Master Program in Computer Science and Networking

High Performance Computing Systems and Enabling Platforms

a.a. 2009-10

primo appello, 9 June 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 by the student).

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 course page as soon as possible.

Question 1

Consider a computation consisting in one locking critical section, to be executed on an all-cache multiprocessor architecture. The critical section operates on a shared data structure S associated to a lock semaphore X.

Describe the computation steps, with regard to the shared memory accesses, in two versions in which cache coherence is implemented, respectively: *1)* according to an automatic, invalidation-based, approach, or *2)* according to a non-automatic approach.

Evaluate possible differences in the completion time of the two versions.

In general (i.e., independently of this example), explain pros and cons of the non-automatic approach, and discuss in which cases, or under which conditions, this approach is able to achieve a good trade-off between machine-independent programmability and performance of parallel programs.

Question 2

Consider a *k-ary 2-cube* interconnection network with deterministic routing and wormhole flow control (one-word links, one-word flits).

Describe the structure and behaviour of a Network Node, acting as a network interface unit of the main processing node (i.e., only communication functionalities are implemented in the Network Node).

The description should clearly show whether it is feasible to achieve the maximum bidirectional bandwidth, provided that the proper traffic conditions hold.

Solution outline

Question 1

The solution is explicitly and fully described in the course notes (multiprocessor part, section 4.3):

- Slides 47, 48, 49 contain the description of the automatic approach in presence of locking.
- The non-automatic approach can be easily derived by the implementation of any critical section in the example of send run-time support: 50, 52, 53.
- Exercize 1, slide 54, is about the completion time difference.
- Pros and cons: slide 46.
- Slide 51 is the outline of the trade-off discussion. The crucial point consists in the features of the application development approach: IF parallel programs are expressed in, or compiled into, a formalism which is really machine independent (e.g., a message passing concurrent language with typed channels; on the contrary, a generic set of system calls has not such a feature), THEN all the architecture dependent characteristics are dealt with in the run-time support. In our case, this means that the run-time support designer, NOT the application programmer, must be aware of the non automatic approach characteristics at the firmware and assembler level.

Comment: the pros/cons and trade-off question is a typical "benchmark" for the student preparation according to the methodology and approach required by SPA. As explained carefully during the lectures, the ability and competence to discuss such kind of concepts and interrelations ***is*** of fundamental importance for this course. That is, the "course big picture" is not merely a drawing: it contains all the concepts, technologies and their interrelations that the students has to know and must be able "to manage".

Question 2

Definition of k-ary n-cubes: slide 10 of section 4.2.

Wormhole and wormhole switching unit: slides16, 17, 18, 19.

Similar exercise (for generalized fat tree): slide 20. The only difference here is that the network node has four bidirectional links, thus it consists of two distinct units, each with four unidirectional links. Each output link has an interface with a 4-input internal multiplexer.

The latency is one clock cycle per transmitted flit. The maximum bandwidth is 4 flits per clock cycle: provided that messages from distinct input links are routed onto distinct output links (no conflicts), the network node is able to achieve the maximum bandwidth, owing to the *nondeterministic behavior in microinstructions* able to start the transmission of a new message during *any* clock cycle, even when other messages are in transit.

Comment: besides the fact that a very similar exercise has proposed during the course and the student should have been done and discussed it, this question is another typical "benchmark". Several parts of the course require ability to reason at the firmware level in terms of feasible structures and achievable latencies and service times.

Architetture Parallele e Distribuite

Domanda 1

Vedi Question 1 SPA.

Domanda 2

Un processo P incapsula due array di interi A[N] e B[N], con $N = 10^6$. P opera su uno stream di interi x e, per ogni x, calcola

$c = x;$

$\forall i = 0 \ldots N-1$

$\quad c = f(c, A[i], B[i])$

Il valore di c così ottenuto è inviato sullo stream di uscita.

La funzione f ha tempo di elaborazione distribuito uniformemente tra 0 e 20τ. I parametri delle comunicazioni valgono $T_{setup} = 10^3\tau$, $T_{trasm} = 10^2\tau$. Il tempo di interarrivo vale mediamente $10^5\tau$.

Parallelizzare P, in modo da ottenere la massima banda possibile, in due versioni distinte che risultino diverse nell'occupazione di memoria per nodo.

Confrontare ulteriormente le due versioni supponendo di eseguire il programma parallelo

- su una architettura SMP con 128 nodi
- su una architettura NUMA con 128 nodi.

Gli array A, B sono staticamente incapsulati in P, quindi saranno staticamente incapsulati (replicati o partizionati) nei processi delle versioni parallele.

Poiché non sono note proprietà particolari della funzione f, per ogni x gli N passi sono ordinati tra loro sequenzialmente. Dunque non può essere usata alcuna forma data-parallel, eccetto, in virtù del funzionamento su stream, quella strutturata a *pipeline con loop unfolding*, con x entrante nel primo stadio e gli stadi che si passano valori intermedi di c. Nella definizione a processori virtuali, gli stadi sono N ognuno che incapsula un elemento di A e uno di B; con grado di parallelismo n, ogni stadio incapsula una partizione di A e una di B ampie N/n interi.

L'altra versione è un *farm*, con A e B interamente replicati in tutti gli n worker. L'occupazione di memoria per nodo è dunque $2N$ parole, contro le $2N/n$ parole dell'altra versione.

In entrambi i casi, il grado di parallelismo ottimo vale $n = 100$. Esso viene effettivamente sfruttato, in quanto la latenza di comunicazione ($T_{send}(1)$) è trascurabile rispetto al tempo di calcolo degli stadi o dei worker e, nella versione farm, l'emettitore (il collettore) non è collo di bottiglia in quanto avente tempo di servizio $T_{send}(1)$ con un supporto zero-copy alle comunicazioni.

La soluzione pipeline, per contro, non garantisce il bilanciamento del carico, che invece è proprio della soluzione farm; il problema è sentito in questa applicazione data la sensibile varianza del tempo di calcolo.

Su una architettura NUMA, nella versione pipeline unfolding, *ognuno* dei 100 nodi utilizzati ha allocate staticamente nelle rispettive memorie locali le due partizioni di A e B, per complessive circa 20K parole. Tali strutture dati, una volta accedute per la prima volta, possono essere allocate nella cache primaria per tutta durata del programma parallelo (non deallocare), per cui la memoria condivisa viene utilizzata pochissimo, eccetto che per le comunicazioni. Con la soluzione farm ogni nodo riserva circa 2M parole alle repliche di A e B, che dunque, data la dimensione, non possono essere allocate permanentemente nella cache primaria, al più in quella secondaria. Di conseguenza, le prestazioni della soluzione pipeline tendono ad essere migliori, pur in presenza di un certo sbilanciamento del carico.

Su una architettura SMP, le partizioni di A e B associate agli stadi del pipeline, o le repliche di A e B associate ai worker del farm, sono allocate nella memoria condivisa equidistante. I conflitti per l'accesso ai blocchi di tali strutture (anche prescindendo dalla latenza di accesso a vuoto) degradano le prestazioni rispetto al caso NUMA. Anche in questo caso, comunque, la soluzione pipeline beneficia del fatto che, una volta accedute per la prima volta, le partizioni possono risiedere nella cache primaria per tutta la durata del programma parallelo (a meno di riallocazione dei nodi "anonimi", che però in questo programma non ha senso imporre o è un evento raro), mentre per la soluzione farm può al più essere sfruttata la cache secondaria (se non condivisa).