1. A system $\Sigma = (P_1, P_2)$ is composed of two processes as in the figure below. The first process $P_1$ is in charge of producing a stream of tuples $(op, x)$ with an average inter-arrival time $T_A > 0$ to the second process. The first field $op$ of each tuple is an integer in the range $[0..K - 1]$ while the second field $x$ is a 32-bit integer value. The second process encapsulates an integer array $A[K]$ statically initialized. The behavior of $P_2$ is described by the following LC-like pseudocode:

```plaintext
P_2:: ...
   while true do {
      receive(ch_in, (op, x));
      A[op] = F(x, A[op]);
      send(ch_out, A[op]);
   }
```

The function $F$ is used to update the value of the array at the position in the first field of the received tuple. The new value is also transmitted on the channel exiting from the second process. Assume that $T_A < T_F$, with $T_F$ the average processing time of the function $F$, and answer the following points by providing a detailed analysis and motivated explanations.

   a) Describe a feasible parallelization of the problem that respects the semantics of the original system $\Sigma$. Show the LC code of the distribution process and of a generic worker by justifying your choices in terms of data partitioning and/or replication.

   b) Study in which cases the parallelization is able to express the optimal parallelism degree and which are the scalability issues that may affect the performance of the parallel solution. Provide a theoretical limit of the expected scalability based on the properties of the discrete probability distribution of the $op$ values in the input tuples. *Hint:* reason in terms of acyclic graph analysis and distribution probabilities to the worker processes.

   c) Discuss whether the proposed parallelization guarantees that the ordering of results is the same of the original system $\Sigma$. Sketch a possible solution to address the problem, if any.

2. A chip multi-processor architecture (CMP) is equipped with 16 PEs interconnected by a bidirectional on-chip ring network. The architecture supports automatic cache coherence with a directory-based implementation working according to an update-based protocol.

   a) Describe the main differences between update-based and invalidation-based protocols and the conceptual pros and cons of the two approaches.

   Assume that the runtime of parallel applications is based on the RDY-ACK solution with shared-memory synchronization flags. Processes are exclusively mapped onto PEs. State whether and why the following sentences are true or false by providing the necessary justifications to your answers.

   b) With respect to an invalidation-based protocol, the overhead of an update-based solution is theoretically lower because no ping-pong effect may arise during the execution of parallel programs.

   c) In the implementation of the emitter process in a farm parallelization, the home-flushing optimization is completely useless because no invalidation messages are transmitted from the workers to the emitter PE.
Solution Outline

1. We can observe that all the stream elements having the same $op$ field always modify the same element of the array $A$. So, in order to provide the equivalence with the sequential computation, we can execute in parallel input tuples with different $op$ fields while the tuples with the same $op$ must be rigorously executed in serial order.

The parallelization is based on a farm with a custom distribution policy and data partitioning of the array $A$. With $n = \lceil TF / TA \rceil$ the optimal parallelism degree, each worker is statically assigned to $K \cdot n$ entries of the array, which is statically partitioned among the workers. The emitter process is in charge of distributing input tuples to the workers as in the following pseudocode:

```pseudocode
EMITTER::
channel in ch_in(1); channel out ch_w[n]; tuple t;
while true do {
  receive(ch_in, t);
  $w = t.op \mod n$;
  send(ch_w[w], t);
}
```

The LC code of the generic worker is straightforward, with the array $A$ statically partitioned among the worker instances. In this solution, we are implicitly assuming that the number of possible $op$ fields is much greater than the optimal parallelism degree and that all the logical substreams (i.e. the sequence of tuples with the same $op$ value) have the same probability, that is the uniform distribution $p = 1/K$. In this case, each worker is assigned to the same number of logical substreams and the optimal parallelism degree can be achieved. In fact, with the assumptions made (i.e. uniform distribution), there are no load balancing problems in the parallel execution and approximately the same number of input tuples are distributed per worker. We can identify at least two different situations where this ideal behavior cannot be achieved:

- if $K < n$, the optimal parallelism degree cannot be reached because the number of substreams limits the maximum parallelism degree that we can exploit in this problem. Therefore, the parallelization remains a bottleneck and the maximum scalability, provided that no other performance degradation reason exists, is given by:

$$S = \frac{T^{(1)}}{T^{(K)}} \approx \frac{TF}{KF} = K$$

- the distribution logic described above assumes that all the substreams have the same probability. However, it might exist some substreams much more frequent that the others and some workers, that is the ones receiving the most probable substreams, may become congested by preventing the optimal parallelism degree to be achieved. Of course, clever hash functions can be designed to assign substreams to workers in the most balanced way as possible. However, as a general evaluation, if the most frequent substream has probability $p_m \gg 1/K$ to occur, the theoretical limit of the maximum scalability achievable is given by:

$$S = \frac{T^{(1)}}{T^{(K)}} \approx \frac{TF}{TF \cdot p_m} = \frac{1}{p_m}$$

As an example, if the most frequent substream has probability of 25%, the maximum scalability of the parallelization is 4. If this value is smaller than the optimal parallelism degree, we can state that the parallelization remains a bottleneck for sure.

The proposed parallelization guarantees that all the output results are correctly computed. However, due to the non-determinism in the workers’ computation and their relative speed, the results can be emitted in a different order than the original system. The problem can be solved as in a traditional farm. The emitter can add to each input tuple a new field counting the number of input elements while the distribution is still performed as previously discussed using a modulo function to map substreams onto worker
identifiers. The workers compute the results accessing to their elements of the array, and they append to each result the same identifier of the corresponding input tuple. The collector is in charge of buffering the results using a priority queue, and a result with identifier $i$ is emitted if and only if all the results with identifier less than $i$ have been received and already forwarded onto the output stream.

2. The material to answer the second exercise can be found in the official slides of the course and in the textbook. Point a) can be answered using the 15th slide block (slides 8-20). The answer to point b) is false, because the effect of ping-pong situations between two processes is approximately the same with invalidation-based and update-based solutions. While in the first case processes continuously invalidate blocks with each other, in the second scheme update messages are exchanged to keep different copies of the same cache block updated and coherent. In both cases, the modified block is continuously transferred among the secondary caches of the involved processes causing the ping-pong detrimental effect. The answer to point c) is false. Home-flushing semantics is useful also in case of update-based CC protocols. While usually we use the home-flushing technique to avoid invalidations directed to the Emitter’s secondary cache (in the scheduling phase of a farm), in case of an update-based CC protocol the home-flushing semantics is useful to avoid the transmission of firmware update messages from the Workers PEs to the Emitter PE, i.e. when a Worker PE modifies the ACK flag the whole block of the channel descriptor (the first cache block of the VTG_S structure) is transmitted to the Emitter’s secondary cache. In case the target variable is also modified by the Worker, all the blocks of the VTG_S structure are updated into the C2 of the Emitter by rendering this problem much more aggressive for the under-load latency viewpoint. Therefore, with high level of parallelism (number of Workers), the contention parameter may easily grow by producing a high contention. The use of home-flushing semantics in this case prevents the Workers PEs to generate firmware messages to the Emitter’s secondary cache to update its copy of the modified cache blocks. In conclusion, the home-flushing semantics is still useful and effective also in case of update-based cache coherence protocols.