Architetture degli Elaboratori – AA 08-09

Seconda prova di verifica intermedia

18 dicembre 2008 – Bozza di soluzione

Domanda 1

Mentre il vettore C è soggetto solo a località, i vettori A e B sono soggetti anche a riuso, rispettivamente nei rami *then* ed *else*. Date le dimensioni dei vettori (15K ciascuno) e la capacità della cache (16K), entrambi i vettori A e B non possono essere mantenuti permanentemente in cache una volta caricati. Ne consegue che il numero di fault è comunque dell'ordine di N^2; per minimizzarne il valore, possiamo solo agire sulla costante moltiplicativa di N^2.

Dalla probabilità dei rami *then* e *else* consegue che la probabilità di fault viene minimizzata mantenendo permanentemente in cache il vettore A una volta caricato (utilizzo dell'opzione *non_deallocare* nell'istruzione LOAD del generico *A[j]*).

Sotto queste ipotesi l'insieme di lavoro consta di:

- tutti i blocchi di B,
- un blocco di C,
- un blocco di A (anche se ne potranno essere presenti più di uno, ma ciò non è significativo nel passare da una iterazione all'altra del loop più esterno e quindi non ha impatto sensibili sulla probabilità di fault),
- i blocchi del codice, dell'ordine di circa 8 blocchi, soggetti a riuso e mantenuti permanentemente in cache.

Il numero medio di fault è dato da:

$$N_{fault} \sim N_{fault\text{-}A} + N_{fault\text{-}B} + N_{fault\text{-}C} = \tfrac{1}{4}(N^2/\sigma) + \tfrac{3}{4}(N/\sigma) + N/\sigma \sim N^2/(4\sigma)$$

(Per B e per il C abbiamo solo i fault fisiologici, per A abbiamo i fault fisiologici ad ogni iterazione del ciclo più esterno)

Se avessimo mantenuto permanentemente in cache il vettore A, avremmo ottenuto un numero medio di fault 3 volte maggiore.

Domanda 2

Si suppone che il codice dei rami *then* ed *else* sia già disponibile come procedure *p_then* e *p_else*. Dal momento che entrambe le procedure prendono come parametri formali uno scalare e un vettore (A nel primo caso e B nel secondo) (*C[i]*) dobbiamo assumere che il primo parametro sia passato per valore ed il secondo per riferimento. Il codice per il main del programma potrebbe dunque essere il seguente:

```
main:  LD R_baseN, R_0, R_N     // lunghezza dei vettori
loop:  LD R_baseC, R_i, R_tmp   // caricamento C[i]
       IF>=0 R_tmp,  else       // salto condizionale al ramo else (cond negata)
```

```
then:  ST R_inP1, R_0, R_tmp   // parametro per valore in memoria
       ST R_inP1, #1, R_n      // lunghezza del vettore è il secondo param per val
       MOV R_baseA, R_inP2     // parametro per riferimento in registro
       CALL R_p_then, R_ret    // chiamata di procedura ramo then
       LD R_out, R0, R_tmp     // risultato procedura
       ST R_baseC, R_i, R_tmp  // memorizzazione in C[i]
       GOTO cont               // quando termina continua dopo l'if (ciclo)
else:  ST R_inP1, R_0, R_tmp   // parametro per valore in memoria
       ST R_inP1, #1, R_n      // lunghezza del vettore è il secondo param per val
       MOV R_baseB, R_inP2     // parametro per riferimento in registro
       CALL R_p_else, R_ret    // chiamata di procedura ramo else
       LD R_out, R0, R_tmp     // risultato procedura
       ST R_baseC, R_i, R_tmp  // memorizzazione in C[i]
cont:  INC R_i                 // incremento variabile di iterazione
       IF< R_i, R_N, loop      // cicla se sotto il limite
```

Si è assunto in questo caso che i parametri in memoria hanno l'indirizzo in R_{inP1} (offset 0 e 1) e quelli per riferimento in R_{inP2} per entrambe le procedure, e che il valore di ritorno (ancora passato in memoria) sia memorizzato sempre all'indirizzo R_{out}
È facile osservare come il parametro passato per valore (in memoria) sia lo stesso nei due rami (*then* ed *else*) e altrettanto vale per la memorizzazione del risultato. Pertanto il codice potrebbe essere ottimizzato, direttamente dal compilatore, come segue:

```
main:  LD R_baseN, R_0, R_N    // lunghezza dei vettori
loop:  LD R_baseC, R_i, R_tmp  // caricamento C[i]
       ST R_inP1, R_0, R_tmp   // parametri per valore in memoria (sono gli stessi
       ST R_inP1, #1, R_n      //              nei due casi then ed else)
       IF>=0 R_tmp,  else      // salto condizionale al ramo else (cond negata)
then:  MOV R_baseA, R_inP2     // parametro per riferimento in registro
       CALL R_p_then, R_ret    // chiamata di procedura ramo then
       GOTO cont               // quando termina continua dopo l'if (ciclo)
else:  MOV R_baseB, R_inP2     // parametro per riferimento in registro
       CALL R_p_else, R_ret    // chiamata di procedura ramo else
cont:  LD R_out, R0, R_tmp     // risultato procedura
                               // (comunque sia stato calcolato)
       ST R_baseC, R_i, R_tmp  // memorizzazione in C[i]
       INC R_i                 // incremento variabile di iterazione
       IF< R_i, R_N, loop      // cicla se sotto il limite
```

Si noti come i flag *non_deallocare* non compaiano nel codice, dal momento che sia A che B sono acceduti solo all'interno delle procedure *p_then* e *p_else*. In particolare, la procedura *p_then* avrà le LOAD per A etichettate con *non_deallocare*.

Domanda 3

Immaginiamo che il programma (pseudo codice ad alto livello) prosegua con una istruzione

```
        print(C);
```

La compilazione di PROVA in LC sarà dunque:

```
PROVA ::    channel out toPrt, toGestMem,
            channel in fromPrt(1), fromGestMem(1);
            // utilizziamo canali asincroni ad una posizione per non bloccare
            // i server. Il client si blocca comunque sulla receive dell'esito
{
  // calcolo del risultato
  for(int i=0; i<N; i++)
        C[i] = (C[i]>0 ? p_then(C[i], N, A) : p_else(C[i], N, B));

  // assumiamo un buffer di canale di 1K
  for(int i=0; i<15; i++)  {

        send(toPrt, <"stampa",slice(C,i)>);   // slice(C,i) dà l'i-esima
                                              // porzione di C lunga 1K
        receive(fromPrt, esito);
        if(esito != "OK")
              gestione_errore();

  }
```

La memoria virtuale del processo conterrà:

- il codice del corpo del processo
- il codice delle procedure *p_then* e *p_else*
- gli array A, B e C
- le routine per il trattamento di interruzioni ed eccezioni
- le routine per le primitive di comunicazione
- il PCB del processo
- la tabella di rilocazione del processo
- i PCB di tutti i processi
- la tabella dei canali di comunicazione

Le strutture condivise in questo caso sono:

- la struttura dei PCB (condivisa con gli altri processi, per via della schedulazione, e con il processo esterno che implementa l'unità stampante)
- i canali di comunicazione (condivisi con il processo esterno che implementa l'unità di I/O e con il processo gestore della memoria)