1. In a single-CMP architecture with directory-based cache coherence we have a $k$-asynchronous LC symmetric channel implemented according to the RDY-ACK run-time support with shared-memory synchronizations.
   a) Describe in detail the evaluation of the term $T_{setup}$ of the send cost model by assuming the cache coherence system behaving according to the basic invalidation semantics (assume the home node of the channel shared data structures a third PE with respect to the sender and the receiver PEs).

2. An OR computation graph is composed of a process $\Sigma$ generating a stream of messages of size $L > 0$ words each transmitted to one of $n > 0$ destination processes with the same probability. Each destination executes a computation with calculation time in the order of $O(L)$ that accesses exactly one time all the cache blocks of the target variable without modifying them. Assume that $L$ is smaller than the capacity of the primary and secondary caches of the PEs.
   a) Choose a homing selection strategy for the shared data structures of the run-time support and explain in detail whether and why the basic invalidation semantics is able to provide low-$p$ in this scenario;
   b) Is the answer of the previous point substantially different if we assume that the destination processes modify the blocks of the target variables during their computation?

3. A system $\Sigma$ is composed of two processes $P_1$ and $P_2$ each generating a stream of $R = 10^5$ arrays of size $M = 16K$ integers with ideal service time $30M\tau$. A third process $P_3$ encapsulates an initialized integer variable $s$ and applies the following computation where $A[M]$ and $B[M]$ are two arrays received respectively from the first and the second process:
   int $y = 0$;
   for i=0 to M-1 do
   C[i] = $F(A[i], B[i], s)$;
   $y = y \oplus C[i]$;

Assume that the function $F$ has calculation time $T_F = 800\tau$ and $\oplus$ is an associative and commutative binary operator with calculation time $T_\oplus = 120\tau$. The result is the integer $y$ transmitted onto the output channel. By referring to the architectural specifications below, answer to the following points:
   a) Study the ideal and effective service times and the relative efficiency of the three processes according to the base latency. Evaluate the completion time of the program;
   b) Parallelize process $P_3$ with a data parallel parallelization. Evaluate the same metrics of the previous point for the parallel system.

Architectural specifications for question 3

- NUMA-SMP CMP-based multiprocessor with 16 CMPs and external NUMA network based on a binary butterfly with wormhole flow control (1 flit = 1 word);
- Each CMP has 8 PEs, an internal crossbar and 2 MINFs connected to 2 parallel WW units. The local memory sub-system of a CMP is composed of 2 mutually interleaved macro-modules directly interconnected with the WW units and having clock cycle $\tau_M = 20\tau$;
- Each PE has L1d of 64K and L2 of 512K on-demand caches, $T_{instr} = 2\tau$, block size $\sigma_1 = 8$;
- Directory-based cache coherence with home flushing. RDY-ACK run-time support based on shared-memory synchronizations;
- All the firmware interfaces use double buffering with $T_{tr} \approx \tau$ inter-chip;
- All the PEs have a communication processor sharing the L1/L2 private caches of the PE;
- Exclusive mapping.
Solution

1. A channel consists of the following data structures at the run-time support level:
   - two private data structures CH_P (one for the sender and one for the receiver) containing pointers to the shared VTG_S instances;
   - two private index variables (one for the sender and one for the receiver);
   - a set of k > 0 VTG_S instances shared between the sender and the receiver processes. Each VTG_S contains a first cache block with the RDY and ACK flags, and L/σ blocks for the target variable.

The execution of a send primitive manipulates the block of the RDY/ACK flags (hereinafter named CH) of the used VTG_S (the one with identifier in the index variable of the sender). Apart from the initial execution phase and for relatively small asynchrony degrees, once a send primitive is executed the CH block of the corresponding VTG_S instance is likely in the C2 of the receiver PE. To execute the send, the sender process (executed on PE_S) loads the block of CH by transferring it from the C2 of PE_R (the receiver PE). The home node of the block is a third PE (let it be PE_H). In general, this C2C read request is transmitted from PE_S to PE_H and from this last to PE_R that forwards the words of the block directly to PE_S. However, by exploiting updated information in its LSK, PE_S can execute an optimized protocol, i.e. the C2C read request is transmitted directly from PE_S to PE_R (and in parallel also PE_H is notified in order to update the directory). On a single-CMP architecture the latency of this LOAD is approximately \( L_{C2C}(\sigma) = 20\tau \) (see Section 21 of the textbook).

We can observe that the block CH is loaded by PE_S exactly one time if PE_S finds the ACK flag equal to 1, or two times if ACK is equal to 0 the first time. In this last case the block is first transferred to C1/C2 of PE_S, then PE_S executes a busy-waiting phase until the block is invalidated by PE_H and transferred back into C1/C2 from C2 of PE_R. Let \( p_w \) be the probability that the flag ACK is equal to 0 the first time.

After this phase, the sender process modifies the blocks of the target variable (not part of \( T_{setup} \)) and sets the flag RDY to 1 (and reset ACK to 0). The last STORE on CH is a write-back that generates a STORE notification to PE_H. The latency of this notification on a single-CMP is roughly \( L_{STORE}(\sigma) = 20\tau \). Once received the STORE notification (and before transmitting the corresponding acknowledgement), PE_H transmits an invalidation to PE_R and waits for the corresponding acknowledgement. This latency is about \( L_{inv}(\sigma) \approx 12\tau \). In summary the setup latency of the send primitive is:

\[
T_{setup} = (1 + p_w) \cdot L_{C2C}(\sigma) + L_{STORE}(\sigma) + L_{inv}(\sigma) + \beta_{code} \leq 80\tau
\]

Note that PE_H is also in charge of updating the main memory copy, however this action is completely asynchronous and the acknowledgement message is transmitted to PE_S without waiting for the completion of the writing in main memory (explain why...).

2. In the basic invalidation semantics, a STORE notification is a firmware message generated by the non-home PE_X executing a write-back STORE instruction on a block B. The notification is transmitted to the home node PE_H and can optionally convey the words of the block. The main purpose of a STORE notification is to inform the home node that the owner of the block becomes PE_X which is the unique PE in the architecture having the valid copy of the cache block. PE_H is in charge of transmitting invalidation messages to all the other PEs currently having a copy of the block. PE_H replies with an acknowledgement only when it is sure that all the invalidations have been completed. In summary, the main difference w.r.t. the home-flushing semantics is: i) the block is not allocated neither in C2 nor in C1 of the home node; ii) the block is still present in C1/C2 of the non-local PE_X.

In the case mentioned by the exercise, the basic invalidation semantics is not able to provide low-p even if we properly choose the best homing (i.e. each channel is homed at the destination PE). Several firmware messages are exchanged between the PE of process S and the PEs of the destination processes. High contention may exist in the secondary cache of S because each destination, once received a message, starts using the blocks of the corresponding target variable, i.e. such blocks must be transferred from the C2 of the sender PE. Therefore, each destination PE generates \( L/\sigma \) C2C read requests to C2_S per effective service time of the destination process. By assuming that none of the destination processes is a bottleneck, we have \( T_{eff-d} = T_{id-s} \cdot n \) and the cost model parameters are \( p = n + 1 \) and \( T_p \) is at least \( T_p \leq (\sigma \cdot T_{id-s})/2L \). This situation likely leads to high contention with a high number of destinations and fine-grained service times.
In this situation the only invalidation message per service time (negligible) transmitted by the PE of each destination to PE_S is the one for invalidating the block of the RDY/ACK flags of the utilized target variable instance. The case considered at point b) of the exercise potentially increases contention, because each destination PE generates additional $1 + L/\sigma$ invalidations to PE_S. Although $p$ does not change, the $T_p$ parameter decreases because more requests are transmitted per service time and this further contributes to contention.

3. The two sources of the graph have the same ideal service time. The inter-arrival time to the third process is equal to the service time of the two sources, because to start the computation $P_3$ needs to have an array from both the sources (AND semantics). Therefore, $T_{A-P3} = 30M\tau$.

The behavior of the third process is the following:

```
START: <receive A from P1>;
   <receive B from P2>;
   <compute phase>;
   <send y outside>;
GOTO START
```

A possible D-RISC compilation of compute phase is provided below:

```
CLEAR Ri
CLEAR Ry
& & σ-unfolding, write_back asynchronous & { }
LOOP: LOAD RA-addr, Ri, Ra
   LOAD RB-addr, Ri, Rb
   CALL RF, Rret
   STORE RC-addr, Ri, Rc
   CALL Rop, Rret
   INCR Ri
   IF < Ri, RM LOOP }
```

In the compilation RM is initialized with the value of the constant $M$, RA-addr, RB-addr and RC-addr contain the initial logical addresses of the two input arrays and of the output array. Register Ri is used to index the arrays. The function $F$ is compiled in order to receive the input arguments from three general registers: Ra contains the i-th element of the first array, Rb of the second and Rs the variable $s$ (the compiler temporarily allocates this variable in a register in the for loop). The second CALL is needed to execute the operator $\otimes$ which we suppose has been compiled separately (its code is not specified in the text). The function $\otimes$ takes two input parameters passed through general registers: the first is Ry where the compiler has allocated the variable $y$, the second Rc contains the i-th element of the output array. The result of the function is directly written in Ry. The pure calculation time is:

$$T_{calc-P3-0} = M(7 \cdot T_{inst} + T_F + T_\otimes) = M(14\tau + 920\tau) = 934M\tau$$

The computation is characterized by locality on the blocks of the input arrays, while we can exploit reuse on the block of the variable $s$. According to the home flushing semantics, and by assuming the home node of the two input channels the third process, the blocks of the two arrays are flushed directly into the C2 and C1 of the home node. Observe that the primary cache capacity is sufficient to likely contain the two arrays and the output array. Therefore, we can assume that no cache fault is generated during the execution of the previous code and:

$$T_{calc-P3 - T_{calc-P3-0} = 934M\tau}$$

The output message generated by the process is just one word, therefore the communication latency can be neglected. The ideal service time is equal to $T_{id-P3} = 934M\tau$.

$$n_{opt} = \left\lfloor \frac{T_{id-P3}}{T_{A-P3}} \right\rfloor = \left\lfloor \frac{934M\tau}{30M\tau} \right\rfloor = 32$$
The number of PEs is 128, thus sufficient to accommodate the necessary parallelism degree. The third process is obviously a bottleneck and the performance parameters of $\Sigma$ are:

<table>
<thead>
<tr>
<th>Module</th>
<th>Ideal Ts</th>
<th>Effective Ts</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$30M\tau$</td>
<td>$934M\tau$</td>
<td>0.03</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$30M\tau$</td>
<td>$934M\tau$</td>
<td>0.03</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$934M\tau$</td>
<td>$934M\tau$</td>
<td>1</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>$30M\tau$</td>
<td>$934M\tau$</td>
<td>0.03</td>
</tr>
</tbody>
</table>

The completion time is given by:

$$T_{c-\Sigma} = R \cdot T_\Sigma = 10^5 \cdot 934M\tau$$

A possible data-parallel parallelization is a MAP based on $M$ virtual processors defined as follows: $VP_{i=0,...,M-1}\{A[i],B[i],C[i],s\}$. The parallel program is composed of a scatter process that receives the pairs of arrays from the two sources, and partitions the arrays to the workers (each one encapsulating a read-only copy of the variable $s$). Each worker computes the elements in its partition of the array $C$ and computes the local reduce result.

The ideal service time of the scatter process is:

$$T_{id-s} \approx T_{scatter}(n) \approx 2MT_{trans}$$

We assume the following homing of the channels:

<table>
<thead>
<tr>
<th>Channel</th>
<th>Home PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1/P2-Scatter</td>
<td>Scatter</td>
</tr>
<tr>
<td>Scatter-Wi</td>
<td>Wi</td>
</tr>
<tr>
<td>Wi-Reducer</td>
<td>Wi</td>
</tr>
</tbody>
</table>

The scatter process finds the blocks of the two arrays in its C1 and C2 and flushes them to the workers. We map the two source processes, the scatter process and the reducer process onto the first CMP. Other four CMPS are needed for the 32 workers. We have:

$$2MT_{trans} = 2\frac{M}{\sigma}L_{write-C2C}(\sigma) + 2M\beta_{code}$$

A C2C flush message is forwarded over the following path: C1-C2-W-INT_NET-MINF-WW-EXT_NUMA_NET-WW-MINF-INT_NET-W-C2-C1, i.e. $d = 12 + d_{numa}$. According to our mapping the distance in the NUMA network is:

$$d_{numa} = 3 \cdot \frac{1}{4} + 5 \cdot \frac{1}{2} + 7 \cdot \frac{1}{4} = 5$$

The write flush latency is $L_{write-C2C}(\sigma) = 84\tau$. We have:

$$T_{id-s} = 2\frac{M}{\sigma}L_{write-C2C}(\sigma) + 2M\beta_{code} = 29M\tau < T_{A-P3}$$

The cache exploitation of the workers is the same of the sequential process $P_3$. The worker ideal service time is:

$$T_{id-w} = \frac{M}{n} \left(7 \cdot T_{instr} + T_P + T_{\phi}\right) = 29.2M\tau$$

It is easy to observe that the computation of the global reduce result in the Reducer process is very small and does not represent a significant overhead in this application, i.e. it is approximately $T_{\phi} \cdot (n-1) < 500\tau$, therefore this process is not a bottleneck. A slightly better latency could be obtained by implementing the global reduce computation according to a binary tree mapped onto the workers. In that case the small overhead of the global reduce communications/computation must be taken into account in the evaluation of the ideal service time of the workers (do it as an exercise). The final analysis is given below:
<table>
<thead>
<tr>
<th>Module</th>
<th>Ideal Ts</th>
<th>Effective Ts</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{1,2}$</td>
<td>$30M\tau$</td>
<td>$30M\tau$</td>
<td>1</td>
</tr>
<tr>
<td>$S$</td>
<td>$29M\tau$</td>
<td>$30M\tau$</td>
<td>0.97</td>
</tr>
<tr>
<td>$Wi$</td>
<td>$29.2M\tau$</td>
<td>$30M\tau$</td>
<td>0.97</td>
</tr>
<tr>
<td>$R$</td>
<td>$O(1)$</td>
<td>$30M\tau$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>$30M\tau$</td>
<td>$30M\tau$</td>
<td>1</td>
</tr>
</tbody>
</table>

The completion time is given by:

$$T_{c-\Sigma} = R \cdot T_{\Sigma} = 10^5 \cdot 30M\tau$$