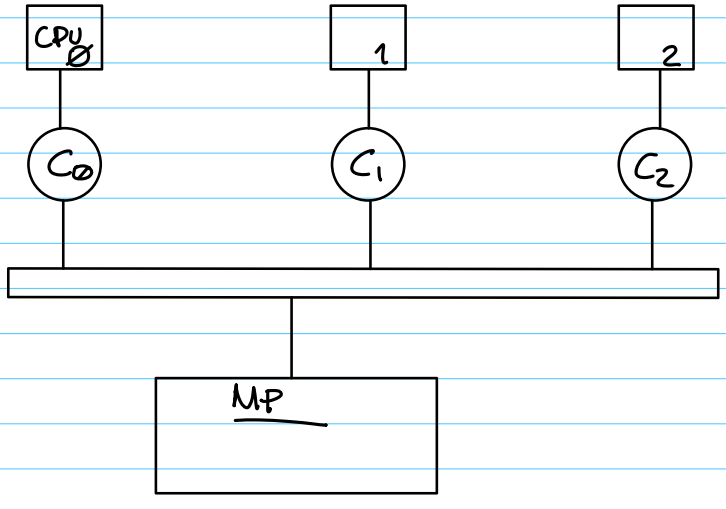
P1

1. Given an SMP system with 3 CPUs, each with its own cache memory initially empty. To keep caches coherent the system uses a Snoopy–based write-invalidate *MSI* coherence protocol. Assuming the following sequence of memory instructions all accessing the same memory direction: `r1, w1, r2, w3, r2, w1, w2, r3, r2, r1` (where `rx` indicates read by processor `x` and `wy` write by processor `y`), fill in the table indicating including the CPU event (PrRd, PrWr), Bus transactions (BusRd, BusRdX, BusUpgr, Flush) and state of the cache line (M, S, I) in each cache memory after each access to memory. In the observations column indicate who is providing the line when a cache requires it and when main memory is updated.

   What would change in the table if a *MESI* protocol is used instead of the original *MSI*?

Memory Access	CPU event	hit/miss	Bus transaction(s)	Cache Line State			Observations
				Cache1	Cache2	Cache3	
r1	PrRd	miss	BusRd	shared	—	—	
w1	PrWr	Hit	Bus Upgr	M	—	—	
r2	PrRd	miss	Bus Rd/F1	shared	shared	—	
w3	PrWr	miss	Bus RdX	I	I	M	
r2	PrRd	miss	Bus Rd/F3	I	S	S	
w1	PrWr	miss	Bus RdX	M	I	I	
w2	PrWr	miss	Bus RdX/F1	I	M	I	
r3	PrRd	miss	Bus Rd/F2	I	S	S	
r2	PrRd	hit	—	I	S	S	
r1	PrRd	miss	BusRd	S	S	S	los BOOS <3



*cada vez que hacemos un flush updateamos mem.

3.-

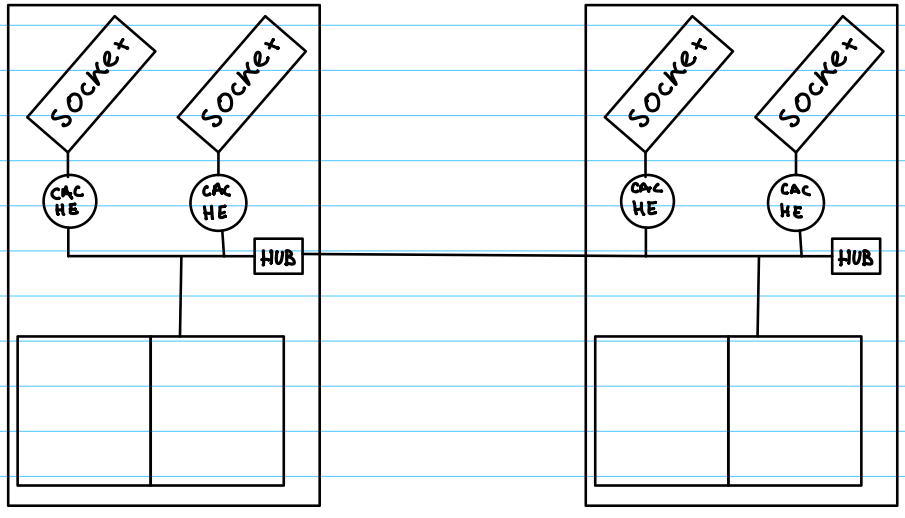
3. Assume a NUMA system with two NUMAnodes, two sockets per NUMAnode, six cores per socket, $24GB$ of main memory and a shared cache memory of $12MB$ (cache line size of 64 bytes) per socket. Data coherence is maintained using Write-Invalidate, Snoopy with MSI Protocol within each NUMAnode and using a Directory-based cache coherency protocol among NUMAnodes.

   Answer the following questions:

(a) How many bits are needed to maintain the coherence at each shared cache memory? What are their function/s? Please, give the number of bits per cache line and the total for each cache.

(b) How many bits are needed to maintain the coherence at the directory structures? What are their function/s? Please, give the number of bits per memory line and the total number of bits per directory.

(c) The MESI protocol adds a new state to the MSI protocol. Explain which is the objective of this new state and how it helps to improve the performance of the MSI protocol. Justify your answer using an example, showing the protocol CPU events and bus transactions with MSI and MESI. Do you need any additional bits in the coherence structures of MSI to include this new state of the MESI protocol?



a) 2 bits → estat
~~1 bit → dirty~~
} 64B + 2 bits

b) 2 bit → estado → 64GB / 64B
2 bit → presencia

c) When in modified state you write. No additional bits needed.

	MESI	MSI
r0	PrRd → BusRd → E	PrRd → BusRd → S
r1	PrWr → — → E → M	PrWr → BusUpgr →

6. 6. Given the following code excerpt including OpenMP directives:

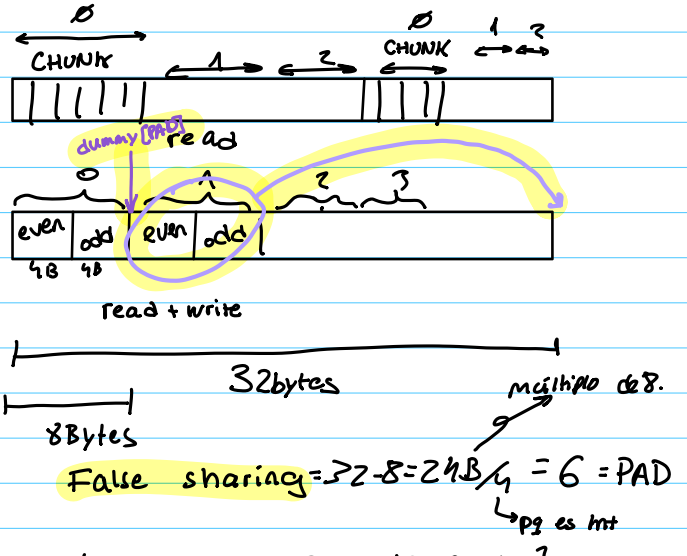
```
int vector[N];
typedef struct {
    int even = 0;
    int odd = 0;
    int dummy[PAD];
} count[NUM_THREADS];

...

#pragma omp parallel
{
   int id = omp_get_thread_num();
   int nt = omp_get_num_threads();
   for (int ii=id*CHUNK; ii < N; ii += nt*CHUNK)
       for (int i=ii; i < min(N, ii+CHUNK); i++)
           if (vector[i]%2) contar[id].odd++;
           else contar[id].even++;
}
```

in which each implicit task executes groups of consecutive `CHUNK` iterations in a round robin (cyclic) way. Assuming that each integer (`int`) variable occupies 4 bytes, a cache line occupies 32 bytes, and that the initial address of `vector` and `count` are aligned with the start of a cache line, we ask:

(a) Compute the minimum value for constant `PAD` in the previous program in order to avoid *false sharing* during the execution of the parallel loop.

(b) Compute the minimum value for constant `CHUNK` in order to improve spatial locality, within each thread, when reading the elements of `vector`.

int → 4B
línea cache → 32B
¿CHUNK?



b) ¿cuántos elementos me caben en una línea de cache?
→ Múltiplos de 8 para completar

5:-

5. Assume a multiprocessor system composed of two NUMA nodes (labeled *NUMAnode0* and *NUMAnode1*), each with 8 GB of main memory shared by two sockets (labeled *socket0* and *socket1* for *NUMAnode0*, and *socket2* and *socket3* for *NUMAnode1*). Each socket labeled *socketX* has one core (labeled *coreX*) with a cache memory of 8 MB. **The cache line size is 16 bytes**. Data coherence in the system is maintained using **Write-Invalidate MSI protocols**, with a **Snoopy attached to each cache memory** to provide coherency within each NUMA node and **directory-based coherence among the two NUMA nodes**.

   Given the following C code:

```
#define N 16
int x[N];
...
#pragma omp parallel num_threads(4)
{
    int myid = omp_get_thread_num();
    int nths = omp_get_num_threads();
    int i_start = (N / nths) * myid;
    int i_end = i_start + N/nths;
    // FOR loop
    for (int i=i_start; i<i_end; i++) x[i]=init();
}
```

   and assuming that: 1) the Operating System decides the data allocation using a "first touch" policy at the memory page level and that each memory page contains a single memory line; 2) vector `x` is the only variable that will be stored in memory (the rest of variables will be all in registers of the processors); 3) the initial address of vector `x` is aligned with the start of a cache line; 4) the size of an `int` data type is 4 bytes; and 5) $thread_i$ always executes on $core_i$, where $i = [0 - 3]$, **we ask you:**

   (a) Compute the amount of bits taken by each snoopy to maintain the coherence between caches inside a NUMA node and, the amount of bits in each node directory to maintain the coherence among NUMA nodes.

   (b) Draw a picture that shows vector `x` and how many memory lines are necessary to store its elements, identifying the range of elements per memory line.

   (c) Assuming that all cache memories are empty at the beginning of the program, fill in the following trable with the information corresponding to each range of elements allocated per memory line once all threads arrive to the end of the parallel region: vector range, the Home node number, the presence bits, main memory line state (`State in MM`) corresponding to accesses to vector `x`, and the state of any copy (`State in cache socket0-3` in the table) of those memory blocks in one or more caches of sockets 0 to 3.

Vector x range	# Home NUMA node	Presence bits	State in MM	State in cache			
				socket0	socket1	socket2	socket3
0...3	Ø	Ø 1	M	M	-	—	—
4...7	Ø	Ø 1	M	-	M	—	-
8...11	1	1 Ø	M	—	-	M	—
12...15	1	1 Ø	M	-	-	—	M

(d) Assuming the final previous state of the multiprocessor system with the presence bits and state for each cache and memory line, fill in the following table with the sequence of processor commands (Core), bus transactions within NUMA nodes (Snoopy), transactions between NUMA nodes (Directory), the presence bits, state for each cache and memory line, to keep cache coherence, **AFTER the execution of each** of the following sequence of commands:

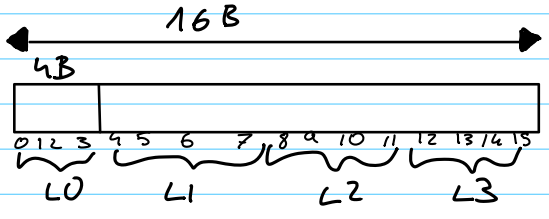
i. $core_2$ reads the contents of `x[2]`
ii. $core_2$ writes the contents of `x[2]`
iii. $core_1$ reads the contents of `x[0]`

Command	Coherence actions			Presence bits	State in MM	State in cache			
	Core	Snoopy	Directory			socket0	socket1	socket2	socket3
$core_2$ reads x[2]	PrRd	BusRd, BusRdo/Flush, Flush,	RdReq 1→0 DReply 0→1	1 1	S	S	–	S	–
$core_2$ writes x[2]	WrRd			1 0	M	. .		.	. .
$core_1$ reads x[0]	PrRd								

(e) Given a new code where we are using only two threads and the iterations are executed by the threads in an interleaved way:

```
#pragma omp parallel num_threads(2)
{
   int myid = omp_get_thread_num();
   int nths = omp_get_num_threads();

   // FOR loop
   for (int i = myid; i < N; i += nths) x[i] = init();
}
```

Assuming that threads 0 and 1 alternately execute the first 4 iterations in the time. What is the number of BusRdx that will be produced? Why is this happening?



1 thread x línea