

PAR – 2nd In-Term Exam – Course 2018/19-Q2

May 29th, 2019

Problem 1 (4 points) Given the following incomplete OpenMP implementation of a code:

```
int A[MAX_HASH]; // hash table
int B[N];         // input vector
int sum = 0;
...
#pragma omp parallel
{
    ...
    for(int i=0; i<N; i++)
        A[hash(B[i])]++;
    ...
    for(int i=0; i<MAX_HASH; i++)
        sum += foo(A[i]);
    ...
}
...
```

Assuming that we want to implement a **geometric data decomposition** that should:

- Avoid the need of synchronisation during the execution of the **first loop**.
- Allow the exploitation of temporal data locality **between the first and second loop**.
- Avoid any synchronisation and false sharing situation during the execution of the **second loop**.
- Minimise the load unbalance (maximum unbalance of one element) among threads (processors).

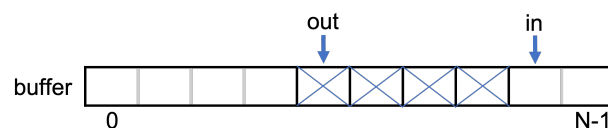
The implementation of the data decomposition cannot make use of the `#pragma omp for` construct neither fuse the two loops (i.e. joint the two loops into a single one). You can assume that both data structures A and B are cache aligned, i.e. their first element is placed at the beginning of a cache line.

We ask you:

- (1 point) Which data structure(s) should you decompose in the first loop (input and/or output)? Which geometric decomposition(s) should you apply to achieve the required load balance (block, cyclic or block-cyclic)? Why? Reason your answers.
- (2 points) Implement the OpenMP parallelisation using the geometric decomposition you have decided in the previous question.

- (1 point) In your previous implementation, is there any possibility of false sharing **during the execution of the first loop** due to the use of the selected geometric data decomposition? Why? Reason your answer. If necessary, explain what would you do to avoid it (**we are not asking to re-implement the code to avoid false sharing, just explain**).

Problem 2 (3 points) Given the following parallel code in OpenMP for a *producer-consumer* problem:



```
int buffer[N];
int full_slots=0; // number of full positions in buffer
int empty_slots=N; // number of empty positions in buffer
void down (int *slots);
void up (int *slots);
...
#pragma omp parallel
#pragma omp single
{
    #pragma omp task // producer
```

```

{
    int in=0; // first position in buffer to insert to
    while (1) {
        int data = produceData(); // generates data
        down(&empty_slots);        // blocks when empty_slots is 0 (buffer is full).
                                   // decreases empty_slots once it is not 0 and return

        buffer[in] = data;
        in = (in+1) % N;
        up(&full_slots);           // increments by 1 the counter of full slots
    } }
#pragma omp task // consumer
{
    int out=0; // first position in buffer to remove from
    while (1) {
        down(&full_slots);         // blocks when full_slots is 0 (buffer is empty).
                                   // decreases full_slots once it is not 0 and return

        int data = buffer[out];
        out = (out+1) % N;
        up(&empty_slots);          // increments by 1 the counter of empty slots
        consumeData(data);         // process data
    } } }

```

We ask you to implement the semaphore operations down and up that are used to synchronise the access to the shared buffer, defined as follows:

```

void down (int *slots) {
    // if the value stored in slots is greater than 0, then decrement slots
    // otherwise wait until slots is greater than 0, then decrement slots
}
void up (int *slots) {
    // Increment slots by 1
}

```

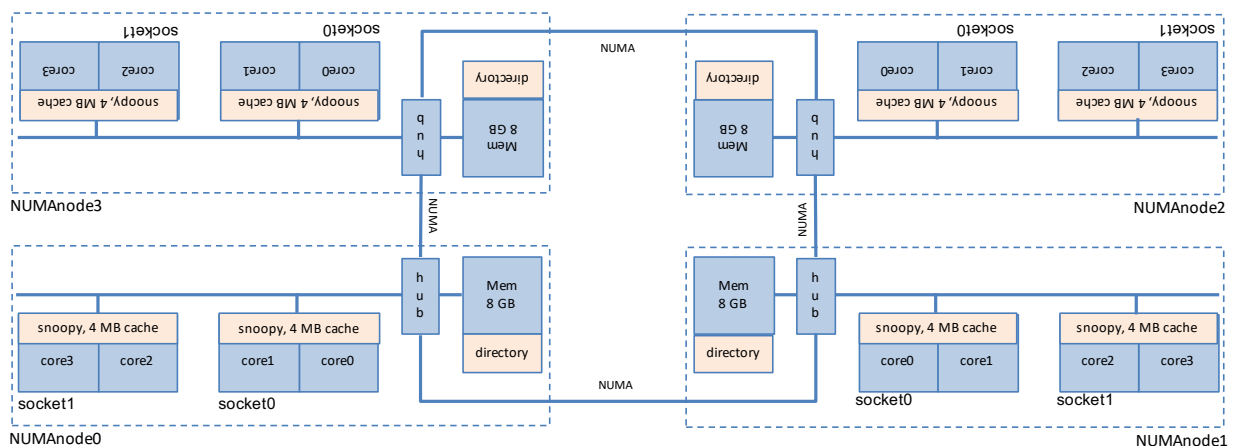
You can assume that the architecture supports any of the low-level synchronisation operations studied during the course and that they are available for use. The implementation **should avoid unnecessary coherence traffic**. Below you have a list of such operations:

```

int test_and_set (int *address, int value);
int fetch_and_op (int *address, int value); // op can be add, sub, ...
int load_linked (int *address);
int store_conditional (int *address, int value); // returns 1 in case of success

```

Problem 3 (3 points) Given the following NUMA system with 4 dual-socket NUMA nodes, each NUMA node with 8 GB of main memory. Each socket has two cores sharing the access to a per-socket 4 MB cache. Coherence inside NUMA nodes is maintained with a snoopy-based mechanism implementing write-invalidate MSI. Coherence among NUMA nodes is maintained with a directory-based mechanism implementing write-invalidate MSU.



coreX: core X within a socket. 16 cores in total.

socketY: socket Y within NUMA node: each socket with 2 cores, sharing 4 MB cache; MSI snoopy coherence protocol.

NUMAnodeW: NUMA node W; each node with two sockets sharing 8 GB of main memory and a hub/directory to keep coherence among nodes, MSU simplified protocol.

Size of memory and cache lines: 64 bytes.

Part I (1 point): Compute the total number of bits that are necessary in the whole system to:

1. Maintain the coherence among NUMA nodes.
2. Maintain the coherence within NUMA nodes.

Part II (2 points): Assume that the home node for the line containing variable `var` is `NUMANode0`, and at a given time there exist 3 clean copies of that line in cache memories: in `socket0` in `NUMANode0`, in `socket0` in `NUMANode1` and in `socket0` in `NUMANode2`. Considering that the following memory accesses are done one after the other: 1) `core2` in `NUMANode0` reads `var`; 2) `core0` in `NUMANode3` reads `var`; 3) `core0` in `NUMANode3` writes `var`; and 4) `core0` in `NUMANode0` writes variable `other`. We ask you to select the sentences that are correct for each one of these 4 memory accesses (for each memory access at least one of the sentences is correct). Each correct selection adds 0.25 points; each wrong selection subtracts 0.17 points; the grade for this part of the problem is always in the range 0–2.

1. When `core2` in `NUMANode0` reads variable `var`, which of the following sentences are correct?
 - (a) `Core2` issues `PrRd`.
 - (b) The snoopy in `socket1` issues `BusRd` on its local bus.
 - (c) The snoopy in `socket0` observes the `BusRd` command and places the line on the bus (`Flush`).
 - (d) The hub associated to `NUMANode0` updates the directory for the line containing `var` to indicate that a new copy of the line exists inside `NUMANode0`.
 - (e) No coherence requests are sent to the rest of the NUMA nodes in the system.

2. Then, when `core0` in `NUMANode3` reads variable `var`, which of the following sentences are correct?
 - (a) The snoopy in `socket0` issues `BusRd` on its local bus.
 - (b) The hub associated to `NUMANode3` finds the closest NUMA node that has a copy of the line and sends a `RdReq` to that NUMA node.
 - (c) The hub of the NUMA node receiving the `RdReq` reads the line from the cache memory that is storing it.
 - (d) `NUMANode3` receives a `Drepy` command with the line containing variable `var` and stores a copy in its main memory.
 - (e) At the end the directory in the home NUMA node is updated so that all bits in the sharers list are set to 1 and the state is kept as S

3. Then, when `core0` in `NUMANode3` writes variable `var`, which of the following sentences are correct?
 - (a) The snoopy in `socket0` of `NUMANode3` issues an `Invalidate` command on its local bus.
 - (b) The hub in `NUMANode3` issues an `Invalidate` command, going to the home NUMA node.
 - (c) The home NUMA node checks if there are copies of the line in other NUMA nodes, sending to each one of them an `Invalidate` command.
 - (d) The state for the line in the caches of the remote nodes receiving the `Invalidate` command (as well as in the home node) changes from S to M to indicate that the line is modified somewhere else.
 - (e) At the end the directory in the home NUMA node only has bit 3 in the sharers list set to 1 and the state set to M.

4. Finally, `core0` in `NUMANode0` writes variable `other` (observe that we are not giving any information about the home NUMA node for the line containing this variable). When this memory accesses finishes the state in the directory for the line containing variable `var` is kept to M, but the bit set to 1 in the sharers list moves from position 3 to position 0. Which of the following sentences are correct?
 - (a) You should tell me which is the home NUMA node for variable `other` in order to decide if the following sentences are correct or not.
 - (b) This state in the directory is not possible since `var` and `other` are different variables.
 - (c) Since both variables reside in different cache lines, this state is possible if both variables are mapped in the same cache line entry (cache line replacement, i.e. the line containing `var` is replaced by the line containing `other`).
 - (d) This is the typical symptom of false sharing.
 - (e) Since variables `var` and `other` are different, there is no need to interchange coherence commands between `NUMANode0` and `NUMANode3`.