Architettura degli Elaboratori

Appello del 10 gennaio 2012

Riportare su tutti i fogli consegnati nome, cognome, numero di matricola, corso A/B, e la sigla NEW (per nuovo ordinamento), oppure OLD-0 (per vecchio ordinamento, nuovo programma), oppure OLD-1 (per vecchio ordinamento, vecchio programma). I risultati verranno pubblicati sulle pagine web del corso/dei docenti appena disponibili.

Domanda 1 (tutti)

a) Una unità di elaborazione U può inviare il risultato A di una operazione esterna all'unità U_0 oppure all'unità U_1 a seconda del valore di un predicato su A. Dire se può essere utilizzata un'unica interfaccia di uscita (includente anche un bit per identificare l'unità destinataria) per inviare tanto a U_0 quanto a U_1, e dimostrare la risposta.

b) Dimostrare che la formula per valutare il ciclo di clock di una unità di elaborazione fornisce un upperbound, e mostrare un esempio in cui il ciclo di clock può essere inferiore a quello valutato con la formula.

Domanda 2 (tutti)

Si consideri un elaboratore con memoria cache. Pronunciarsi sulla verità o meno delle seguenti affermazioni, spiegando la risposta:

a) il riuso dei dati di un programma può essere riconosciuto direttamente dall'unità cache;

b) la progettazione dell'unità cache è diversa a seconda che il sistema preveda o non preveda le ottimizzazioni legate al riuso dei dati.

Domanda 3 (NEW, OLD-0)

Si consideri la seguente computazione

int X[N], Y[N], Z[N];

$\forall i = 0 .. N - 1 : Z[i] = if\ (X[i] = Y[i])\ then\ X[i] * Y[i]\ else\ X[i] / Y[i]$

a) Implementarla in modo ottimizzato e, nell'ipotesi che *prob* $(X[i] \neq Y[i])$ sia trascurabile, valutarne il tempo di completamento, per una CPU D-RISC con architettura superscalare a 2 vie, unità funzionale moltiplicatore/divisore in pipeline a 4 stadi, con cache dati associativa e blocchi di 8 parole, e cache secondaria on chip con tempo di accesso 2τ. Si suppone che *X, Y, Z* siano presenti interamente nella cache secondaria.

b) Come al punto *a)* per un sistema avente macchina assembler D-RISC con in più un'istruzione, rappresentata su due parole, capace di eseguire la seguente computazione su interi:

$$c = if\ (a = b)\ then\ a * b\ else\ a / b$$

dove *a* e *b* denotano valori in memoria e *c* un registro generale.

Domanda 3 (OLD-1)

Una computazione LC contiene i processi S, C_1, .., C_n. Il processo S, ricevendo da un generico C_i un messaggio di valore intero *N*, si procura da una determinata unità di I/O un array di *N* interi. L'unità di I/O opera su blocchi di 4K parole.

a) Mostrare il processo S in LC e spiegare come, nel codice sorgente e nel codice oggetto di S, vengono implementate le interazioni con l'unità di I/O nel caso che questa funzioni solo in Memory Mapped I/O oppure anche in DMA.

b) Spiegare se, ed eventualmente come, il codice sorgente e/o il codice oggetto di S vengono modificati nel caso che esista un processo Driver per interfacciare l'unità di I/O.

Soluzione

Domanda 1 (tutti)

a) La soluzione non è corretta usando una interfaccia a transizione di livello. Infatti, data una coppia di indicatori di interfaccia a transizione di livello tra loro connessi (RDYOUT_mitt, RDYIN_dest), condizione necessaria per la rilevazione della presenza di un messaggio in ingresso è che il contatore modulo 2 facente parte di RDYIN_dest abbia, prima dell'operazione *set* RDYOUT_mitt, lo stesso valore di RDYOUT_mitt. Inviando una sequenza di due messaggi, il primo a U_0 e il secondo a U_1, se il primo messaggio è ricevuto correttamente da U_0, allora U_1 non si può accorgere della presenza del secondo messaggio.

b) Nella formula ognuno dei ritardi èvalutato al valore *massimo in assoluto* per tutto il microprogramma. Questo permette di avere un modello dei costi a bassa complessità, indipendentemente dalle dimensioni del microprogramma, ma il valore del ciclo di clock risultante è in generale maggiore o uguale del ritardo che sarebbe sufficiente per stabilizzare la transizione dello statointerno (PC, PO) di qualunque microistruzione valutata a sé stante. Ad esempio, con il seguente microprogramma:

0. A + B $\rightarrow$ M, 1

1. (zero (M – C) = 0) A $\rightarrow$ D, 0; (zero (A - C) = 1) A $\rightarrow$ E, 0

la somma dei ritardi della funzione delle uscite e della funzione di transizione dello stato interno della PO vale T_{ALU} per entrambe le microistruzioni, mentre applicando la formula tale somma varrebbe $2\ T_{ALU}$.

Domanda 2 (tutti)

a)Falsa. Il riuso di una certa struttura dati è una proprietà globale su tutto il programma, quindi, per definizione, solo un compilatore è in grado di riconoscerla. Il microprogramma dell'unità cache è invece un interprete, quindi capace solo di analizzare ogni singolo accesso o, al più, la storia di una sequenza parziale di accessi (anche nel secondo caso, non si può ugualmente né dedurre la proprietà su tutta l'esecuzione del programma, né se la proprietà sarebbe sfruttabile in relazione alla capacità della memoria cache).

b) Vera. Una volta che a tempo di compilazione è stata rilevata la proprietà di riuso, e che è statoverificatoche la struttura dati può risiedere permanentemente in cache, opportune annotazioni a livello assembler(possibilmente nelle stesse istruzioni con dati in memoria, come le LOAD e STORE in D-RISC) permettono, nell'interprete firmware, di dare informazioni all'unità cache su come deve comportarsi circa l'allocazione/deallocazione dei blocchi ("non_deallocare"). L'unità cache progettata di conseguenza.

Domanda 3 (NEW, OLD-0)

a) Con le opportune assunzioni sull'allocazione e inizializzazione dei registri generali, il codice assembler scalare non ottimizzato è:

```
LOOP: 1.    LOAD   RX, Ri, Rx
      2.    LOAD   RY, Ri, Ry
      3.    IF =   Rx, Ry, THEN
      4.    DIV    Rx, Ry, Rz
      5.    STORE  RZ, Ri, Rz
      6.    GOTO   CONT
THEN: 7.    MUL    Rx, Ry, Rz
      8.    STORE  RZ, Ri, Rz
CONT: 9     INCR   Ri
      10.   IF <Ri, RN, LOOP
      11.   END
```

Per le ipotesi, ci interessa solo la parte di 7 istruzioni includente il ramo THEN.

Il codice scalare contiene le dipendenze logiche IU-EU della 2 sulla 3 ($k = 1$, $N_Q = 2$), della 7 sulla 8 ($k = 1$, $N_Q = 1$, Lpipe = 4) e della 9 sulla 10 ($k = 1$, $N_Q = 1$), mentre non esistono dipendenze EU-EU. Inoltre, le prestazioni sono degradate dai salti nella 3 (*nota:* non è consentito scrivere una versione equivalente con il predicato invertito, in quanto il compilatore non è a conoscenza della probabilità del predicato, conoscenza che qui viene usata solo a posteriori) e nella 10.

Un codice scalare ottimizzato (con base dell'array Z decrementata di uno) è il seguente:

```
LOOP: 1.    LOAD   RX, Ri, Rx
      2.    LOAD   RY, Ri, Ry
      9     INCR   Ri
      3.    IF =   Rx, Ry, THEN
            …
THEN: 7.    MUL    Rx, Ry, Rz
CONT: 10.   IF <Ri, RN, LOOP, delayed_branch
      8.    STORE  RZ, Ri, Rz
```

Lo spostamento della 9 ha annullato la dipendenza della 9 sulla 10 e ha aumentato la distanza della dipendenza della 2 sulla 3. Un risultato equivalente si sarebbe ottenuto usando la 9 dopo la 3 per permettere delayedbranch della IF =.

Il codice superscalare a 2 vie, secondo la tecnica VLIW, è il seguente (nel ramo ELSE devono essere replicate la IF < e la STORE):

```
LOOP:   1-2   LOAD   RX, Ri, Rx  |  LOAD   RY, Ri, Ry
        9-3   INCR   Ri  |  IF =   Rx, Ry, THEN
              …
THEN:   7-10  MUL    Rx, Ry, Rz  |  IF <Ri, RN, LOOP, delayed_branch
        8-x   STORE  RZ, Ri, Rz  |  NOP
```

La dipendenza logicadella 2 sulla 3 ($k_{2\text{-}3}$ =1, $N_{Q2\text{-}3}$ = 2) ha effetto, mentre non ha effetto quella della 9 sulla 10 (k = 1, N_Q= 1)in quanto concatenata e a bassa latenza, come si vede anche nella simulazione grafica. Invece, l'effetto della dipendenza della 7 sulla 8 ($k_{7\text{-}8}$ = 1, $N_{Q7\text{-}8}$ = 1, Lpipe = 4) rappresenta la principale causa di degradazione.

Per una *IU in-order*,con la simulazione grafica si ottieneil tempo di servizio per istruzione:

$$T = \frac{12}{7}\, t$$

Si ha pochissimo vantaggio rispetto all'architettura scalare (*T = 13t/7*) a causa delle piccole dimensioni del programma.

Usando il modello dei costi (*corso A*), si verifica il risultato precedente:

$$T = \frac{t}{2}(1 + \theta) + \lambda\, t + \Delta$$

$$\theta = 1/7$$

$$\lambda = 1/7$$

$$\Delta = t\, d_{2-3}\left(N_{Q2-3} + 1 - k_{2-3}\right) + t\, d_{7-8}\left(N_{Q7-8} + L_{pipe} + 1 - k_{7-8}\right) = \frac{7}{7}\, t$$

Il tempo di completamento ideale (con cache perfetta) vale:

$$T_{c-id} = 7\, N\, T = 12\, N\, t = 24\, N\, \tau$$

Considerando la gerarchia di memoria, il programma è caratterizzato da solo località dei dati, per cui (essendo Z in sola scrittura):

$$T_{fault} = N_{fault} * T_{copy} = 2\, \frac{N}{\sigma} * 2\, \sigma\, \tau = 4\, N\, \tau$$

$$T_c = T_{c-id} + T_{fault} = 28\, N\, \tau$$

b) L'istruzione aggiunta alla macchina D-RISC è del tipo

IF&CALC Ra-base, Ra-indice, Rb-base, Rb-indice, Rc

che occupa due parole. In una macchina superscalare a 2 vie, che funzioni fondamentalmente come una macchina D-RISC, IF&CALC è codificata da *una* istruzione lunga. Il comportamento temporale del suo interprete è del tutto simile a quello di una *istruzione lunga con operandi in memoria*: uno slot in IM, uno in IU che genera due richieste di lettura e informa la EU, uno in DM per leggere i due operandi, uno in EU_Master per effettuare il test e comandare l'esecuzione della moltiplicazione o della divisione in pipeline (altri 4 slot di latenza).

Il codice VLIW diviene:

```
        IF&CALC  RX, Ri, RY, Ri, Rz
LOOP:   INCR  Ri  |  STORE  RZ, Ri, Rz
        IF <Ri, RN, LOOP, delayed_branch  |  NOP
        IF&CALC  RX, Ri, RY, Ri, Rz
```

Ha effetto la dipendenza logica indotta dalla IF&CALC sulla STORE (k = 1, N_Q = 2, Lpipe = 4), mentre la dipendenza della INCR sulla IF < è concatenata e non ha effetto in quanto il suo ritardo è dominato dalla precedente.

Con la simulazione grafica si ottiene:

$$T = \frac{9}{4}\, t$$

che risulta maggiore rispetto al tempo di servizio della macchina D-RISC. Infatti l'istruzione complessa IF&CALC impedisce le stesse ottimizzazioni del codice possibili con una macchina D-RISC. D'altra parte, il notevole abbassamento del numero di istruzioni (da 7 a 4) per un loop così piccolo porta ad ottenere un *tempo di completamento* sensibilmente migliore:

$$T_{c-id} = 4\, N\, T = 9\, N\, t = 18\, N\, \tau$$

$$T_c = T_{c-id} + T_{fault} = 22\, N\, \tau$$

Domanda 3 (OLD-1)

Traccia di soluzione, da completare con il codice LC e con parti di pseudo-codice del supporto alle comunicazioni

a)Il processo S usa un canale asimmetrico, con grado di asincronia uno, per ricevere i messaggi dai clienti. Interagisce con il processo esterno I/O mediante una *send (..., N)* su canale asincrono di una posizione, ed un ciclo di $\lceil N/4K \rceil$ *receive (..., dest[i])* su canale asincrono. *Inserire il codice LC*.

Il supporto della send e della receive con il processo esterno è diverso da quello delle comunicazioni tra processi interni, in quanto lo scheduling a basso livello necessita di comunicazioni esplicite da CPU ad I/O e viceversa. *Spiegare questa parte dettagliando la descrizione del supporto.*

Quindi, il compilatore deve inserire delle librerie diverse rispetto alle normali primitive di comunicazione.

A parte eventuali differenze in prestazioni, non c'è differenza nel codice delle librerie usando Memory Mapped I/O oppure DMA, in quanto si tratta sempre di memoria condivisa riferita con indirizzi logici. Il file di configurazione che descrive le librerie è invece diverso per permettere la corretta corrispondenza tra indirizzi logici e fisici nel caso di strutture dati da allocare nella memoria locale di I/O.

b) Le comunicazioni di S avvengono solo con il Driver intermediario, quindi usando solo primitive valide per comunicazioni tra processi interni. Il codice sorgente può rimanere invariato se i nomi dei canali che nella versione precedente usava il processo I/O sono ora usati dal Driver, ma il codice oggetto è diverso dovendo, come detto sopra, utilizzare librerie di comunicazione diverse.