PAR – In-Term Exam – Course 2022/23-Q1

November 3^{rd}, 2022

Problem 1 (3.0 points) Given the following code:

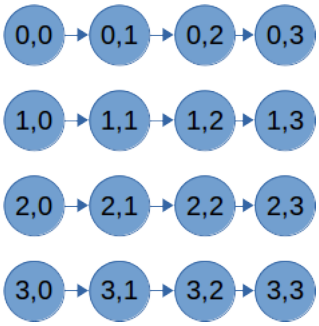
```
#define N  256
#define BS  64
int m[N][N];

for (int ii=0; ii<N/BS; ii++) {
   for (int jj=0; jj<N/BS; jj++) {
      tareador_start_task("task");

      // In general, a task processes a block of BSxBS elements
      // However, if the task is labeled with (ii,jj=0),
      //          this task processes BSx(BS-1) elements
      int i_start = ii*BS; int i_end = i_start+BS;
      int j_start = jj*BS; int j_end = j_start+BS;
      for (int i=i_start; i<i_end; i++)
        for (int j=max(j_start,1); j<j_end; j++)
          m[i][j] = compute (m[i][j], m[i][j-1]);  // tc t.u. (time units)

      tareador_end_task("task");
} }
```

Assume the innermost loop body takes t_c t.u., all variables but matrix m are in registers, function `compute` only reads the values received as arguments and does not modify other positions in the memory, BS perfectly divides N, being BS and N defined in the code above. **We ask you:**

1. (1.0 points) Draw the task dependence graph (TDG), indicating the cost of each task as a function of BS and t_c. Label each task with the values of ii and jj.

   Solution:

   Tasks (ii,jj) when jj!=0 have a cost of $BS \times BS \times t_c$. Tasks (ii,jj) when jj=0 have a cost $BS \times (BS-1) \times t_c$



2. (1.0 points) Compute T_1, T_∞, P_{min} as a function of BS, N and t_c.

   Solution:

   T_1 can be computed as:

   $T_1 = N \times (N-1) \times t_c$;

   T_∞ is the execution time of any of the rows of tasks of the TDG:

   $T_\infty = (BS-1) \times (BS) \times t_c + (\frac{N}{BS} - 1) \times (BS^2) \times t_c$;

This T_∞ is obtained when one processor is assigned to the computation of a row of tasks in the TDG. Therefore:

$P_{min} = \frac{N}{BS}$;

3. (1.0 points) Assuming the assignment of tasks to 4 processors of the table below, calculate T_4 and speedup S_4.

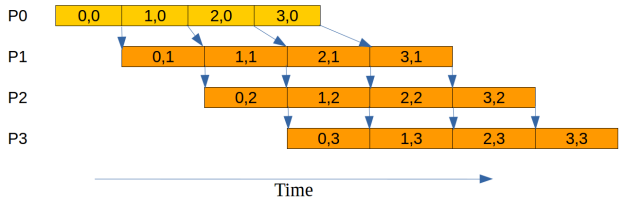
Processor	Tasks
P_0	$(0,0),(1,0),(2,0),(3,0)$
P_1	$(0,1),(1,1),(2,1),(3,1)$
P_2	$(0,2),(1,2),(2,2),(3,2)$
P_3	$(0,3),(1,3),(2,3),(3,3)$

Solution:

$S_4 = \frac{T_1}{T_4}$

$T_1 = N \times (N-1) \times t_c = 65280 \times t_c$ t.u.

Figure below shows the execution timeline considering the assignment of tasks to 4 processors above and their dependences. Each processor executes $\frac{N}{BS}$ tasks. Let's identify those that originate the critical path in the execution timeline. Processor 0 executes tasks with cost $BS \times (BS-1) \times t_c$. First task of Processor 0 contributes to T_4: $((BS-1) \times BS \times t_c)$. Then, first tasks of processors 1 an 2 also contribute to T_4: $((\frac{N}{BS} - 2) \times BS^2 \times t_c)$. Finally, all tasks executed in processor 3 contribute to T_4: $(\frac{N}{BS} \times BS^2 \times t_c)$.



$T_4 = ((BS-1) \times BS + (\frac{N}{BS} - 2) \times BS^2 + \frac{N}{BS} \times BS^2) \times t_c = 28608 \times t_c$ t.u.

$S_4 = \frac{T_1}{T_4} = 2.28\times$

Problem 2 (2.0 points) Given the same code and mapping of tasks to 4 processors as in the previous exercise, assume

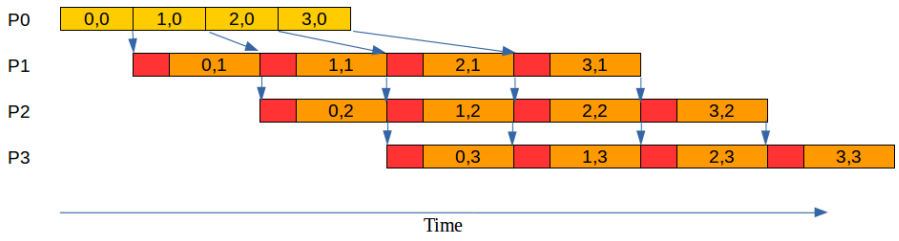
- A distributed-memory architecture with $P = 4$ processors;
- Matrix m is initially distributed by columns (BS consecutive columns per processor).
- Data sharing model with $t_{comm} = t_s + W \times t_w$, being W the number of elements to transfer, and t_s and t_w the start-up time and transfer time of one element, respectively;
- The execution time for a single iteration of the innermost loop body takes t_c t.u.

We ask you: Draw the execution timeline of the execution of tasks and write the expression that determines the execution time T_P, clearly indicating the contribution of the computation time $T_{P\,comp}$ and data sharing overhead $T_{P\,mov}$, as a function of N, BS, P, t_c, t_s and t_w.

Solution:

Figure below shows the execution timeline considering the assignment of tasks to 4 processors above, their dependences and the remote memory accesses. Processors 1, 2 and 3 have to wait for completness of tasks $ii, jj-1$ whose results are saved in the remote memory of the previous processor. As we are now considering data sharing overheads, processors 1, 2 and 3 have to perform a remote memory access of the BS elements of the left boundary before each task is executed. Therefore, the cost of each communication is $t_s + BS \times t_w$. Tasks in processor 0 perform local memory access with no data sharing overhead.

Each processor executes $\frac{N}{BS}$ tasks.



In particular, the contribution of computation and data sharing overheads is the following:

- Processor 0 executes tasks with cost $BS \times (BS-1) \times t_c$. First task of Processor 0 contributes to T_4: $(BS-1) \times BS \times t_c$.
- First tasks of processors 1 an 2 also contribute to T_4: $(\frac{N}{BS} - 2) \times ((t_s + BS \times t_w) + (BS^2 \times t_c))$.
- Finally, all tasks executed in processor 3 contribute to T_4: $(\frac{N}{BS}) \times ((t_s + BS \times t_w) + (BS^2 \times t_c))$.

$T_4 = T_{comp} + T_{comm}$, being:

$T_{comp} = ((BS-1) \times BS + (\frac{N}{BS} - 2) \times BS^2 + \frac{N}{BS} \times BS^2) \times t_c$

$T_{comm} = (\frac{N}{BS} - 2) \times (t_s + BS \times t_w) + \frac{N}{BS} \times (t_s + BS \times t_w) = (2 \times \frac{N}{BS} - 2) \times (t_s + BS \times t_w)$

Problem 3 (5.0 points) Consider a multiprocessor system with a hybrid NUMA/UMA architecture which is composed by 3 identical NUMAnodes. Each NUMAnode has 20 Gbytes of main memory and 2 processors with its own private cache of 8 Mbytes. The memory cache lines are 32 bytes wide, and data coherence is guaranteed using *Write-Invalidate MSI protocol* within each NUMAnode and using a *Write-Invalidate MSU Directory-based* cache coherency protocol among NUMAnodes.

We ask you to answer the following questions:

1. (1.0 points) Compute the total number of bits that are necessary **in each cache memory** to maintain the coherence, indicating the function of those bits.

   Solution:

   We need 2 state bits (MSI) for each line of cache memory. Each cache memory has $(8 \times 2^{20}) \div 32$ lines, that is 2^{18} lines; therefore the number of bits per cache is $2^{18} \times 2 = 2^{19}$ bits.

2. (1.0 points) Compute the total number of bits that are necessary **in each NUMAnode's directory** to maintain the coherence, indicating the function of those bits.

   Solution:

   We need 2 state bits (MSU) and 3 presence bits for each line of main memory. For the overall 20 GB, this is $(20 \times 2^{30}) \div 32$ lines, that is 20×2^{25} lines; therefore the number of bits in the directory is $20 \times 2^{25} \times (2 + 3) = 100 \times 2^{25}$ bits.

Given the following OpenMP code excerpt which is executed on processor 0 from the previous described multiprocessor system:

```
#define N (6*1024)
#define NUM_THREADS 6
int v[N], count[NUM_THREADS];
...
for (int k = 0; k < NUM_THREADS; k++)
   count[k]=0;

/*** POINT A ***/

#pragma omp parallel num_threads (NUM_THREADS)
{
   int id=omp_get_thread_num();
   int num_iter = N/NUM_THREADS;

   for (int k = id*num_iter; k < id*num_iter+num_iter; k++) {
       int value = v[k];
       if (is_prime(value)) /* returns true if "value" is a prime number */
           count[id]++;
   }
}
```

and assuming that: 1) the initial memory address of vector `count` is aligned to the start of a memory/cache line; 2) the size of an `int` data type is 4 bytes; and 3) processors 0 and 1 belong to NUMAnode0, processors 2 and 3 belong to NUMAnode1 and processors 4 and 5 belong to NUMAnode2; and 4) the Operating System applies the "first touch" policy for data allocation in memory. **We ask you to:**

3. (1.25 points) Complete the table in the provided answer sheet with the required information: affected cache line (numbered from the first position where vector `count` is allocated), cache line states (`I/S/M`) in processors 0 to 5, directory entry state (`U/S/M`) and presence bits (0/1, where the lowest ordered bit, the rightmost one, corresponds to NUMAnode0), to keep cache coherence, **when the execution of the previous code reaches POINT A**.

   Solution: The cache line size is 32 bytes, and each element of the vector occupies 4 bytes, so the entire vector `count` fits in a unique memory line.

	Affected line	Home NUMAnode	Cache line state						Directory entry	
			0	1	2	3	4	5	State	Presence bits
count[0]	0	0	M	-	-	-	-	-	M	001
count[1]	0	0	M	-	-	-	-	-	M	001
count[2]	0	0	M	-	-	-	-	-	M	001
count[3]	0	0	M	-	-	-	-	-	M	001
count[4]	0	0	M	-	-	-	-	-	M	001
count[5]	0	0	M	-	-	-	-	-	M	001

4. (1.25 points) Complete the table in the provided answer sheet with the required information: affected cache line, access in cache (hit/miss), CPU command for processor k ($PrRd_k/PrWr_k$), Bus transaction(s) from Snoopy in processor k ($BusRd_k$ / $BusRdX_k$ / $BusUpgr_k$ / $Flush_k$ / $Nothing$), cache line states in processors 0 to 5, directory entry state and presence bits, to keep cache coherence, **AFTER the execution of each** memory access in the table. Assume initially memory and cache states from previous question.

   Solution:

Memory access	Affected line	Hit/Miss	CPU command	Bus transaction(s)	Cache line state						Directory entry	
					0	1	2	3	4	5	State	Presence bits
Processor 1 reads count[1]	0	Miss	PrRd1	BusRd1 / Flush0	S	S	-	-	-	-	S	001
Processor 0 reads count[0]	0	Hit	PrRd0	Nothing	S	S	-	-	-	-	S	001
Processor 1 writes count[1]	0	Hit	PrWr1	BusUpgr1	I	M	-	-	-	-	M	001
Processor 2 reads count[2]	0	Miss	PrRd2	BusRd2 / Flush1	I	S	S	-	-	-	S	011
Processor 0 writes count[0]	0	Miss	PrWr0	BusRdX0	M	I	I	-	-	-	M	001

   Notice that, as the entirely vector `count` fits in a cache line, access to any element of the vector provokes access to the same cache line.

5. (0.5 points) Have you observed any potential efficiency problem in the previous code? Justify briefly your answer.

   Solution:

   False sharing: Vector `count` lies on a single memory line, then all the threads are eventually accessing for writing the same cache line during the concurrent execution of the program, but to different memory addresses, resulting in unnecessary coherence traffic, even between nodes.

   Possible memory bottleneck in the access to memory of processor 0: Vector count is entirely allocated in that memory, and all the threads have to access to it during execution.

Student name: .....

Answer for question 3.3

	Affected line	Home NUMAnode	Cache line state						Directory entry	
			0	1	2	3	4	5	State	Presence bits
count[0]										
count[1]										
count[2]										
count[3]										
count[4]										
count[5]										

Answer for question 3.4

Memory access	Affected line	Hit/Miss	CPU command	Bus transaction(s)	Cache line state						Directory entry	
					0	1	2	3	4	5	State	Presence bits
Processor 1 reads count[1]												
Processor 0 reads count[0]												
Processor 1 writes count[1]			.									
Processor 2 reads count[2]												
Processor 0 writes count[0]												