

Esercizio Cache

Andrea Bartolini – a.bartolini@unibo.it

FP Example: Array Multiplication

- $C = C + A \times B$ *moltiplicazione di A e B e somma su C*
 - All 32×32 matrices, 64-bit double-precision elements *tutte le matrici sono 32 x 32 elementi e ciascun elemento è composto da 64 bit.*
 - DGEMM (Double precision GEneral Matrix Multiply)
- C code: *l'implementazione in C di questa operazione è con 3 cicli for innestati, dove il ciclo + esterno scorre le righe della matrice C, il 2° for scorre le colonne della matrice C, e il ciclo k scorre le righe della m.A e le colonne della m.B.*

```
void mm (double c[32][32], double a[32][32], double b[32][32]) {
    size_t i, j, k;
    for (i = 0; i < 32; i = i + 1)
        for (j = 0; j < 32; j = j + 1)
            for (k = 0; k < 32; k = k + 1)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

- Addresses of c, a, b in x10, x11, x12, and i, j, k in x5, x6, x7

Contatori contenuti nei registri

$\&c[0][0]$
indirizzo del 1° elemento della matrice c.

$\&a[0][0]$

$\&b[0][0]$

Suppongo che gli indirizzi base di A, B, C trovino all'interno dei registri x10, x11, x12.

$$A = \begin{bmatrix} \cdot & & \\ & \cdot & \\ & & \cdot \\ & & & \cdot \end{bmatrix}$$

$i \downarrow$

$$B = \begin{bmatrix} x & & \\ & \cdot & \\ & & \cdot \\ & & & x \end{bmatrix}$$

$j \downarrow$

$$C = \begin{bmatrix} & & & j \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ & & & & x \end{bmatrix}$$

$i \downarrow$

l'indice i come le righe di C , quindi anche quelle di A

l'indice j come le colonne della matrice C , e le colonne della B

l'indice k non è in C , in quanto è quello che permette di fare le operazioni riga \times colonna.

\downarrow

tutti gli elementi di una riga di A (k) vengono scelti

\times calcolare l'elemento ij esimo di C bisogna prendere tutti gli elementi della riga di A e moltiplicarli per tutti gli elementi della colonna di B . Infatti k lo trova come indice di colonna di A e come indice di riga di B .

FP Example: Array Multiplication

■ RISC-V code:

il codice assembly precarica una serie di immediati
come 32 (dimensione dei 3 cicli for ($i < 32, j < 32, k < 32$))

mm:
load immediate
→ li x28, 32
→ li x5, 0
L1: → li x6, 0
L2: → li x7, 0
slli x30, x5, 5
add x30, x30, x6
slli x30, x30, 3
add x30, x10, x30
fld f0, 0(x30)
L3: slli x29, x7, 5
add x29, x29, x6
slli x29, x29, 3
add x29, x12, x29
fld f1, 0(x29)

alla fine di queste op su x30, trovo l'offset in mem. a cui fare l'accesso x $c[i][j]$

MO. ASSIETO
8
Sono i 3 loop associati ai cicli for.

↑ ACCESSO A B

// x28 = 32 (row size/loop end)

// i = 0; initialize 1st for loop

// j = 0; initialize 2nd for loop

// k = 0; initialize 3rd for loop

// x30 = $i * 2^{**5}$ (size of row of c)

// x30 = $i * \text{size}(\text{row}) + j$

// x30 = byte offset of $[i][j]$

// x30 = byte address of $c[i][j]$

// f0 = c[i][j]

// x29 = $k * 2^{**5}$ (size of row of b)

// x29 = $k * \text{size}(\text{row}) + j$

// x29 = byte offset of $[k][j]$

// x29 = byte address of $b[k][j]$

// f1 = b[k][j]

inizializzo i, j, k a zero.

offset di riga i in c

← sommiamo all'offset di riga i in c, l'offset di colonna j.

← shift x30 di 3 bit verso sx

← 1° load dell'elemento $c[i][j]$

prendo il valore dell'indice di riga i-esimo e lo shift verso sx di 5 bit

FP Example: Array Multiplication

...

```
slli    x29,x5,5      // x29 = i * 2**5 (size of row of a)
add     x29,x29,x7     // x29 = i * size(row) + k
slli    x29,x29,3      // x29 = byte offset of [i][k]
add     x29,x11,x29    // x29 = byte address of a[i][k]
fld     f2,0(x29)      // f2 = a[i][k]
( fmul.d f1, f2, f1    // f1 = a[i][k] * b[k][j]
  fadd.d f0, f0, f1    // f0 = c[i][j] + a[i][k] * b[k][j]
  addi   x7,x7,1       // k = k + 1 Quando k=32 & go store
  bltu   x7,x28,L3     // if (k < 32) go to L3
  fsd    f0,0(x30)     // c[i][j] = f0
  addi   x6,x6,1       // j = j + 1 Incremento j finché non arriva a 32 range L2
  bltu   x6,x28,L2     // if (j < 32) go to L2
  addi   x5,x5,1       // i = i + 1
  bltu   x5,x28,L1     // if (i < 32) go to L1
  ↓
indirizzo assoluto di b
```

- La cpu dispone di una cache dati a 2 vie da 16KiB complessivi e linee da 64 byte, gestita con stato MESI e con politica di scrittura Write-Around in caso di miss.
- Si considerino A,B,C immagazzinate in memoria a partire rispettivamente dagli indirizzi: 0x0100 8000, 0x0100 A000, 0x0100 C000
- 1) Si disegni la mappa della memoria *→ gli indirizzi in memoria degli elementi delle matrici (solo i primi e gli ultimi).*
- 2) Si analizzi la dinamica della cache dati, e, tenendo ben presente che il sistema ha un solo caching agent, si risponda in modo preciso, schematico, conciso e tabellare ai seguenti quesiti:
 - Quali sono gli indici di set e linea, e i tag associati ad A,B,C per il primo e l'ultimo blocco contenenti le matrici?
 - Quante linee di cache occuperanno nel loro insieme? Possono i due vettori e la variabile stare per intero e simultaneamente in cache? *se tutte e 3 le matrici riescono ad esser contenute in cache*
- 3) Si consideri la dinamica della cache nel calcolo della prima iterazione $i=0, j=0, k=0$ nel calcolo del primo elemento di C, disegnando lo stato MESI, il contenuto della cache e il valore del bit LRU dopo ogni operazione elementare (Load e Store) e si indichi il numero di accessi, il numero di miss e il numero di cicli di writeback e gli eventuali cicli di write allocate.
- 4) Si indichi il numero di accessi, il numero di miss e il numero di cicli di writeback e gli eventuali cicli di write allocate, nonché lo stato MESI della cache al termine del calcolo del primo elemento di $C[0][0]$. (si riporti il contenuto del primo ed ultimo set della cache)

↓
facendo variare k da 0 a 32.

FP Example: Array Multiplication

Le istruzioni che perturbano lo stato delle cache sono istruzioni di load/store. Nel codice: lo store è l'assegnamento.

Le load ci sono quando si leggono i singoli elementi della matrice.

■ $C = C + A \times B$

- All 32×32 matrices, 64-bit double-precision elements
- DGEMM (Double precision GEneral Matrix Multiply)

■ C code:

```
void mm (double c[][],  
         double a[][], double b[][]) {  
    size_t i, j, k;  
    for (i = 0; i < 32; i = i + 1)  
        for (j = 0; j < 32; j = j + 1)  
            for (k = 0; k < 32; k = k + 1)  
                c[i][j] = c[i][j]  
                    + a[i][k] * b[k][j];  
}
```

qui abbiamo 32 store e $32 \cdot 2 + 1$ load
quindi a ogni ciclo abbiamo:
32x32 load in c
32x32 store in c
32x32x32x2 load tra a e b.

- Addresses of c, a, b in x10, x11, x12, and
i, j, k in x5, x6, x7

↓
memo: i registri x sono registri interi

da quanti byte è composto un singolo elemento delle tre matrici? 8 byte
e prendo $a[0][0]$, quali indirizzi in memoria occuperà (parte da $0x01008000$)?
ma sono \downarrow double
 $\rightarrow 64 \text{ bit}$

da $0x01008000$
fino a $0x01008007$

$$c[i][j] = c[i][j] + a[i][k] * b[k][j];$$

suppongo di avere un certo
valore di i e di j

Prendo i (che è l'indice di riga) e lo shifto a sx di 5 bit, moltiplicandolo per 32. Questo ci porta nella parte di memoria associata a quella riga, poi gli sommo j e ci porta a un certo valore.
Poi devo dirgli che ciascun elemento è 8 byte, quindi moltiplico per 8.

