Master Program in Computer Science and Networking

High Performance Computing Systems and Enabling Platforms

a.a. 2009-10

terzo appello, 13 July 2010

The answer can be written in English or in Italian.

Write name and surname at the top of the first page of each foolscap.

Present the answer in a legible form (writing in pencil is accepted, if preferred).

Consultation of any kind of didactic material or notes is not permitted.

Don't submit possible rough copies.

Results and date of oral exam will be published in the course page as soon as possible.

Question 1

a) Consider the under-load memory access latency in all-cache multiprocessor architectures.

For all the parameters, that have influence on such latency, discuss their qualitative impact. Where possible, introduce some quantitative considerations, in particular under which conditions the impact of each parameters is more or less significant.

b) Consider an all-cache NUMA architecture with 32 nodes connected by a toroidal ring with wormohole flow control and 32-bit links.

Show the architecture of a generic node based on D-RISC pipelined CPU.

Evaluate the base memory access latency.

Explain qualitatively, yet formally, why the under-load memory access latency decreases by replacing the single ring with *m* independent toroidal rings.

Question 2

Consider a sequential program to compute the number of occurrences of an integer *x* in an integer array *A[N]*.

Compare the completion time T_{c1} on a D-RISC pipelined CPU and the completion time T_{c2} on a pipelined machine that directly provides the required computation through a single primitive assembler instruction:

OCCURRENCES RA, RN, Rx, Rc

where *RA, RN, Rx, Rc* are addresses of general registers containing respectively the base logical address of *A*, the value of *N*, the value of *x*, and the value of the result.

Assumptions for the sake of simplicity, cercerning the completion time evaluation only: *1)* neglect the effect of cache faults, *2)* the probability that an element of *A* is equal to *x* is negligible.

Explain the conceptual reasons leading to the difference of T_{c1} and T_{c2}.

Solution outline

(to be expanded properly)

Question 1

a) The under-load memory access latency is evaluated as the server response time R_Q in a queueing system with a generic memory macromodule as a server and an average number of p client processors depending on the parallel program structure and allocation: **sect. 4.3, slides 29 – 39.**

All the curves of slides 33 – 39 can be motivated qualitatively in terms of the queueing system, e.g. as p increases, the interarrival time …, thus the utilization factor …, and the response time ….

The relative impact of the parameters is discussed in slides 33 – 39.

b) The node architecture includes an interface unit W connected to the network node SW, which is in charge of routing and flow control functionalities.

The average distance is $d = N/2 = 16$ hops. The base memory access latency is evaluated according to the scheme of sect. 4.2, slides 23 – 28 (calculations to be adapted to the specific case).

From the point of view of the under-load memory access latency evaluation, *with this kind of interconnection network* the server in the modeled queueing system includes not only a generic memory macromodule, but mainly the network itself (while with other networks, e.g. fat trees, the network impact is negligible, such assumption cannot be done with a ring-based network).

Replacing the single ring with m independent rings, the bandwidth of the server (network) increases by m times, so its service time decreases by m times, thus the utilization factor decreases and the mean queueing time W_Q decreases and so $R_Q = W_Q + L_s$ decreases, though the server latency L_s remains *constant* and may have a significant impact.

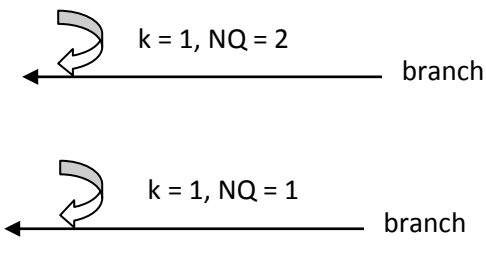
Question 2

The pseudo-code of the required computation is:

```
input int A[N], int x;
int c = 0;
for (i= 0; i < N; i++)
        if (A[i] = = x)
                c++;
output c
```

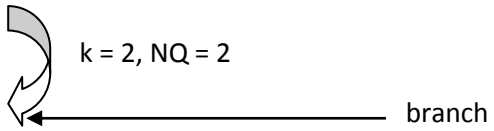
For a D-RISC machine the non-optimized compiled code is of the kind:

```
LOOP: LOAD   RA, Ri, Ra
      IF < >   Ra, Rx, CONT
      INCR   Rc
CONT: INCR   Ri
      IF <   Ri, RN, LOOP
      END
```



For a pipelined CPU, it can be optimized in the following way:

```
        LOAD   RA, Ri, Ra
LOOP:   INCR   Ri
        IF < >   Ra, Rx, CONT
        INCR   Rc
CONT:   IF < … Ri, RN, LOOP, delayed_branch
        LOAD   RA, Ri, Ra
        END
```



Since INCR Rc is executed with negligible probability, the cost model gives the following result:

$$\lambda = 0$$

$$d_2 = 1/4,\ N_Q = 2,\ \text{thus } \Delta = t/4$$

$$\text{service time per instruction: } T = 5t/4$$

$$\text{completion time: } \mathbf{T_{c1} = 4NT = 5Nt = 10N\tau}$$

In the other pipelined machine, the instruction

OCCURRENCES RA, RN, Rx, Rc

is executed as follows: once received by IM, IU generates a stream of the N logical addresses of A, one per time interval *t*, including the index increment and loop control. These addresses are used by DM to read the A elements that form a stream sent to EU. For each value received by DM, EU compares it with *x* and possibly increases *c* in the same clock cycle

For large *N* (>> 4):

$$\mathbf{T_{c2} = Nt = 2N\tau}$$

The reasons for the difference of T_{c1} and T_{c2} lay in the characteristics of the firmware level implementation compared to the assembler level implementation. That is, in the machine having the primitive OCCURRENCES instruction:

- there is parallelism in EU microinstructions: equality test and increment are done in a single clock cycle (the assumption about the negligible probability of incrementing *c* is of no utility in this case),
- there is parallelism in IU microinstructions: incrementing index *i* and testing it for loop control are done in a single clock cycle,
- because of the local execution of sequences of tests and arithmetic operations in IU and EU, there are no logical dependencies nor branches degradations.

Architetture Parallele e Distribuite

Domanda 1

Vedi SPA.

Domanda 2

Un processo P operante su stream è così definito:

- gli elementi dello stream d'ingresso sono coppie *(op, A)* dove *op* può assumere i valori *0, 1, 2, 3*, ed *A* è un array di *M* interi, con *M* dell'ordine delle decine di migliaia;
- a seconda del valore *op* (*op= 0, 1, 2, 3*), P esegue una computazione del tipo:

$$x = 0;$$
$$i = 0 .. M-1$$
$$x = F_{op}(x, A[i]);$$

- il risultato intero è inviato sullo stream di uscita;
- i tempi medi di elaborazione delle funzioni F_0, F_1, F_2, F_3 sono rispettivamente uguali a *10t, 20t, 30t, 40t*, con *t* unità di tempo opportuna. Le probabilità che *op = 0, 1, 2, 3* sono rispettivamente uguali a *1/10, 2/10, 3/10, 4/10*.

Il tempo di interarrivo a P è uguale a *Mt*.

Il modello dei costi delle comunicazioni tra processi, sovrapponibili al calcolo interno, è caratterizzato da $T_{setup} = t/10$ e $T_{trasm} = t/100$.

P viene parallelizzato in una prima versione P0 secondo il paradigma con *partizionamento funzionale*.

Si vuole ulteriormente parallelizzare P0 allo scopi di ottenere una versione P1 in grado di raggiungere la stessa banda che avrebbe una parallelizzazione P2 di P secondo il paradigma farm con grado di parallelismo ottimo.

Spiegare in cosa consiste la versione P1, determinarne il grado di parallelismo complessivo, e confrontarla con la versione P2 da diversi punti di vista.

Supponendo di ridurre il tempo di interarrivo a *Mt/k* (*k* intero > 0), spiegare come si modifica la versione P1 in modo da raggiungere la massima banda.

Si verifichi che in nessuna versione le comunicazioni hanno impatto sulle prestazioni.

La versione P0 consta di un processo distributore D, quattro worker specializzati, ed un processo collettore. Il tempo di servizio dei quattro worker è rispettivamente: *10Mt, 20Mt, 30Mt, 40Mt*. Il loro tempo di interarrivo vale rispettivamente: *10Mt, 10Mt/2, 10Mt/3, 10Mt/4*, per cui solo il primo worker non è collo di bottiglia.

La versione farm P2 ha grado di parallelismo ottimo *30* (*30Mt* è il tempo di servizio di P), con tempo di servizio uguale al tempo di interarrivo *Mt*.

La versione P1 si ottiene dalla P0 parallelizzando *ogni worker* con una soluzione *data-parallel di tipo pipeline con loop unfolding*. Il numero di stadi necessari per eliminare tutti i colli di bottiglia è, rispettivamente: *1, 4, 9, 16*, complessivamente *30* come nella soluzione farm. Il tempo di servizio così ottenuto è *Mt*. Si osservi che ogni worker deve effettuare una opportuna scatter dell'array ricevuto da D, ma questa non comporta la presenza di ulteriori colli di bottiglia. Rispetto alla soluzione farm si ha una occupazione di memoria per nodo ed una latenza inferiore (valutare).

Riducendo il tempo di interarrivo a *Mt/k*, la versione P1 raggiunge la massima banda aumentando di *k* volte il grado di parallelismo dei worker, rispettivamente: *k, 4k, 9k, 16k*, fino ad un valore di *k* per cui le comunicazioni divengono collo di bottiglia.