Architettura degli Elaboratori, 2007-08

Esercitazione 5 di verifica

Soluzione: entro mercoledì 5 dicembre

Domanda 1

Si considerino i seguenti due programmi:

- P1: opera su tre array A[N], B[N], C[N] di interi, con $N = 4K$, e calcola la seguente funzione:

$$\forall\, i = 0 \,..\, N-1 : C_i = min\,(A_i, B_i)$$

- P2:opera su tre array A[N], B[N], C[N][N] di interi, con $N = 4K$, e calcola la seguente funzione:

$$\forall\, i, j = 0 \,..\, N-1 : C_{i,j} = min\,(A_i, B_j)$$

I programmi vengono eseguiti su una architettura con le seguenti caratteristiche:

1. la macchina assembler è D-RISC;
2. la CPU, con ciclo di clock $\tau = 0{,}25$ nsec, dispone di cache primaria operante su domanda, completamente associativa, capacità di 16K parole, blocchi di 8 parole, scritture con il metodo Write-Through. Non è presente cache secondaria;
3. la memoria principale è interallacciata con 4 moduli e ciclo di clock uguale a 200 τ. I collegamenti inter-chip hanno latenza di trasmissione di 19 τ.

Per i due programmi:

a) spiegare come si caratterizzano dal punto di vista della località e del riuso;

b) determinare l'insieme di lavoro e il numero di cache fault del programma. Spiegare *se* per garantire la presenza di tale insieme di lavoro è necessario intervenire a tempo di compilazione, oppure a tempo di esecuzione, ed eventualmente *come*;

c) sapendo che la probabilità che un elemento di A sia maggiore di un elemento di B è uguale a ¼, valutare il tempo di completamento e la performance.

Inoltre:

d) nel caso che la cache primaria sia associativa su insiemi, con

- 2 blocchi per insieme,
- 4 blocchi per insieme,

spiegare se le prestazioni cambiano rispetto al caso precedente;

e) nel caso che sia supportata anche la modalità di funzionamento con prefetching, spiegare *se* e *come* questa può essere sfruttata per aumentare le prestazioni;

f) nel caso che nell'architettura venga introdotta una cache secondaria, spiegare se le seguenti affermazioni sono vere, false o vere sotto determinate condizioni:

i. i suoi blocchi sono ampi 8 parole;

ii. i suoi blocchi sono ampi più di 8 parole;

iii. i programmi devono essere ricompilati.

Soluzione

a) Per le domande *a)* e *b)* è sufficiente ragionare sui programmi scritti in pseudo-codice, come in effetti fa (ovviamente) il compilatore:

P1::

```
for (i = 0; i < N; i++)
        C[i] = max(A[i], B[i])
```

P2::

```
for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
                C[i][j]= max(A[i], B[j])
```

Nel caso di P1, si ha località per quanto riguarda il codice e i tre array, mentre il riuso si applica solo alla parte di codice che costituisce il corpo del loop.

Nel caso di P2, oltre che quanto detto per P1, si ha riuso sull'intero array B.

Il compilatore ricava le proprietà suddette sugli array, sia per P1 che per P2, dall'analisi del codice, esaminando la variazione degli indici: in entrambi i casi, per ogni *i* il prossimo indice ad essere generato è *i + 1*; in P2, per ogni *i* vengono generati tutti i possibili valori di *j*, quindi *tutto B è riusato per ogni i*.

b) Dall'analisi del punto *a)* siamo in grado di affermare che, essendo la cache *su domanda*, l'insieme di lavoro di P1 è costituito da 3 blocchi per i dati, e precisamente *un blocco per A, uno per B e uno per C*, e da *tanti blocchi per quanti sono necessari a contenere il codice del loop* (presumibilmente uno: da verificare al punto *c)*). Il compilatore ricava questa proprietà dall'analisi del codice, verificando quali oggetti (codice del loop, A_i, B_i, C_i) sono necessari *contemporaneamente*, e quindi è necessario che siano allocati contemporaneamente in cache per *minimizzare* la probabilità di fault. Quindi, il numero di fault è dato da:

$$N_{fault} = 2\,\frac{N}{\sigma}$$

valutando, come di regola, solo il numero di fault che danno luogo a trasferimento di blocco (ciò non avviene per C). Si noti che, con una cache su domanda, per P1 la penalità dovuta ai fault è dello stesso ordine di grandezza (ordine di *N*) del tempo di completamento ideale, per cui l'efficienza relativa della cache non potrà avvicinarsi a uno.

Con lo stesso ragionamento, il compilatore ricava che l'insieme di lavoro è costituito dai *blocchi di codice del doppio loop* (presumibilmente due), *da un blocco di A, un blocco di C e da tutti i blocchi di B*, quindi, *se è possibile garantire la presenza in cache dell'insieme di lavoro,* si ha:

$$N_{fault} = 2\,\frac{N}{\sigma}$$

in particolare, se è possibile garantire che tutto B rimanga allocato in cache una volta utilizzato durante la prima iterazione del loop più esterno (per *i = 0*). Ovviamente, il compilatore, conoscendo le caratteristiche dell'architettura, è in grado di verificare che *la dimensione dell'insieme di lavoro è minore della capacità della cache*.

La penalità dovuta ai fault è, quindi, di ordine N, invece che N^2, mentre il tempo di completamento ideale è dell'ordine di N^2: quindi per P2, riuscire a mantenere l'insieme di lavoro in cache rende trascurabile l'impatto dei fault e l'efficienza relativa della cache praticamente uguale a uno.

In entrambi i programmi, mantenere in cache i blocchi di codice può essere garantito dall'architettura firmware (su questi blocchi, l'unità cache può applicare per default anche la tecnica del prefetching). Inoltre, per P2, allo scopo di mantenere tutto B in cache, una volta utilizzato per $i = 0$, occorre annotare le istruzioni di LOAD su B con l'opzione *non_deallocare*.

c) Vediamo la compilazione dei due programmi in D-RISC, limitandoci per P2 al loop più interno (ripetuto N^2 volte, rispetto al loop più esterno ripetuto N volte):

P1::

```
LOOP:    LOAD   RA, Ri, Ra
         LOAD   RB, Ri, Rb
         IF >   Ra, Rb, THEN
         STORE  RC, Ri, Ra
         GOTO   CONT
THEN:    STORE  RC, Ri, Rb
CONT: INCR  Ri
         IF <   Ri, RN, LOOP
         END
```

Si verifica che il loop consta di 8 istruzioni, quindi è contenuto in un solo blocco di cache (complessivamente, i fault sulle istruzioni sono due).

P2::

```
…
LOOP2:   LOAD, RB, Rj, Rb, non_deallocare
         IF >   Ra, Rb, THEN
         STORE  RCriga, Ri, Ra
         GOTO   CONT
THEN:    STORE  RCriga, Ri, Rb
CONT: INCR  Rj
         IF <   Rj, RN, LOOP2
…
```

Per P1 si ha, con $p = 1/4$:

$$T_{c\text{-}id\text{-}1} = N \left[n_{iter\text{-}1}\, T_{ch} + 3\, T_{ex\text{-}LD} + 2\, T_{ex\text{-}IF} + T_{ex\text{-}INCR} + (1-p)\, T_{ex\text{-}GOTO} \right]$$

dove

$$n_{iter}\text{-}1 = 7 - p$$

Per P2:

$$T_{c\text{-}id\text{-}2} \sim N^2 \left[n_{iter\text{-}2}\, T_{ch} + 2\, T_{ex\text{-}LD} + 2\, T_{ex\text{-}IF} + T_{ex\text{-}INCR} + (1-p)\, T_{ex\text{-}GOTO} \right]$$

dove

$$n_{iter\text{-}2} = 6 - p$$

I valori si ricavano, come nel,'Esercitazione 4, sostituendo le usuali espressioni dei tempi di elaborazione delle varie istruzioni, e ponendo il tempo di accesso in memoria

$$t_a = t_c = 3\tau$$

per una cache completamente associativa.

Il tempo medio di trasferimento di un blocco dalla memoria principale alla memoria cache è dato da:

$$T_{trasf} = \frac{\sigma}{m}\,\tau_M + 2\,T_{tr} + m\,\tau$$

che, moltiplicato per N_{fault}, fornisce in entrambi i casi la penalità T_{fault} dovuta ai fault di cache. Si verifichi che, visti gli ordini di grandezza, l'efficienza relativa della cache per P2 è data da

$$\varepsilon_2 \sim 1$$

mentre per P1 risulterà sostanzialmente più bassa (da calcolare).

Per la performance si ragiona come nell'Esercitazione 4.

Per avere una valutazione rigorosa, occorre verificare se le *scritture in memoria principale*, con il metodo *Write-Through*, impattano o meno sule prestazioni. Questo si verifica confrontando la banda delle richieste di scrittura da parte del processore:

$$B_{rich} = \frac{1}{n_{iter}\,T}$$

con la banda offerta dalla memoria:

$$B_M = \frac{m}{\tau_M}$$

Se la prima è minore della seconda, allora le scritture non hanno impatto sulle prestazioni, in quanto la memoria interallacciata è in grado di smaltire tutto il carico di richieste che, ricordiamo, avvengono a blocchi di indirizzi consecutivi (quindi, 4 scritture possono avvenire simultaneamente). Altrimenti, il tempo medio di completamento di una iterazione va assunto uguale a $1/B_M$ (da calcolare per P1 e P2).

d) In P1, con il metodo completamente associativo è assicurato che l'insieme di lavoro è sempre presente in cache. Con il metodo associativo su insiemi, esiste un probabilità non nulla che blocchi omologhi di A, B; C corrispondano allo stesso insieme, in quanto non è dato fare ipotesi sull'allocazione della memoria principale. In una tale situazione, avendo 2 blocchi per insieme non è possibile farli risiedere contemporaneamente in cache, quindi non sarebbe possibile soddisfare l'insieme di lavoro. Questo non avviene disponendo di 4 blocchi per insieme.

In P2, nel caso peggiore può accadere che corrispondano allo stesso insieme blocchi omologhi di A e C ed uno dei blocchi di B; dunque, con 2 blocchi si avrebbe un aumento del numero di fault su A e C.

In tutte queste considerazioni, si può assumere che i blocchi di codice non vadano a collidere con i blocchi di dati, in quanto, tipicamente, la cache è suddivisa in una parte istruzioni ed una parte dati.

e) Il compilatore verifica, in P1, che il prefetching è applicabile, sempre per la proprietà che per ogni i viene generato $i+1$. Annotando le istruzioni di LOAD e STORE con l'opzione prefetching, l'unità cache provvederà a iniziare i trasferimenti di un blocchi di A, B, C appena concluso il trasferimento del blocco precedente, di fatto annullando la probabilità di fault. Questa strategia non

avrebbe comunque effetto per P2, visto che l'efficienza relativa è già molto vicina al massimo assoluto: il compilatore, avendo prima verificato che B gode della proprietà del riuso, deciderà di non applicare prefetching.

f) In presenza di una cache secondaria C2, detta C1 la cache primaria:

i, ii) l'ampiezza dei blocchi della gerarchia M-C2 è molto maggiore di quella della gerarchia C2-C1, come avviene sempre salendo di livello nella gerarchia complessiva, ad esempio è dell'ordine della centinaia di parole, in quanto, statisticamente, aumentando la capacità del livello più basso di una gerarchia aumenta il valore di σ che minimizzala probabilità di fault ;

iii) i programmi non devono essere ricompilati: non solo perché un cambiamento tecnologico dei supporti di memoria non può impattare sulla portabilità del codice, ma ***soprattutto*** perché le ottimizzazioni introdotte dal compilatore non cambiano, sia dovute a prefetching, che dovute a non deallocazione di blocchi.