Esercizi vari

Salvatore Orlando

Domande

- Per quali istruzioni MIPS viene impiegata la modalità di indirizzamento PC-relative, e quali sono le motivazioni che hanno portato i progettisti ad adottare tale modalità?

- Perché potrebbe essere scorretto comparare le prestazioni di due CPU usando la Frequenza del clock o la misura MIPS? Quali alternative?

- Si può verificare un page fault dopo un TLB hit? Motivare.

- Considerare il datapath della CPU multiciclo vista a lezione. Come viene usata l'ALU durante i 3 cicli impiegati per l'esecuzione delle istruzioni di branch?

- Aumentare la dimensione del blocco della cache o aumentare l'associatività della cache possono diminuire il miss rate. Motivare brevemente.

Domande

- Discutere, rispetto ad un accesso alla memoria tramite lw, i ruoli di TLB, page table e cache. Si assuma che l'indirizzo fornito dalla lw sia virtuale, e che la cache sia acceduta con indirizzo fisico. Fare un esempio accesso con hit a tutti i livelli, ed un esempio con qualche tipo di miss.

- Polling e Interrupt sono due tecniche per programmare l'I/O. Discuterle brevemente, e spiegare quando entrambe non sono sufficienti e richiedono l'uso di DMA.

- Discutere a grandi linee i collegamenti necessari nel datapath pipeline visto a lezione per realizzare il *forwarding*. In particolare, se un'istruzione add segue una lw, e tra le due esiste una dipendenza RAW, discutere come questo collegamento riduca (ad uno solo) gli stalli della pipeline.

- Illustrare la necessità di avere programmi di benchmark accuratamente selezionati per valutare e comparare le prestazioni dei sistemi. Esemplificare rispetto ai benchmark SPEC.

Domande

- Si potrebbe sempre adottare sempre la politica callee-save per salvare i registri. In cosa consiste questa politica. Quali sono i pro e i contro rispetto alla quantità di registri da salvare e ripristinare?

- Discutere le convenzioni di chiamata MIPS per il passaggio di parametri, e per il ritorno delle funzioni.

- Perché è necessario allocare uno stack frame per ogni chiamata (istanza) di una certa procedura / funzione? Per cosa viene usato lo stack frame.

- I registri `$a0, … $s0, … $v0, … $t0, … $k0 …` sono dei registri generali usati, convenzionalmente, per scopi diversi. Quali?

- Qual è il compito dell'assemblatore e del linker, ovvero che differenza c'è tra i file oggetto e i file eseguibili ?

CPI e influenza cache

- Siano dati un processore, una cache e un mix di programmi
  - le istruzioni di lw/sw eseguite sono il 20% di IC
  - il CPI delle lw/sw (con hit in cache) è: $CPI_{lw/sw} = 5$
  - il CPI_{avg}, calcolato per la specifica cache, è 4
  - l'istruction miss rate è del 2%
  - il data miss rate è del 4%
  - il miss penalty è di 30 cicli
- Rispondere alle seguenti domande:
  - calcolare CPI_{miss}, ovvero i cicli mediamente spesi (per istruzione) a causa dei miss.
  - Calcolare CPI_{altro}, ovvero il CPI delle istruzioni diverse da lw/sw, che sono appunto l'80% del totale

- Cicli persi in media per istruzione a causa dei miss:
  - $Cicli_{miss} = Cicli_{istr} + Cicli_{dati}$ = (2% IC 30 + 4% (20% IC) 30) / IC = $0.02 * 30 + 0.04 * 0.2 * 30 = 0.84$ cicli
- Quindi
  - $CPI_{avg} = 80\% \, CPI_{altro} + 20\% \, CPI_{lw/sw} + Cicli_{miss}$
- Sostituendo:
  - $4 = 0.8 * CPI_{altro} + 0.2 * 5 + 0.84$ da cui: $CPI_{altro} = 2.7$

CPI e influenza cache (cont.)

- **Qual è lo speedup massimo ottenibile modificando l'architettura della cache ? (supponete, per effettuare il calcolo, di riuscire ad avere una cache ottimale, che evita completamente i miss)**

- **La formula per il CPI medio ottimo è quella precedente, senza però considerare i cicli dovuti ai miss:**
  - $CPI_{ottimo} = 80\%CPI_{altro} + 20\% CPI_{lw/sw} = 0.8 * 2.7 + 0.2 * 5 = 3.16$

- **Per calcolare lo speedup, visto che IC e frequenza del processore rimangono invariate, abbiamo che:**
  - $speedup_{max} = CPI_{avg} / CPI_{ottimo} = 4/3.16 = 1.27$

Cache

- **Considerare una cache diretta la cui dimensione dati è di 32 KB, con blocchi di 16 B.**
  - **Determinare TAG, INDEX e OFFSET per un indirizzo fisico di 32 b.**
- **OFFSET = log 16 = log 2^4 = 4 b**
- **INDEX = log 32K/16 = log 2^{11} = 11 b**
- **TAG = 32 - 11 - 4 = 17 b**

- **Supporre di effettuare ripetuti accessi alla cache, con indirizzi (alla word) consecutivi. Ad esempio, considerare di scorrere un array di interi.**
- **Considerando che ogni blocco è identificato da un certo valore di INDEX, in che sequenza vengono acceduti i vari blocchi della cache?**
- **I blocchi della cache sono acceduti sequenzialmente. Supponiamo che il primo intero dell'array cada nel blocco il cui INDEX corrisponde a *x*.**
- **Gli accessi successivi cadranno nei blocchi *(x*+1) mod 2K , *(x*+2) mod 2K e così via.**

Tracce di accesso alla cache dati

- **Si supponga che un programma sia stato tracciato nei sui accessi alla memoria fisica effettuati da istruzioni di `lw` e `sw`. In particolare, gli accessi tracciati sono i seguenti (ind. a 32 b):**

```
0x17 11 00 00
0x17 11 00 04
0x17 20 00 00
0x17 20 00 04
0x17 20 10 04
0x17 20 10 08
0x17 20 10 0a
...
```

- **La cache dati di primo livello del sistema è di 64 KB, con blocchi da 16 B.**
- **Rispondere alle seguenti domande considerando le due possibili organizzazioni della cache: diretta oppure associativa a 16 vie**

1. **Per ogni indirizzo tracciato, quali sono TAG, INDEX e OFFSET?**
2. **Se la cache è inizialmente vuota, quali sono i riferimenti di tipo *miss* o *hit*, e tra i miss quali provocano dei conflitti con il rimpiazzo del blocco?**

Tracce di accesso alla cache dati (cont.)

- Il numero di blocchi della cache, ovvero il numero di insiemi nella cache diretta, è pari a 64K/16 = 4K = 2^{12}.
- Per la cache associativa a 16 vie, il numero di insiemi è invece 4K/16 = 2^8.

- Per la cache diretta, l'INDEX è di log 2^{12} = 12 bit.
- Per la cache associativa, l'INDEX è di log 2^8 = 8 bit.

- Il Block Offset è invece di log 16 = 4 bit per entrambe le organizzazioni della cache

Tracce di accesso alla cache dati (cont.)

- Cache diretta, TAG, INDEX e OFFSET:

```
0x17 11 00 00    TAG=1711 INDEX =000 OFFSET=0
0x17 11 00 04    TAG=1711 INDEX =000 OFFSET=4
0x17 20 00 00    TAG=1720 INDEX =000 OFFSET=0
0x17 20 00 04    TAG=1720 INDEX =000 OFFSET=4
0x17 20 10 04    TAG=1720 INDEX =100 OFFSET=4
0x17 20 10 08    TAG=1720 INDEX =100 OFFSET=8
0x17 20 10 0a    TAG=1720 INDEX =100 OFFSET=a
...
```

- Miss e hit:

```
0x17 11 00 00   miss
0x17 11 00 04   hit (stesso blocco del rif. precedente)
0x17 20 00 00   miss con conflitto (stesso index, tag differente)
0x17 20 00 04   hit (stesso blocco del rif. precedente)
0x17 20 10 04   miss
0x17 20 10 08   hit (stesso blocco del rif. precedente)
0x17 20 10 0a   hit (stesso blocco del rif. precedente)
...
```

Tracce di accesso alla cache dati (cont.)

- Cache associativa a 16 vie, TAG, INDEX e OFFSET:

```
0x17 11 00 00 TAG=17110 INDEX =00 OFFSET=0
0x17 11 00 04 TAG=17110 INDEX =00 OFFSET=4
0x17 20 00 00 TAG=17200 INDEX =00 OFFSET=0
0x17 20 00 04 TAG=17200 INDEX =00 OFFSET=4
0x17 20 10 04 TAG=17201 INDEX =00 OFFSET=4
0x17 20 10 08 TAG=17201 INDEX =00 OFFSET=8
0x17 20 10 0a TAG=17201 INDEX =00 OFFSET=a
...
```

- Miss e hit:

```
0x17 11 00 00    miss
0x17 11 00 04    hit (stesso blocco del rif. precedente)
0x17 20 00 00    miss con conflitto sull'insieme (stesso index, tag differente)
0x17 20 00 04    hit (stesso blocco del rif. precedente)
0x17 20 10 04    miss con conflitto sull'insieme (stesso index, tag differente)
0x17 20 10 08    hit (stesso blocco del rif. precedente)
0x17 20 10 0a    hit (stesso blocco del rif. precedente)
...
```

In questo caso, i miss con conflitto sull'insieme non provocano il rimpiazzo del blocco, in quanto ogni insieme contiene ben 16 blocchi.

Esempio di transazione su bus sincrono (cont.)

- Si vuole calcolare l'ampiezza di banda di un bus sincrono nell'accedere singole word da una memoria da 200 ns
  - ampiezza dati pari a 1 word (32 b)
  - Bus a 20 MHz, dove ogni trasmissione richiede un ciclo
- Protocollo per effettuare una transazione:
  - 1 ciclo per invio indirizzo da parte del dispositivo
  - 200 nsec per lettura word
  - 1 ciclo per invio dato al dispositivo

- Ciclo del bus: T = 1/20M sec. = K/20 nsec. = 50 nsec.
- Accesso alla memoria: 200/50 = 4 cicli
- Per ogni transazione: 1+4+1=6 cicli ovvero 6*50 ns = 300 ns = 10^{-9} 300 s
- Banda: 4 B / 300 ns = 4B / (10^{-9} 300) s = 4 MB / 0,3 s = 13,3 MB/s

Impatto dell'I/O sulle prestazioni della CPU

- Considerare una CPU a 500MHz, con CPI=2.
- Il disco invece trasferisce ad una banda di 10 MB/s.
- La CPU viene chiamata in causa per ogni trasferimento in DMA di blocchi di 1KB, spende per questo 500 istr., e il disco lavora in continuazione.
- Qual è la percentuale di tempo che viene sprecata dalla CPU per attività di I/O?

- La CPU spende quindi CPI * 500 = 1000 cicli per il trattamento di ogni operazione di I/O.
- Il disco, nell'ipotesi che lavori in continuazione, effettua
  10MB / 1KB trasf. al sec = 10K trasf. al sec.
- Il numero di cicli totali sprecati è quindi 10K * 1000 cicli = 10M cicli.
- La percentuale *(x)* di cicli di CPU sprecata può essere ricavata dalla proporzione
  500M : 100% = 10M : *x%*, da cui *x%* = 100 * 10M / 500M = 2%.

- Se il disco lavorasse per solo il 50% del tempo, la risposta quale sarebbe?

- Se la CPU lavorasse al 50%, il numero di trasferimenti al secondo sarebbe la metà,
  ovvero 5K trasf. al sec per un totale di 5K * 1000 cicli = 5M cicli,
  da cui *x%* = 100 *5M / 500M = 1%.

Pipeline

- **Dato il seguente frammento di codice, determinare le dipendenze RAW.**

```
1. add $7, $8, $9
2. sub $5, $7, $8
3. lw $4, 0($3)
4. add $6, $4, $5
```

- **Disegnare anche i diagrammi temporale per architetture pipeline a 5 stadi con le seguenti caratteristiche:**
  1. **senza forwarding, ma con register file speciale, che prima scrive e poi legge nello stesso ciclo.**
  2. **con forwarding e register file speciale.**

- **Per la seconda architettura pipeline, proporre un re-scheduling delle istruzioni che, mantenendo la stessa semantica del programma, elimini tutti gli stalli.**

Pipeline (cont.)

- **Dato il seguente frammento di codice, determinare le dipendenze RAW.**

```
1. add $7, $8, $9
2. sub $5, $7, $8
3. lw $4, 0($3)
4. add $6, $4, $5
```

- **Le dipendenze sono 1 ➔ 2 (reg. $7), 3 ➔ 4 (reg. $4) e 2 ➔ 4 (reg. $5).**

- **Diagramma prima architettura:**

```
add   IF  ID  EX  ME   WB
sub       IF  ID <ID> <ID> EX  ME  WB
lw            IF <IF> <IF> ID  EX  ME   WB
add                        IF  ID <ID> <ID> EX  ME  WB
```

Pipeline (cont.)

- **Dato il seguente frammento di codice, determinare le dipendenze RAW.**

```
1. add $7, $8, $9
2. sub $5, $7, $8
3. lw $4, 0($3)
4. add $6, $4, $5
```

- **Le dipendenze sono 1 ➔ 2 (reg. $7), 3 ➔ 4 (reg. $4) e 2 ➔ 4 (reg. $5).**

- **Diagramma seconda architettura:**

```
add        IF   ID   EX   ME   WB
sub             IF   ID   EX   ME   WB
lw                   IF   ID   EX   ME   WB
add                       IF   ID <ID> EX   ME   WB
```

- **Codice dopo il re-scheduling (invertiamo le istr. 2 e 3) che elimina tutti gli stalli:**

```
add $7, $8, $9
lw $4, 0($3)
sub $5, $7, $8
add $6, $4, $5
```

Assembly

- **Tradurre in Assembly la seguente procedura, usando le solite convenzioni di chiamata.**
- **Usare il registro `$s0` per l'indice `int i`**

```
void update_odd(int a[], int dim) {
  int i;
  for (i=1; i<dim; i=i+2)
        a[i] = a[i]+100;
}
```

Assembly

- Psuedo:

```
# a in $a0
# dim in $a1
update_odd:
  < gestione stack >
  i = 1;      # uso del registro $s0 (callee-save)
init_for:
  if (i >= dim) goto exit_for;
  a[i] = a[i]+100;
  i = i + 2;
  goto init_for;
exit_for:
  < gestione stack >
  jr $ra
```

Assembly

- Per accedere all'elemento `a[i]`, l'indirizzo lo calcolo come:
  - `a + i * sz_elem = a + i * 4;`
- Codice:

```
# a in $a0
# dim in $a1
update_odd:
   addu  $sp, $sp -4
   sw    $s0, 0($sp)     # salva $s0
   li    $s0, 1          # i = 1
init_for:
   bge   $s0, $a1, exit_for
   sll   $t0, $s0, 2     # $t0 = i*4
   addu  $t0, $t0, $a0   # $t0 = a+i*4
   lw    $t1, 0($t0)
   addi  $t1, $t1, 100
   sw    $t1, 0($t0)
   addi  $s0, $s0, 2     # i = i + 2;
   j     init_for
```

```
exit_for:
   lw    $s0, 0($sp)
   addu  $sp, $sp 4
   jr    $ra
```