparing the behavior of all possible schedulers.

SOLUCIÓN

a) Directive at line 9 performs two operations:

1.- Generates as many threads as cores (real or virtual) are available in the machine executing. A join is performed upon loop termination.

2.- Splits the loop among the created threads. As no scheduler is specified, by default the static scheduler is selected and iterations are equitatively split among threads.

b) As an exmple a 4 threads execution:

As the workload increases as the iteration index increases, the worst scheduler will be the "static" without chunk sizes as it will give 25 iterations to each thread and the last 25 iterations require much more time that the first 25. The estimated time for 4 threads for the last 25 iterations is: 99 + 98 + 97 + 96 … + 76 + 75 = 2175.

Dynamic and static schedulers with a chunk size of 10 will behave similarly. Dynamic will give 10 iteration to each thread, and when they finish, will give them another 10. Static will give iterations also in chunks of 10 with a similar order to dynamic. A possible sharing will be:

Thread 0	0 to 9	36		
	40 to 49	435		
	80 to 89	835	1306	Seconds
Thread 1	10 to 19	135		
	50 to 59	535		
	90 to 99	935	1605	Seconds
Thread 2	20 to 29	235		
	60 to 69	635		
			870	seconds
Thread 3	30 to 39	335		
	70 to 79	735		
			1070	Seconds

Consequently the total time would go down to 1605 seconds.

The last case would be the “guided” scheduler. In this case, the number of iterations is adjusted as the program advances to adapt to the remaining workload. Thus, at the end of execution, when sharing has a higher impact, the grain will be finer getting a better sharing. This will approximate the best possible scheduling.

Exercise 4 [1.5 points]: Given the following code, answer the questions below:

```
01  #include <iostream>
02  #include <iomanip>
03  #include <thread>
04  #include <vector>
05
06  double sum;
07  double step;
08  int n_threads;
09
10  void calculate_steps(int thread_id, long nsteps) {
11      for (int i=thread_id; i<nsteps; i=i+n_threads) {
12        double x = (i+0.5) * step;
13        sum += 4.0 / (1.0 + x * x);
14      }
15  }
16
17  int main(){
18      using namespace std;
19
20      // Init global variables
21      n_threads = 4;
22      long nsteps = 100000000;
23      step = 1.0 / double(nsteps);
24      sum = 0.0;
25      std::vector<std::thread> threads;
26
27      // Create threads
28      for(int i = 0; i < n_threads; i++){
29        threads.push_back(std::thread(calculate_steps,i,nsteps));
30      }
31      for (int i = 0; i < n_threads; ++i) {
32         threads[i].join();
33      }
34
35      // Calculate reduction
36      double pi = step * sum;
37      cout << "PI= " << setprecision(10) << pi << endl;
38
39      return 0;
40  }
```

4.1 [0.25 points]: Specify the lines of code composing a critical section. Justify your answer.

4.2 [0.25 points]: Propose a lock based solution to the critical section. Justify your answer.

4.3 [0.5 points]: Propose a lock-free solution to the critical section. Justify your answer.

4.4 [0.5 points]: If the program is run in a 2-core machine. Which from the following solutions should have a better performances using two threads (n_threads = 2). And using 32 threads (n_threads=32). Justify both answers.

Universidad Carlos III de Madrid

Department of Computer Science and Engineering
Bachelor in Computer Science and Engineering
Computer Architecture

Exam

SOLUCION

4.1.- The only conflictive part in the code is in oine 12, when all threads try to modify concurrently the shared variable sum.

4.2

```
#include <iostream>
#include <iomanip>
#include <thread>
#include <vector>
#include <mutex>

double sum;
double step;
int n_threads;
std::mutex mtx;


void calculate_steps(int thread_id, long nsteps) {
  for (int i=thread_id; i<nsteps; i=i+n_threads) {
    double x = (i+0.5) * step;
    std::unique_lock<std::mutex> l{mtx};
    sum += 4.0 / (1.0 + x * x);
  }
}

int main(){
  using namespace std;

  // Init global variables
  n_threads = 16;
  long nsteps = 10000000;
  step = 1.0 / double(nsteps);
  sum = 0.0;
  std::vector<std::thread> threads;

  // Create threads
  for(int i = 0; i < n_threads; i++){
    threads.push_back(std::thread(calculate_steps,i,nsteps));
  }
  for (int i = 0; i < n_threads; ++i) {
    threads[i].join();
  }
```

```
  // Calculate reduction
  double pi = step * sum;
  cout << "PI= " << setprecision(10) << pi << endl;

  return 0;
}
```

4.3.-

```
#include <iostream>
#include <iomanip>
#include <thread>
#include <vector>
#include <atomic>

double sum;
double step;
int n_threads;
std::atomic_flag f;

void calculate_steps(int thread_id, long nsteps) {
   for (int i=thread_id; i<nsteps; i=i+n_threads) {
      double x = (i+0.5) * step;
      while(f.test_and_set());
      sum += 4.0 / (1.0 + x * x);
      f.clear();
   }
}

int main(){
   using namespace std;

   // Init global variables
   n_threads = 4;
   long nsteps = 100000000;
   step = 1.0 / double(nsteps);
   sum = 0.0;
   std::vector<std::thread> threads;

   // Create threads
   for(int i = 0; i < n_threads; i++){
      threads.push_back(std::thread(calculate_steps,i,nsteps));
   }
   for (int i = 0; i < n_threads; ++i) {
```

```
        threads[i].join();
    }

    // Calculate reduction
    double pi = step * sum;
    cout << "PI= " << setprecision(10) << pi << endl;

    return 0;
}
```

4.4.- A valid answer is that atomics perform better if there are enough available resources to perform busy waiting (two threads and two cores), but mutes will behave better when there are more threads performing waiting.

A mutex implies invocation of operating system calls while threads make use of hardware support.