Architettura degli elaboratori

Quinto appello A.A. 2011-2012

Scrivere Nome, Cognome, Matricola, Corso e Programma presentato su ognuno dei fogli consegnati. La bozza di soluzione, i risultati e il calendario degli orali verranno pubblicati sulle pagine web del corso appena disponibili.

Domanda 1 (Tutti)

Si consideri il seguente pseudocodice e se ne fornisca una compilazione in codice D-RISC (A, B e C sono vettori di interi da N posizioni ciascuno, $N=2^k$).

```
sum=0;
for(i=0,i<N,i++) {
        ind = (a[i]+b[i]) % N;
        c[ind]=c[ind]+ind;
        sum = sum + ind;
}
```

Si fornisca il working set del programma e si discutano le caratteristiche relative agli accessi effettuati in memoria: traccia degli indirizzi generati, numero di fault, banda della memoria principale. Si assuma la presenza del solo livello di cache primaria con cache istruzioni e dati separate, associative su insiemi, a due vie, con σ=8 e 1K insiemi ciascuna.

Domanda 2 (tutti)

Si realizzi una unità firmware che calcola il programma descritto dallo pseudocodice della Domanda 1, con le seguenti assunzioni:

- i vettori A a B si trovano in una singola memoria esterna, cui si accede mediante interfaccia standard. La memoria ha un tempo di accesso pari a t_M,
- il vettore C si trova in una memoria interna all'unità, con tempo di accesso pari a t_a,
- il calcolo parte all'arrivo di un segnale di pura sincronizzazione da una unità U_1,
- il valore calcolato per sum viene inviato ad una unità U_2.
- non è necessario testare or(ESITO) sulle operazioni in memoria.

Si forniscano microprogramma, ciclo di clock, tempo di servizio per l'unità e numero di ALU incluse nella PO.

Domanda 3 (NEW, OLD-0)

Discutere l'impatto sul sottosistema di memoria di una architettura superscalare ad N vie.

Domanda 3 (OLD-1)

Valutare il tempo necessario al completamento di un'operazione di I/O che, utilizzando MM I/O, legge la coppia di coordinate relative alla posizione di un dispositivo di puntamento tipo mouse, cioè una coppia di interi da 32 bit certamente presenti nella memoria interna dell'unità di I/O e li restituisce nella coppia di registri R50 e R51.

Bozza di soluzione

Domanda 1

Compilazione in Assembler D-RISC, con opportune allocazioni e inizializzazioni dei registri generali:

```
for:  LOAD   Ra, Ri, Rai
      LOAD   Rb, Ri, Rbi
      ADD    Rai, Rbi, Rab
      MOD    Rab, RN, Rind
      LOAD   Rc, Rind, Rcind
      ADD    Rind, Rcind, Rcind
      STORE  Rc, Rind, Rcind
      ADD    Rsum, Rind, Rsum
      INCR   Ri
      IF<    Ri, RN, for
      END
```

Il working set del programma comprende:

- i blocchi di cache che contengono il codice (due, dato che σ=8, qui abbiamo sia località che riuso)
- un blocco di cache contenente i valori di *a* e uno per quelli di *b* (località nell'accesso ma non riuso)
- il vettore *c* (possibile riuso ma senza località). Poiché l'indirizzo di *c* è il risultato di un calcolo, non è possibile fare analisi statiche precise circa l'entità del riuso, cioè circa la riutilizzazione di un blocco già riferito. In ogni caso, a condizione che la capacità della cache (16K) sia maggiore di N, richiediamo di non deallocare i blocchi di *c*:

  LOAD R_c, R_{ind}, R_{cind}, non_deallocare

La traccia degli indirizzi logici generati verso il sistema di memoria è

> INDIRIZZO(LOAD1), A[0], INDIRIZZO(LOAD2), B[0], INDIRIZZO(ADD1), INDIRIZZO(MOD), INDIRIZZO(LOAD3), C[IND], INDIRIZZO(ADD2), INDIRIZZO(STORE), INDIRIZZO(ADD3), INDIRIZZO(INC), INDIRIZZO(IF<), INDIRIZZO(LOAD1) …

Per capire quali accessi vengono effettivamente richiesti alla cache dobbiamo considerare la traduzione degli indirizzi effettuata in MMU. La traccia degli indirizzi fisici presentati alla cache sarà quindi (in assenza di fault di pagina):

> μ(INDIRIZZO(LOAD1)), μ(A[0]), μ(INDIRIZZO(LOAD2)), μ(B[0]), μ(INDIRIZZO(ADD1)), ...

Avendo a disposizione una cache separata per dati e codice, il codice produrrà i soli fault per il caricamento iniziale (sono due, dal momento che la lunghezza del codice è compresa fra σ e 2σ). Per quanto riguarda i dati, a seconda di come sono mappate in memoria principale le pagine logiche che contengono i tre vettori, potremmo trovarci nella situazione che le pagine fisiche utilizzate mappano sullo stesso insieme di blocchi di cache. In questo caso potrebbe darsi la situazione che l'accesso a *c[ind]* provochi la de-allocazione del blocco di cache che contiene *a[i]* o *b[i]*, incrementando il numero di fault rispetto a $2N/\sigma$. Tuttavia, l'accesso al vettore *c* avviene

in modo dipendente dal risultato della somma fra *a[i]* e *b[i]*, quindi la probabilità che la porzione dell'indirizzo fisico che determina l'insieme di cache in cui andare a cercare il blocco di *c* coincida con l'indirizzo in cui sono mappati sia il blocco che contiene *a[i]* che quello che contiene *b[i]* è bassa. Dovrebbe accadere infatti che per

Indirizzo(A[I]) =	TAG (19 bit)	#SET (10 bit)	OFFSET (3 bit)
Indirizzo(B[I]) =	TAG'	#SET'	OFFSET'
Indirizzo(C[ind]) =	TAG"	#SET"	OFFSET"

i campi #SET, #SET' e #SET" abbiano tutti lo stesso valore. Dovremmo quindi avere un numero di fault complessivi pari a

- 2 (codice)
- N/σ (scorrimento di *a*)
- N/σ (scorrimento di *b*)
- al più N/σ (aggiornamento di *c*)

Verifichiamo se la banda della memoria principale è sufficiente a sopportare il carico di richieste di scrittura negli elementi di *c* senza degradare il tempo di completamento.

Il tempo di servizio per generare richieste di scrittura alla memoria è dato dal tempo di completamento di una singola iterazione del loop:

$10T_{ch} + 4\ Tex_{LD/ST} + 4Tex_{ALcorte} + 1\ Tex_{ALlunghe} + 1Tex_{if} =$

$10\ (2\tau + t_a) + 4\ (2\ \tau + t_a) + 4\ (\tau) + 1\ (50\ \tau) + 1\ (2\ \tau) =$

$84\ \tau + 14\ t_a$

La banda massima della memoria principale è data da τ_M/m. Quindi, nel caso più favorevole, la condizione richiesta è:

$$\frac{m}{\tau_M} \geq \frac{1}{84\ \tau + 14\ t_a}$$

In realtà, poiché in generale gli accessi agli elementi di *c* non avvengono a indirizzi consecutivi, è possibile che si verifichino conflitti di richieste consecutive per l'accesso allo stesso modulo. Dall'analisi della banda di una memoria interallacciata (sez. 3.4.3 della dispensa su ILP) risulta che la banda non è proporzionale a *m*, bensì a circa la radice di *m*: questa è la modifica da apportare alla relazione precedente per renderla più realistica. *Agli effetti del programma svolto, questo tipo di analisi vale per NEW e OLD-0.*

Domanda 2

Assumendo che

- l'interfaccia verso U_1 sia costituita da RDY_{sync} e ACK_{sync}
- l'interfaccia verso U_2 sia costituita da RDY_{out}, ACK_{out} e registro OUT
- che BASEA e BASEB contengano gli indirizzi base in memoria per i vettori A e B
- che I sia un registro di *(k+1)* bit, con $k = log_2 N$.

possiamo scrivere il microprogramma dell'unità come mostrato nel seguito.

Il valore di *DATAIN + AI* che compare nel microprogramma consiste dei *k* bit meno significativi del risultato di *DATAIN + AI*. L'ottimizzazione del tempo medio di elaborazione è ottenuta utilizzando tale valore per indirizzare la memoria interna C: rispetto alla soluzione in cui il valore venga prima scritto in un registro e poi utilizzato quest'ultimo per indirizzare C, la soluzione adottata comporta un ciclo di clock più lungo ma un numero di cicli di clock dell'ordine della metà, con un bilancio complessivo sul tempo di elaborazione favorevole. Dato che il contenuto di *DATAIN* è il risultato dell'accesso in memoria esterna, non è possibile una soluzione alternativa che anticipi il calcolo dell'indirizzo di C minimizzando anche il ciclo di clock.

0. ($RDY_{sync} = 0$) nop, 0;

   (= 1) 0 → SUM, 0 → I, reset RDY_{sync}, set ACK_{sync}, BASEA → INDA, BASEB → INDB, BASEA → IND, "read" → OP, set RDY_{outM}, 1

1. ($RDY_{inM} = 0$) nop, 1;

   (= 1) reset RDY_{inM}, DATAIN → AI, INDA + 1 → INDA, I + 1 → I, INDB → IND, "read" → OP, set RDY_{outM}, 2

2. (RDY_{inM}, ACK_{out}, $I_0 = 0 - -$) nop, 2;

   (= 1 – 0) reset RDY_{inM}, C[DATAIN + AI] + DATAIN + AI → C[DATAIN + AI], DATAIN + AI + SUM → SUM, I + 1 → I, INDB + 1 → INDB, , INDA → IND, "read" → OP, set RDY_{outM}, 1;

   (= 1 1 1) reset RDY_{inM}, C[DATAIN + AI] + DATAIN + AI → C[DATAIN + AI], DATAIN + AI + SUM → OUT, reset ACK_{out}, set RDY_{out}, 1;

   (= 1 0 1) reset RDY_{inM}, C[DATAIN + AI] + DATAIN + AI → C[DATAIN + AI], DATAIN + AI + SUM → SUM, 3;

3. ($ACK_{out} = 0$) nop, 3;

   (= 1) SUM → OUT, set RDY_{out}, reset ACK_{out}, 0

La microistruzione 3 può essere eliminata se si effettua l'attesa di ACK_{out} nella stessa microistruzione 2.

Agli effetti della valutazione del ciclo di clock si ha:

$$T_{\omega PO} = 0 \qquad T_{\sigma PC} = T_{\omega PC} = 2t_p$$

$T_{\sigma PO}$ è sicuramente determinato dalle micro-operazioni contenenti

$$C[DATAIN + AI] + DATAIN + AI \rightarrow C[DATAIN + AI]$$

per cui paghiamo

$$t_{alu} \text{ (calcolo indirizzo C)} + t_a \text{ (lettura di C)} + t_{alu} \text{ (somma di IND)}$$

essendo il ritardo di stabilizzazione di C per la scrittura sovrapposto a quello della lettura e somma, per un totale di

$$2\,t_{alu} + t_a$$

Dunque:

$$\tau = 3t_p + 2\ t_{alu} + t_a$$

Per ottenere questo risultato dobbiamo essere in grado di eseguire in parallelo, nella microistruzione 2, tutte le somme:

I + 1, INDB + 1, DATAIN + AI, C[..] + …, … + SUM

e quindi dovremo includere 5 ALU nella PO.

Il tempo di servizio è dato da:

$$T \sim 2N\ (\tau + t_M)$$

Domanda 3 (NEW, OLD-0)

La banda della cache istruzioni deve permettere la lettura di *N* istruzioni consecutive contemporaneamente (istruzione lunga in VLIW): questo si può ottenere con una memoria interallacciata con *N* moduli, oppure con locazioni di *N* parole (parole lunghe).

Qualora non si vieti esplicitamente la presenza di più istruzioni LOAD/STORE nella stessa istruzione lunga, la cache dati debba deve anch'essa permettere, nel caso peggiore, l'esecuzione di *N* letture o scritture contemporaneamente, cosa che si può ottenere con una memoria interallacciata di *m* moduli, con $m \geq N$. Detto *q* il numero medio di LOAD/STORE per istruzione lunga, la condizione per non avere degradazioni di prestazioni dovute a conflitti sugli stessi moduli è data da:

$$B_M \geq q$$

Cioè:

$$m^{0,56} \geq q$$

Domanda 3 (OLD-1)

È sufficiente leggere, utilizzando gli indirizzi su cui è mappata la memoria dell'unità di I/O nello spazio di indirizzamento logico del processo in esecuzione, mediante due LOAD:

LOAD $R_{baseUI/O}$, $R_{offsetx}$, R_{50}

LOAD $R_{baseUI/O}$, $R_{offsety}$, R_{51}

Ovviamente, le letture verranno reindirizzate verso il bus di I/O dalla MMU e quindi costeranno ciascuna il tempo di un trasferimento di una parola dalla unità di I/O al processore.

Le istruzioni di cui sopra possono far parte dello handler di una interruzione dall'unità connessa al mouse, con la quale si comunica il riferimento alla base delle due parole da leggere.

È inoltre richiesta una sincronizzazione tra CPU e unità di I/O:

- può essere organizzata una coda di coppie di locazioni, e quindi trasmesso alla CPU il riferimento alla testa della coda, demandando interamente la gestione della coda all'unità di I/O,
- oppure, se la coppia di parole è sempre la stessa, utilizzare un flag che deve essere settato dallo handler, mediante una STORE, dopo la lettura delle due parole.