Master Degree Program in Computer Science and Networking, 2014-15

High Performance Computing

5^{th} appello – September 8, 2015

Write your name, surname, student identification number (numero di matricola), e-mail. The answers can be written in English or in Italian. Please, present the work in a legible and readable form. All the answers must be properly and clearly explained.

1) Questions *a)* and *b)* refer to the Architectural Specifications given below:

a) Two processes have the following structure, where *y* is a shared integer variable and *x*, *z* are private:

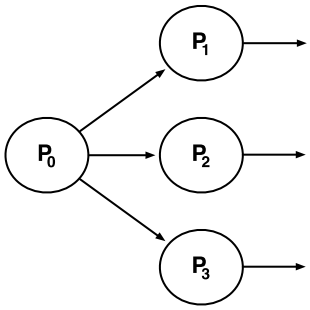
Q_0 :: y = F(x); *notify*(ok)

Q_1 :: *wait*(ok); z = G(y)

Determine the conditions for which the final value of *z* is equal to *G(F(x))*.

b) Characterize the structure and behavior of the generic switching node with the goal of maximizing its bandwidth and minimizing the firmware complexity, and determine the switching node internal latency.

2) Consider a system Σ composed of four processes as in the figure. P_0 generates an input stream of elements x of integer type. The probability of transmitting a message to P_1 and to P_2 is $1/5$ and $2/5$ respectively. The computation is characterized as follows:



1. the ideal service time of P_0 is $5 \times 10^3\tau$;
2. P_1 executes the following code: ∀ input x: output $y = F(x,z)$, where z is the previous stream element received from P_0 (z is supposed to be initialized to zero). The mean processing time is $T_F = 7 \times 10^3\tau$;
3. P_2 executes the following code:

   ∀ input x: output $y = \{y_1 = G_1(x); s = G_2(x,s); y = G_3(y_1,s)\}$

   where s is an integer variable initialized to zero. The mean processing times of G_1, G_2 and G_3 are $2 \times 10^3\tau$, $15 \times 10^3\tau$ and $2.5 \times 10^3\tau$ respectively;
4. P_3 encapsulates an integer array $A[M]$ with $M = 10^3$ and is defined as follows, with $T_{H_1} = T_{H_2} = 50\tau$:

```
∀ input x: output y ={
  y = 0;
  for i = 0 to M − 1 do
    if (i < M − 1) A[i] = H1(A[i], x, A[(i + 1)%M]);
    else A[i] = H2(A[i], x);
    y = y + A[i];
}
```

The Architectural Specifications are given below. In addition $T_{setup} = 10^1\tau$, $T_{transm} = 10^3\tau$, and $R_Q \sim R_{Q0}$.

a) Evaluate the ideal/effective service times and the relative efficiency of all the processes and of Σ. Transform all the possible bottlenecks in order to achieve the maximum bandwidth. Re-evaluate the ideal/effective service times and the relative efficiency of all P_is and of the whole system.

b) Explain whether and why these sentences are true or false: *i)* "*To achieve the maximum bandwidth it is necessary to perfectly balance the workload of P_1, P_2 and P_3*"; *ii)* "*The ideal parallelism degree of any P_i can be determined independently from the performance of the other P_is*".

Architectural Specifications

1. Single-CMP with 16 PEs, 4-ary 2-cube wormhole interconnect, 4 MINFs, 1-Tera external memory;
2. each PE is D-RISC with on-demand inclusive 2-level cache, 32K+32K C1, 512K C2, 8-word C1-blocks. The mean service time per instruction, in absence of cache faults, is equal to 2τ;
3. directory-based cache coherence with home flushing strategy;
4. zero-copy communication with communication processor, and exclusive mapping.

Solution

1)

a) The existence of cache coherence mechanisms is not sufficient, because the main problem is *memory ordering.* It is necessary that *notify(ok)* is executed when *y* has been actually assigned the value of *F(x)*, thus the STORE instruction with which *y* is modified must be synchronous (or followed by a Memory_Barrier).

b) The maximum bandwidth is numerically equal to the node degree: four flits, received from the four input interfaces, can be transmitted in parallel, during the same clock cycle, onto distinct output interfaces in the most favourable case of conflict absence. This behavior could be achieved even with a single-unit implementation, as the required parallelism can be exploited at the clock cycle level in the same microinstruction. However, the resulting firmware complexity is $O(a^4)$, with $a >> 2$ (new header flits can be received during the transmission of value flits from other interfaces, thus *a* is of the order of the average number of flits per message).

The minimum *O(1)* firmware complexity is achieved by providing four input units, one for each input link, and four output units, one for each output link. The input and the output units are fully connected (16 links, i.e. the same number of links in the Operation Part of a single-unit implementation). An input unit applies the routing function (e.g. deterministic dimension routing) to a received header flit and forwards the header flit itself, followed by the value flits, to the correspondent output unit (or to the PE). Each output unit works non-deterministically on the header flits received from the four input units (and from the PE): when a link is selected, only this link is listened deterministically until all the flits of the same message have been received and forwarded.

The counterpart to the minimum firmware complexity is the internal latency, which passes from one clock cycle (single-unit implementation) to two clock cycles (both an input and an output unit must be traversed).

2) This exercise gives to the student the values of T_{setup} and T_{trasm} that are necessary to apply the communication cost model. In general, the student must explicitly find the values of these parameters by applying the fundamental concepts of the course (e.g. evaluating the impact of the cache coherence according to the run-time support used). The communication latency (in all the cases is an integer data type of one word) is:

$$T_{send}(1) = T_{setup} + T_{transm} \sim T_{transm} = 10^3\tau$$

which is smaller than the ideal service time of P_0, therefore it can be completely masked by the communication processor.

a) The first step is to evaluate the initial transient inter-arrival times to the processes P_1, P_2 and P_3. The results are the following:

$$T_{A-1} = \frac{T_{p_{0-id}}}{p_1} = \frac{5 \times 10^3\tau}{1/5} = 25 \times 10^3\tau$$

$$T_{A-2} = \frac{T_{p_{0-id}}}{p_2} = \frac{5 \times 10^3\tau}{2/5} = 12.5 \times 10^3\tau$$

$$T_{A-3} = \frac{T_{p_{0-id}}}{1 - p_1 - p_2} = \frac{5 \times 10^3\tau}{2/5} = 12.5 \times 10^3\tau$$

The next step is to determine which process is a bottleneck and properly parallelize it. To this end, we need to evaluate the ideal service times of P_1, P_2 and P_3.

In the ***first phase*** we analyze the system "as is" in order to correctly evaluate the utilization factors of the processes, their relative efficiency and to detect bottlenecks that will be removed in the next step.

- *Ideal service time of P_1*

P_1 executes for each stream element a stateful computation in which the internal state, an integer variable z, is read and modified by the process. There is reuse of z in C1. No fault is generated during the execution on each stream element. P_0 writes the value of the message x into the target variable of the channel which is directly flushed in the C1 (and C2) of P_1's PE (if this PE is set as the home node). With very good approximation the ideal service time is equal to the calculation time of the function F executed on each stream element. Therefore we have:

$$T_{P_1-id} \sim T_F = 7 \times 10^2\tau$$

The utilization factor is:

$$\rho_1 = \frac{T_{P_1-id}}{T_{A-1}} = \frac{7 \times 10^3\tau}{25 \times 10^3\tau} = \frac{7}{25} = 0.28 < 1$$

In conclusion, we can state that P_1 is not a bottleneck.

- *Ideal service time of P_2*

P_2 executes a stateful computation on each stream element, as the integer variable s is modified by the function G_2 and read by G_3 to produce the output. The message x is directly flushed into the C1 (and C2) of P_2's PE (which we can suppose it the home node of channel's blocks) while reuse can be exploited on the state variable s, which is in cache C1 at each execution of a stream element. The ideal service time can be determined as follows:

$$T_{p_2-id} = T_{G_1} + T_{G_2} + T_{G_3} = 2 \times 10^3\tau + 15 \times 10^3\tau + 2.5 \times 10^3\tau = 19.5 \times 10^3\tau$$

The ideal service time is greater than the communication latency of the output result that can be fully masked. The utilization factor is:

$$\rho_2 = \frac{T_{P_2-id}}{T_{A-2}} = \frac{19.5 \times 10^3\tau}{12.5 \times 10^3\tau} = \frac{19.5}{12.5} = 1.56 > 1$$

Therefore, P_2 is a bottleneck and we need to find a parallelization of it.

- *Ideal service time of P_3*

P_3's computation reads and modifies an integer array A of size $M = 1000$ integer elements. The whole array can be contained both in C1 and C2, thus we can exploit fully *reuse* of A's blocks from stream element to stream element. The working set contains the whole array A. The ideal service time is roughly equal to the calculation time since this last is $O(M)$ and the communication latency $L_{com}(1)$ is a (small) constant.

A naive compilation in D-RISC code of the calculation phase is the following:

```
        SUB RM, 1, RM-1          RM contains the constant M = 1K
        CLEAR  Ri                Ri contains the index of the for loop
        CLEAR  Ry                Ry contains the value of the result
        LOAD   Rvtg, 0, Rx       Rx contains the value of the input message
LOOP:   LOAD   RA, Ri, Ra don't deallocate   RA contains the initial logical address of array A
        IF <   Ri, RM-1, THEN    RH1, RH2 contain the initial logical addresses of the two
        CALL   RH1, Rret           functions of the program (inputs on registers Ra, Rx
        STORE  RA, Ri, Rris        and Ra+1, and output on register Rris).
        GOTO   CONT
THEN:   ADD    Ri, 1, Ri+1
        LOAD   RA, Ri+1, Ra+1
        CALL   RH2, Rret
        STORE  RA, Ri, Rris
CONT:   ADD    Ry, Rris, Ry
        INCR   Ri
        IF <   Ri, RM, LOOP
```

We can neglect the execution of the else statement because it is executed only at the last iteration. The calculation time, which is also the ideal service time in this case, is:

$$T_{P_3-id} = M\left(9T_{instr} + T_{H_1}\right) = M(18\tau + 50\tau) = M68\tau = 68 \times 10^3\tau$$

It is worth noting that no cache fault penalty is paid during the for loop, as we can assume that P1, P2, P3 are home nodes of their respective input channel descriptors, thus the message variable x has been flushed in C1 (and C2) by the *send* executed by P_0. Moreover all the blocks of A can be contained both in C1 and C2. The communication latency to transmit the message y can be obviously masked to the internal calculation phase.

To correctly evaluate the utilization factor, we need to correct the inter-arrival time to P_3 since P_2 is a bottleneck and the source P_0 is periodically blocked from sending stream elements. The new correct inter-arrival time to P_3 is equal to:

$$T'_{A-3} = \frac{T_{P_2-id}}{\frac{2}{5}} \times \frac{2}{5} = T_{P_2-id} = 19.5 \times 10^3\tau$$

The utilization factor is:

$$\rho_3 = \frac{T_{P_3-id}}{T'_{A-3}} = \frac{68 \times 10^3\tau}{19.5 \times 10^3\tau} = 3.48 > 1$$

Therefore also P_3 is a bottleneck needing to be parallelized.

To complete the analysis we need to correctly re-evaluate the inter-arrival times to P_1 and P_2 since P_3 is the biggest bottleneck of the system. The corrected inter-arrival times are:

$$T'_{A-1} = \frac{T_{P_3-id}}{\frac{1}{5}} \times \frac{2}{5} = T_{P_3-id} \times 2 = 136 \times 10^3\tau$$

$$T'_{A-2} = \frac{T_{P_3-id}}{\frac{2}{5}} \times \frac{2}{5} = T_{P_3-id} = 68 \times 10^3\tau$$

and the corrected utilization factors become:

$$\rho_1 = \frac{T_{P_1-id}}{T'_{A-1}} = \frac{7 \times 10^3\tau}{136 \times 10^3\tau} = 0.05 < 1$$

$$\rho_2 = \frac{T_{P_2-id}}{T'_{A-2}} = \frac{19.5 \times 10^3\tau}{68 \times 10^3\tau} = 0.287 < 1$$

- *Evaluation of the system Σ*

We report in the following table the ideal service times, the effective ones and the relative efficiency of each process and of the whole system.

	Ideal TS	***Effective TS***	***Efficiency***
$\boldsymbol{P_0}$	$5 \times 10^3\tau$	$27.2 \times 10^3\tau$	0.18
$\boldsymbol{P_1}$	$7 \times 10^3\tau$	$136 \times 10^3\tau$	0.051
$\boldsymbol{P_2}$	$19.5 \times 10^3\tau$	$68 \times 10^3\tau$	0.287
$\boldsymbol{P_3}$	$68 \times 10^3\tau$	$68 \times 10^3\tau$	1
$\boldsymbol{\Sigma}$	$5 \times 10^3\tau$	$27.2 \times 10^3\tau$	0.18

The ***second phase*** starts from the beginning. We analyze the system from the source and this time we (try to) remove each bottleneck by finding the ideal parallelism degrees and a proper parallelization of each of them.

- *Parallelization of the bottlenecks*

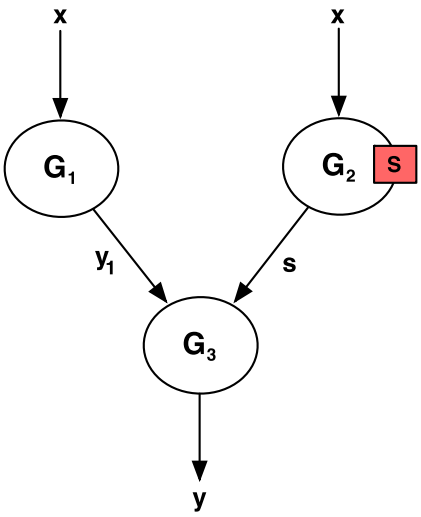
We already know that P_1 is not a bottleneck whereas P_2 needs to be parallelized. The *initial* inter-arrival time to P_2 was $T_{A-1} = 12.5 \times 10^3\tau$ and the ideal parallelism degree is:

$$n_2^{opt} = \left\lceil \frac{T_{P_2-id}}{T_{A-2}} \right\rceil = \left\lceil \frac{19.5 \times 10^3\tau}{12.5 \times 10^3\tau} \right\rceil = 2$$

A feasible parallelization is the *data-flow* one. By applying the Bernstein conditions, we obtain the data-flow graph depicted in the next figure, where node G_2 encapsulates the integer state variable s.

Notice that the computational equivalence with the sequential version is respected, since the s value in G_2 domain corresponds to the present value while the one in G_3 domain is the next state, i.e. the output of G_2's computation which is transmitted to G_3. The graph has overall parallelism degree three, however it is not able to remove the bottleneck because the calculation time of G_2 is still greater than the inter-arrival time to P_2. Moreover, G_2 cannot be further parallelized since it has a modifiable internal state. In conclusion the ideal service time of the data-flow parallelization of P_2 is:

$$T_{DF-P_2} = max\{T_{G_1}, T_{G_2}, T_{G_3}\} = 15 \times 10^3\tau$$



The utilization factor becomes:

$$\rho_2 = \frac{T_{DF-P_2}}{T_{A-2}} = \frac{15 \times 10^3 \tau}{12.5 \times 10^3 \tau} = 1.2 > 1$$

It is interesting to observe that an alternative parallelization can be designed by using the *pipeline* paradigm with two stages: the first executes the second function G_2 on the modifiable state (encapsulated in this stage) and transmits the new state and propagates the input integer x to the second stage executing function G_1 and G_3 serially. This solution has the same ideal service time of the data-flow approach with fewer resources. However, it has also a higher latency. In the following we assume to use only the data-flow solution.

Now we can proceed with the parallelization of P_3. It is worth noting that the optimal parallelism degree of P_3 must be calculated by taking into account that P_2 has been parallelized but the bottleneck is not removed. Without this information the optimal parallelism degree of P_3 would be:

$$n_3^{opt} = \left\lceil \frac{T_{P_3-id}}{T_{A-3}} \right\rceil = \left\lceil \frac{68 \times 10^3 \tau}{12.5 \times 10^3 \tau} \right\rceil = 6$$

It is erroneous and, more precisely, it is an overestimation of the real optimal parallelism degree. To calculate the correct value we need to evaluate the new inter-arrival time to P_3:

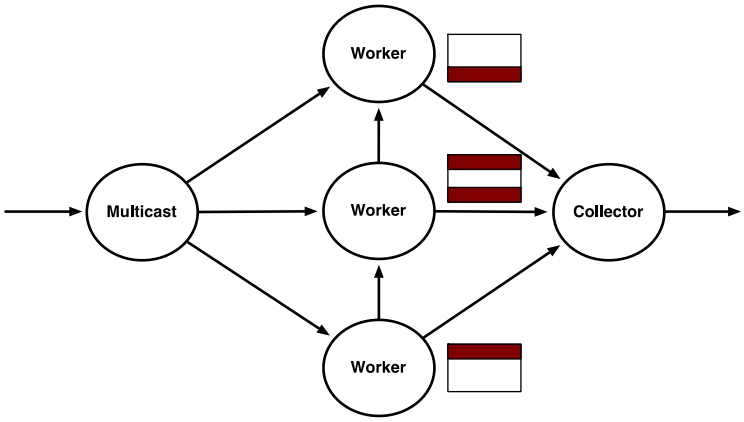
$$T'_{A-3} = \frac{T_{DF-P_2}}{\frac{2}{5}} \times \frac{2}{5} = T_{DF-P_2} = 15 \times 10^3 \tau$$

and the correct optimal parallelism degree of P_3 is:

$$n'^{opt}_3 = \left\lceil \frac{T_{P_3-id}}{T'_{A-3}} \right\rceil = \left\lceil \frac{68 \times 10^3 \tau}{15 \times 10^3 \tau} \right\rceil = 5$$

Since P_3 has a modifiable internal state (the array A), the only feasible parallelization is a *data parallel* one. We can recognize M virtual processors $VP_i = \{A[i], x\}$ for $i = 0, \dots, M-1$. All the virtual processors except VP_{M-1} need to read an element owned by another virtual processor. Therefore, we can recognize a *static fixed stencil*. The element $A[i]$ is updated by executing function H_1 on its old value, on the input integer x and on the received element from the next virtual processor. All the VP_i from $i = 1, \dots, M-2$ execute a send of their actual value $A[i]$ to the previous virtual processor on the ring, receive the value $A[i+1]$ from their successor and compute the function H_1 to update $A[i]$. Then, they execute the reduce phase to compute $y = (A, +)$. The first and the last virtual processors have a different code. VP_0 does not perform any send but only a receive of $A[1]$ from VP_1. VP_{M-1} executes only a send of its $A[M-1]$ without any receive.

The scheme of the actual version with coarse-grained workers is sketched in the following figure in the case of three workers:



The multicast phase has service time:

$$T_{multicast}(1,n) = nT_{send}(1) = 5 \times 10^3\tau < T'_{A-3} = 15 \times 10^3\tau$$

Thus, it is not a bottleneck.

As a simple solution we design a reduce executed in parallel by each worker on its partition, then the partial results are transmitted to the collector which executes the global reduce serially with a $O(n)$ latency. The service time of a generic worker is:

$$T_{w-id} = T_{send}(1) + \frac{M}{n}68\tau \sim \frac{68}{n} \times 10^3\tau = 13.6 \times 10^3\tau$$

The impact of the communication of one word is negligible and the send to the collector for the local reduce result can be easily overlapped with the internal calculation owing to the presence of a communication processor.

The ideal service time of the collector consists in executing four additions to compute the global reduce result and to perform a send of one word y which can not be overlapped. Therefore, we have:

$$T_{coll-id} = T_{send}(1) \sim 10^3\tau$$

which is not a bottleneck. Therefore this parallelization is able to remove the bottleneck in P_3.

- *Evaluation of the parallelized system*

We report in the following table the ideal service times, the effective ones and the relative efficiency of module of the system.

	Ideal TS	***Effective TS***	***Efficiency***
$\boldsymbol{P_0^{(1)}}$	$5 \times 10^3\tau$	$6 \times 10^3\tau$	0.83
$\boldsymbol{P_1^{(1)}}$	$7 \times 10^3\tau$	$30 \times 10^3\tau$	0.023
$\boldsymbol{P_2^{(3)}}$	$15 \times 10^3\tau$	$15 \times 10^3\tau$	1
$\boldsymbol{P_3^{(5)}}$	$13.6 \times 10^3\tau$	$15 \times 10^3\tau$	0.91
$\boldsymbol{\Sigma}$	$5 \times 10^3\tau$	$6 \times 10^3\tau$	0.83

b)

i) "*To achieve the maximum bandwidth it is necessary to perfectly balance the workload between P_1, P_2 and P_3*"

The sentence is *false* in general. As demonstrated in the exercise, to achieve the maximum bandwidth $1/T_A$ is just sufficient that the utilization factors of all P_1, P_2 and P_3 are all less than one. Instead, if a bottleneck exists, then the effective service time of the system becomes greater than T_A.

ii) "*The ideal parallelism degree of any P_i can be determined independently from the performance of the other P_is"*.

The sentence is *false*. To determine the optimal parallelism degree of any P_i we need to find the correct inter-arrival time which may depend on the presence of a bottleneck in the graph. This concept has been applied in the exercise. The optimal parallelism degree of P_3 takes into account that P_2 has been parallelized although the bottleneck still exists in that module. For this reason the optimal parallelism degree of P_3 is smaller than the one that would be necessary if the bottleneck in P_2 has been completely removed.