PRODOTTO TRA MATRICI

Progetto di Architettura degli Elaboratori

creato da

Stefano Zizzi

Università degli Studi di Urbino
Informatica Applicata

MATRICOLA: 312793
GIUGNO 2022

Indice

1 Specifica

1.1 Scopo del progetto

Effettuare in codice Assembly un prodotto tra due matrici, di cui la prima A formata da 2×4 elementi, e la seconda B formata da 4×4 ; ottimizzare il programma con l'aiuto del software *Winmips64*, cercando di renderlo il più efficiente possibile sulla base di diversi parametri.

1.2 Specifica funzionale

In matematica una matrice è una tabella ordinata di elementi. Il prodotto tra matrici non è commutativo e si può eseguire solo in certe condizioni. Date due matrici A e B è possibile effettuarne il solo se il numero di colonne di A è uguale al numero di righe di B, il risultato del loro prodotto sarà una matrice possedente il numero di colonne di B e il numero di righe di A. In questo caso quindi:

$$\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \end{pmatrix} \times \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}$$

$$= \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \end{pmatrix}$$

1.3 Specifica parametrica

Come obiettivi di ottimizzazione si punta a diminuire in modo prioritario il CPUC del programma, ossia l'intervallo di tempo (misurato in cicli di clock) che intercorre tra l'inizio e la fine dell'esecuzione del programma e, con priorità secondaria, si cerca di diminuire anche la dimensione del codice.

2 Progettazione

2.1 Input e Output

I dati da fornire in input ai programmi sono gli stessi, in modo da poter confrontare in modo più preciso la loro efficienza. Nello specifico i dati di input e output previsti sono i seguenti:

$$\begin{pmatrix} 4.3 & 2.1 & 3.3 & 5.2 \\ 4.8 & 9.3 & 2.3 & 0.5 \end{pmatrix} \times \begin{pmatrix} 2.3 & 1.1 & 4.3 & 6.2 \\ 5.3 & 5.5 & 3.5 & 7.3 \\ 4.9 & 2.3 & 4.3 & 2.1 \\ 2.3 & 0.5 & 1.8 & 1.1 \end{pmatrix}$$

$$= \begin{pmatrix} 49.15 & 26.47 & 49.39 & 54.64 \\ 72.75 & 61.97 & 63.98 & 103.03 \end{pmatrix}$$

2.2 Dati

A differenza del linguaggio C, in Assembly le matrici sono rappresentate come array unidimensionali:

```
1   .data
2           ;MATRICE A
3   a:      .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4           ;MATRICE B
5   b:      .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
        .1,2.3,0.5,1.8,1.1
6           ;MATRICE C
7   c:      .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Essendo però le matrici formate da due dimensioni sono stati implementati due metodi per poter indicizzarle per scorrerle:

- Metodo Statico: aggiorna il puntatore all'elemento da indicizzare utilizzando l'esatta posizione nell'array lineare, moltiplicata per 8 (la grandezza in bit del tipo double).
- Metodo Dinamico: aggiorna il puntatore con un'equazione basata sugli indici dei loop.

Entrambi i metodi sono stati implementati, nello specifico il Metodo Statico è stato utilizzato fino alla Versione 6 del codice, mentre quello Dinamico solo nella Versione 7, infatti chiamata Versione Dinamica.

2.3 WinMIPS64

L'Architettura dell'Elaboratore simulata da WINMIPS che è stata utilizzata è quella di default:

Architettura	
Code Address Bus	10
Data Address Bus	10
FP Addition Latency	4
Multiplier Latency	7
Division Latency	24

Le statistiche utilizzate per confrontare le diverse versioni del codice in Assembly sono le seguenti:

Execution	
Cycles	0
Instructions	0
CPI	0
Stalls	
Raw Stalls	0
WAW Stalls	0
WAR Stalls	0
Structural Stalls	0
Branch Taken Stalls	0
Branch Misprediction Stalls	0
Code Size	
Bytes	0

3 Implementazione Algoritmo

3.1 Codice in C

In ANSI C la matrice viene rappresentata come un'array di due dimensioni, in questo caso di tipo double, la moltiplicazione tra i due array viene effettuata tramite di 3 cicli for annidati. L'algoritmo non supporta matrici di qualsiasi grandezza in quanto gli array sono stati dichiarati in modo statico. Di seguito viene riportato il codice:

```
#include <stdio.h>

int main()
{
    int i,
        j,
        k,
        n = 4, /*colonne A*/
        m = 4, /*colonne B*/
        o = 2; /*righe A*/
double  A[2][4] = {{4.3,2.1,3.3,5.2},
                   {4.8,9.3,2.3,0.5}},
        B[4][4] = {{2.3,1.1,4.3,6.2},
                   {5.3,5.5,3.5,7.3},
                   {4.9,2.3,4.3,2.1},
                   {2.3,0.5,1.8,1.1}},
        C[2][4];

    /* effettuare la moltiplicazione */
    for(i = 0; i < o; i++)
        for(j = 0; j < m; j++)
            for(k = 0; k < n; k++)
                C[i][j] += A[i][k] * B[k][j];

    /* stampare la matrice prodotto AB */
    printf("\nRisultato: \n\n");
    for(i = 0; i < o; i++)
    {
        for(j = 0; j < m; j++)
            printf("%4.2f ", C[i][j]);
        printf("\n");
    }
}
```

3.2 Codice in Assembly

In Assembly il funzionamento è simile a quello in ANSI C, con qualche minima differenza a causa del diverso metodo di indicizzazione.

3.2.1 Versione 1

```
1   .data
2         ;MATRICE A
3   a:    .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5   b:    .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
      .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7   c:    .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9   p:    .double 0
10        ;COLONNE A
11  n:    .word   4
12        ;COLONNE B
13  m:    .word   4
14        ;RIGHE A
15  o:    .word   2
16
17    .text
18  start:  LW  r1, n(r0)           ; puntatore colonne a
19        DADDI   r2, r0, a       ; puntatore a primo elemento
      a
20        DADDI   r3, r0, b       ; puntatore a primo elemento
      b
21        DADDI   r4, r0, c       ; puntatore a primo elemento
      c
22        LW  r5, m(r0)           ; puntatore colonne b
23        LW  r6, o(r0)           ; puntatore righe a
24        L.D f0, p(r0)           ; inizializzazione parziale
25  loop:   L.D f1, 0(r2)         ; leggi a[i]
26        L.D     f2, 0(r3)       ; leggi b[i]
27        MUL.D   f1, f1, f2      ; a[i]*b[i]
28        ADD.D   f0, f0, f1      ; p = p + a[i]*b[i] (
      risultato parziale)
29        DADDI   r2, r2, 8       ; scorri elemento a
30        DADDI   r3, r3, 32      ; scorri elemento b
31        DADDI   r1, r1, -1      ; decrementa contatore
      colonne a
32        BNEZ    r1, loop        ; ripeti se contatore colonne
       a diverso da 0
33        S.D     f0, 0(r4)       ; salva il risultato
```

```
34          MUL.D   f0, f0, f3       ; resetta il risultato
    parziale
35 pointers: DADDI   r4, r4, 8    ; scorri registro risultati
36   DADDI   r3, r3, -120  ; riporta puntatore b all'inizio + 1
37   DADDI   r2, r2, -32 ; riporta puntatore a all'inizio
38   DADDI   r1, r1, 4 ; resetta contatore colonne a
39   DADDI   r5, r5, -1  ; decrementa contatore colonne b
40   BNEZ  r5, loop  ; ripeti se contatore colonne b diverso da
    0
41 rows: DADDI r2, r2, 32         ; riporta puntatore a alla
    seconda riga
42   DADDI r3, r3, -32 ; riporta puntatore b all'inizio
43   DADDI   r5, r5, 4 ; resetta numero di colonne b
44   DADDI r6, r6, -1  ; decrementa contatore righe a
45   BNEZ  r6, loop    ; ripeti se contatore right a diverso da
    0
46 end:  HALT
```

La maggior parte dei *RAW Stalls*[1] avviene a causa di una dipendenza dei dati tra l'istruzione MUL e ADD, in quanto l'istruzione ADD necessita del dato prodotto dall'istruzione MUL, causando diversi stalli per loop. Altri stalli sono dovuti alla dipendenza tra l'istruzione MUL e la seconda istruzione L.D. I *WAR Stalls*[2] sono dovuti al MUL per resettare il risultato parziale. I *Branch Taken Stalls* sono causati dai salti condizionali, nello specifico dall'istruzione BNEZ. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	479
Instructions	338
CPI	1.417
Stalls	
Raw Stalls	314
WAW Stalls	0
WAR Stalls	24
Structural Stalls	40
Branch Taken Stalls	31
Branch Misprediction Stalls	0
Code Size	
Bytes	116

[1]Read After Write, avviene in caso un'istruzione richieda un dato non ancora calcolato.

[2]Write After Read, si verifica nel momento in cui un'istruzione legge un dato che si trova in una locazione in cui un'istruzione successiva sta per salvare un altro dato.

3.2.2 Versione 2

```
1   .data
2         ;MATRICE A
3  a:     .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5  b:     .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
       .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7  c:     .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9  p:     .double 0
10        ;COLONNE B
11 m:     .word   4
12        ;RIGHE A
13 o:     .word   2
14
15    .text
16 start:  DADDI   r2, r0, a       ; puntatore a primo elemento
       a
17       DADDI   r3, r0, b        ; puntatore a primo elemento b
18       DADDI   r4, r0, c        ; puntatore a primo elemento c
19       LW  r5, m(r0)            ; puntatore colonne b
20       LW  r6, o(r0)            ; puntatore righe a
21       L.D f0, p(r0)            ; inizializzazione parziale
22
23 loop:   L.D f1, 0(r2)            ; leggi a[i]
24       L.D     f2, 0(r3)        ; leggi b[i]
25       MUL.D   f1, f1, f2       ; a[i]*b[i]
26       ADD.D   f0, f0, f1       ; p = p + a[i]*b[i] (risultato
       parziale)
27
28    L.D     f1, 8(r2)        ; leggi a[i]
29       L.D     f2, 32(r3)       ; leggi b[i]
30       MUL.D   f1, f1, f2       ; a[i]*b[i]
31       ADD.D   f0, f0, f1       ; p = p + a[i]*b[i] (risultato
       parziale)
32
33    L.D     f1, 16(r2)       ; leggi a[i]
34       L.D     f2, 64(r3)       ; leggi b[i]
35       MUL.D   f1, f1, f2       ; a[i]*b[i]
36       ADD.D   f0, f0, f1       ; p = p + a[i]*b[i] (risultato
       parziale)
37
38    L.D     f1, 24(r2)       ; leggi a[i]
39       L.D     f2, 96(r3)       ; leggi b[i]
40       MUL.D   f1, f1, f2       ; a[i]*b[i]
41       ADD.D   f0, f0, f1       ; p = p + a[i]*b[i] (risultato
       parziale)
```

```
42
43        S.D      f0, 0(r4)        ; salva il risultato
44        MUL.D    f0, f0, f3       ; resetta il risultato parziale
45 pointers: DADDI   r4, r4, 8 ;scorri registro risultati
46      DADDI    r3, r3, 8      ; punta alla nuova colonna
47      DADDI    r5, r5, -1     ; decrementa contatore colonne b
48      BNEZ   r5, loop     ; ripeti se contatore colonne b diverso
        da 0
49 rows: DADDI r2, r2, 32         ; riporta puntatore a alla
    seconda riga
50      DADDI r3, r3, -32    ; riporta puntatore b all'inizio
51      DADDI    r5, r5, 4    ; resetta numero di colonne b
52      DADDI r6, r6, -1      ; decrementa contatore righe a
53      BNEZ   r6, loop       ; ripeti se contatore right a diverso
        da 0
54 end:   HALT
```

Nella seconda versione del codice è stato attuato un *Loop Unrolling* del loop più interno, al fine di eliminare alcune delle dipendenze e avere la possibilità di eseguire altre tecniche per ridurre il numero di cicli. Grazie al *Loop Unrolling* è stato possibile eliminare anche il contatore delle colonne della matrice A in quanto la sua funzione era quella di indice del ciclo. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	464
Instructions	193
CPI	2.404
Stalls	
Raw Stalls	330
WAW Stalls	0
WAR Stalls	168
Structural Stalls	10
Branch Taken Stalls	7
Branch Misprediction Stalls	0
Code Size	
Bytes	136

Tabella 1: Statistiche versione attuale

Execution	
Cycles	-15
Instructions	+145
CPI	+0.98
Stalls	
Raw Stalls	+16
WAW Stalls	0
WAR Stalls	+144
Structural Stalls	-30
Branch Taken Stalls	-24
Branch Misprediction Stalls	0
Code Size	
Bytes	+20

Tabella 2: Confronto versione precedente

3.2.3 Versione 3

```
1   .data
2         ;MATRICE A
3  a:     .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5  b:     .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
       .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7  c:     .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9  p:     .double 0
10        ;COLONNE B
11 m:     .word   4
12        ;RIGHE A
13 o:     .word   2
14
15    .text
16 start:  DADDI   r2, r0, a       ; puntatore a primo elemento
       a
17       DADDI   r3, r0, b       ; puntatore a primo elemento b
18       DADDI   r4, r0, c       ; puntatore a primo elemento c
19       LW  r5, m(r0)           ; puntatore colonne b
20       LW  r6, o(r0)           ; puntatore righe a
21       L.D f0, p(r0)           ; inizializzazione parziale
22
23 loop:   L.D f1, 0(r2)           ; leggi a[i]
24       L.D     f2, 0(r3)       ; leggi b[i]
25         L.D     f1, 8(r2)       ; leggi a[i]
26       L.D     f2, 32(r3)      ; leggi b[i]
27         L.D     f1, 16(r2)      ; leggi a[i]
28       L.D     f2, 64(r3)      ; leggi b[i]
29         L.D     f1, 24(r2)      ; leggi a[i]
30       L.D     f2, 96(r3)      ; leggi b[i]
31       MUL.D   f1, f1, f2      ; a[i]*b[i]
32       ADD.D   f0, f0, f1      ; p = p + a[i]*b[i] (risultato
       parziale)
33       MUL.D   f1, f1, f2      ; a[i]*b[i]
34       ADD.D   f0, f0, f1      ; p = p + a[i]*b[i] (risultato
       parziale)
35       MUL.D   f1, f1, f2      ; a[i]*b[i]
36       ADD.D   f0, f0, f1      ; p = p + a[i]*b[i] (risultato
       parziale)
37       MUL.D   f1, f1, f2      ; a[i]*b[i]
38       ADD.D   f0, f0, f1      ; p = p + a[i]*b[i] (risultato
       parziale)
39       S.D     f0, 0(r4)       ; salva il risultato
40       MUL.D   f0, f0, f3      ; resetta il risultato parziale
41 pointers: DADDI   r4, r4, 8 ; scorri registro risultati
```

```
42      DADDI   r3, r3, 8      ; punta alla nuova colonna
43      DADDI   r5, r5, -1     ; decrementa contatore colonne b
44      BNEZ  r5, loop     ; ripeti se contatore colonne b diverso
        da 0
45  rows: DADDI r2, r2, 32         ; riporta puntatore a alla
      seconda riga
46      DADDI r3, r3, -32    ; riporta puntatore b all'inizio
47      DADDI   r5, r5, 4    ; resetta numero di colonne b
48      DADDI r6, r6, -1     ; decrementa contatore righe a
49      BNEZ  r6, loop       ; ripeti se contatore right a diverso
        da 0
50  end:  HALT
```

Nella terza versione è stato attuato un *Istruction Reordering* al fine di eliminare alcune delle dipendenze, nello specifico le dipendenze tra il caricamento dei dati nei registri e le operazioni su di essi. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	440
Instructions	193
CPI	2.280
Stalls	
Raw Stalls	282
WAW Stalls	0
WAR Stalls	168
Structural Stalls	10
Branch Taken Stalls	7
Branch Misprediction Stalls	0
Code Size	
Bytes	136

Tabella 3: Statistiche versione attuale

Execution	
Cycles	-24
Instructions	0
CPI	-0.12
Stalls	
Raw Stalls	-48
WAW Stalls	0
WAR Stalls	0
Structural Stalls	0
Branch Taken Stalls	0
Branch Misprediction Stalls	0
Code Size	
Bytes	+20

Tabella 4: Confronto versione precedente

3.2.4 Versione 4

```
1   .data
2         ;MATRICE A
3  a:     .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5  b:     .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
       .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7  c:     .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9  p:     .double 0
10        ;COLONNE B
11 m:     .word   4
12        ;RIGHE A
13 o:     .word   2
14
15    .text
16 start:  DADDI   r2, r0, a       ; puntatore a primo elemento
       a
17       DADDI   r3, r0, b       ; puntatore a primo elemento b
18       DADDI   r4, r0, c       ; puntatore a primo elemento c
19       LW  r5, m(r0)           ; puntatore colonne b
20       LW  r6, o(r0)           ; puntatore righe a
21       L.D f0, p(r0)           ; inizializzazione parziale
22
23 loop:   L.D f1, 0(r2)           ; leggi a[i]
24       L.D     f2, 0(r3)       ; leggi b[i]
25         L.D     f3, 8(r2)       ; leggi a[i]
26       L.D     f4, 32(r3)      ; leggi b[i]
27         L.D     f5, 16(r2)      ; leggi a[i]
28       L.D     f6, 64(r3)      ; leggi b[i]
29         L.D     f7, 24(r2)      ; leggi a[i]
30       L.D     f8, 96(r3)      ; leggi b[i]
31       MUL.D   f1, f1, f2      ; a[i]*b[i]
32         MUL.D   f3, f3, f4      ; a[i]*b[i]
33         MUL.D   f5, f5, f6      ; a[i]*b[i]
34         MUL.D   f7, f7, f8      ; a[i]*b[i]
35       ADD.D   f9, f3, f1      ; p = p + a[i]*b[i] (risultato
       parziale)
36       ADD.D   f10, f5, f7     ; p = p + a[i]*b[i] (risultato
       parziale)
37       ADD.D   f0, f10, f9     ; p = p + a[i]*b[i] (risultato
       parziale)
38       S.D     f0, 0(r4)       ; salva il risultato
39       MUL.D   f0, f0, f31     ; resetta il risultato parziale
40 pointers: DADDI   r4, r4, 8 ; scorri registro risultati
41     DADDI   r3, r3, 8     ; punta alla nuova colonna
42     DADDI   r5, r5, -1    ; decrementa contatore colonne b
```

```
43      BNEZ   r5, loop     ; ripeti se contatore colonne b diverso
        da 0
44  rows: DADDI r2, r2, 32         ; riporta puntatore a alla
        seconda riga
45      DADDI r3, r3, -32    ; riporta puntatore b all'inizio
46      DADDI   r5, r5, 4    ; resetta numero di colonne b
47      DADDI r6, r6, -1     ; decrementa contatore righe a
48      BNEZ   r6, loop      ; ripeti se contatore right a diverso
        da 0
49  end:   HALT
```

Nella quarta versione ho attuato un *Register Renaming* al fine di eliminare alcune delle dipendenze, in quanto alcune istruzioni richiedevano gli stessi registri per compiere le operazioni. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	296
Instructions	185
CPI	1.600
Stalls	
Raw Stalls	114
WAW Stalls	0
WAR Stalls	24
Structural Stalls	10
Branch Taken Stalls	7
Branch Misprediction Stalls	0
Code Size	
Bytes	132

Tabella 5: Statistiche versione attuale

Execution	
Cycles	-144
Instructions	-8
CPI	-0.68
Stalls	
Raw Stalls	-168
WAW Stalls	0
WAR Stalls	-144
Structural Stalls	0
Branch Taken Stalls	0
Branch Misprediction Stalls	0
Code Size	
Bytes	-4

Tabella 6: Confronto versione precedente

3.2.5 Versione 5

```
1   .data
2         ;MATRICE A
3  a:     .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5  b:     .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
       .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7  c:     .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9  p:     .double 0
10        ;RIGHE A
11 o:     .word   2
12
13    .text
14 start:  DADDI   r2, r0, a       ; puntatore a primo elemento
       a
15       DADDI   r3, r0, b       ; puntatore a primo elemento b
16       DADDI   r4, r0, c       ; puntatore a primo elemento c
17       LW  r6, o(r0)           ; puntatore righe a
18       L.D f0, p(r0)           ; inizializzazione parziale
19
20 loop:   L.D     f1, 0(r2)       ; leggi a[i]
21       L.D     f2, 0(r3)       ; leggi b[i]
22         L.D     f3, 8(r2)       ; leggi a[i]
23       L.D     f4, 32(r3)      ; leggi b[i]
24         L.D     f5, 16(r2)      ; leggi a[i]
25       L.D     f6, 64(r3)      ; leggi b[i]
26         L.D     f7, 24(r2)      ; leggi a[i]
27       L.D     f8, 96(r3)      ; leggi b[i]
28       MUL.D   f9, f1, f2      ; a[i]*b[i]
29         MUL.D   f10, f3, f4     ; a[i]*b[i]
30         MUL.D   f11, f5, f6     ; a[i]*b[i]
31         MUL.D   f12, f7, f8     ; a[i]*b[i]
32       ADD.D   f9, f9, f10     ; p = p + a[i]*b[i] (risultato
       parziale)
33       ADD.D   f11, f11, f12   ; p = p + a[i]*b[i] (risultato
       parziale)
34       ADD.D   f0, f9, f11     ; p = p + a[i]*b[i] (risultato
       parziale)
35       S.D     f0, 0(r4)       ; salva il risultato
36       MUL.D   f0, f0, f31     ; resetta il risultato parziale
37
38       L.D     f2, 8(r3)       ; leggi b[i]
39       L.D     f4, 40(r3)      ; leggi b[i]
40       L.D     f6, 72(r3)      ; leggi b[i]
41       L.D     f8, 104(r3)     ; leggi b[i]
42       MUL.D   f9, f1, f2      ; a[i]*b[i]
```

```
43          MUL.D   f10, f3, f4      ; a[i]*b[i]
44          MUL.D   f11, f5, f6      ; a[i]*b[i]
45          MUL.D   f12, f7, f8      ; a[i]*b[i]
46        ADD.D   f9, f9, f10     ; p = p + a[i]*b[i] (risultato
    parziale)
47        ADD.D   f11, f11, f12   ; p = p + a[i]*b[i] (risultato
    parziale)
48        ADD.D   f0, f9, f11     ; p = p + a[i]*b[i] (risultato
    parziale)
49        S.D     f0, 8(r4)       ; salva il risultato
50        MUL.D   f0, f0, f31     ; resetta il risultato parziale
51
52      L.D     f2, 16(r3)      ; leggi b[i]
53        L.D     f4, 48(r3)      ; leggi b[i]
54        L.D     f6, 80(r3)      ; leggi b[i]
55        L.D     f8, 112(r3)     ; leggi b[i]
56        MUL.D   f9, f1, f2      ; a[i]*b[i]
57          MUL.D   f10, f3, f4     ; a[i]*b[i]
58          MUL.D   f11, f5, f6     ; a[i]*b[i]
59          MUL.D   f12, f7, f8     ; a[i]*b[i]
60        ADD.D   f9, f9, f10     ; p = p + a[i]*b[i] (risultato
    parziale)
61        ADD.D   f11, f11, f12   ; p = p + a[i]*b[i] (risultato
    parziale)
62        ADD.D   f0, f9, f11     ; p = p + a[i]*b[i] (risultato
    parziale)
63        S.D     f0, 16(r4)      ; salva il risultato
64        MUL.D   f0, f0, f31     ; resetta il risultato parziale
65
66      L.D     f2, 24(r3)      ; leggi b[i]
67        L.D     f4, 56(r3)      ; leggi b[i]
68        L.D     f6, 88(r3)      ; leggi b[i]
69        L.D     f8, 120(r3)     ; leggi b[i]
70        MUL.D   f9, f1, f2      ; a[i]*b[i]
71          MUL.D   f10, f3, f4     ; a[i]*b[i]
72          MUL.D   f11, f5, f6     ; a[i]*b[i]
73          MUL.D   f12, f7, f8     ; a[i]*b[i]
74        ADD.D   f9, f9, f10     ; p = p + a[i]*b[i] (risultato
    parziale)
75        ADD.D   f11, f11, f12   ; p = p + a[i]*b[i] (risultato
    parziale)
76        ADD.D   f0, f9, f11     ; p = p + a[i]*b[i] (risultato
    parziale)
77        S.D     f0, 24(r4)      ; salva il risultato
78        MUL.D   f0, f0, f31     ; resetta il risultato parziale
79
80      DADDI   r4, r4, 32  ; scorri registro risultati
81  rows: DADDI r2, r2, 32        ; riporta puntatore a alla
    seconda riga
```

```
82      DADDI r6, r6, -1      ; decrementa contatore righe a
83      BNEZ  r6, loop        ; ripeti se contatore right a diverso
        da 0
84  end:  HALT
```

Nella quinta versione ho effettuato un nuovo *Loop Unrolling* eliminando così il secondo loop e con esso alcune delle dipendenze, e rimuovendo il contatore delle colonne di B che faceva da indice del loop e aprendo la possibilità a possibili ottimizzazioni. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	223
Instructions	127
CPI	1.756
Stalls	
Raw Stalls	106
WAW Stalls	0
WAR Stalls	24
Structural Stalls	9
Branch Taken Stalls	1
Branch Misprediction Stalls	0
Code Size	
Bytes	268

Tabella 7: Statistiche versione attuale

Execution	
Cycles	-73
Instructions	-58
CPI	+0,15
Stalls	
Raw Stalls	-8
WAW Stalls	0
WAR Stalls	0
Structural Stalls	-1
Branch Taken Stalls	-6
Branch Misprediction Stalls	0
Code Size	
Bytes	+136

Tabella 8: Confronto versione precedente

3.2.6 Versione 6

```
1   .data
2         ;MATRICE A
3 a:      .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5 b:      .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
    .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7 c:      .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9 p:      .double 0
10        ;RIGHE A
11 o:     .word   2
12
13    .text
14 start:  DADDI   r2, r0, a       ; puntatore a primo elemento
    a
15       DADDI   r3, r0, b       ; puntatore a primo elemento b
16       DADDI   r4, r0, c       ; puntatore a primo elemento c
17       LW  r6, o(r0)           ; puntatore righe a
18       L.D f0, p(r0)           ; inizializzazione parziale
19
20 loop:   L.D     f1, 0(r2)       ; leggi a[i]
21       L.D     f2, 0(r3)       ; leggi b[i]
22         L.D     f3, 8(r2)       ; leggi a[i]
23       L.D     f4, 32(r3)      ; leggi b[i]
24         L.D     f5, 16(r2)      ; leggi a[i]
25       L.D     f6, 64(r3)      ; leggi b[i]
26         L.D     f7, 24(r2)      ; leggi a[i]
27       L.D     f8, 96(r3)      ; leggi b[i]
28
29         L.D     f9, 8(r3)       ; leggi b[i]
30       L.D     f10, 40(r3)     ; leggi b[i]
31       L.D     f11, 72(r3)     ; leggi b[i]
32       L.D     f12, 104(r3)    ; leggi b[i]
33
34         L.D     f13, 16(r3)     ; leggi b[i]
35       L.D     f14, 48(r3)     ; leggi b[i]
36       L.D     f15, 80(r3)     ; leggi b[i]
37       L.D     f16, 112(r3)    ; leggi b[i]
38
39         L.D     f17, 24(r3)     ; leggi b[i]
40       L.D     f18, 56(r3)     ; leggi b[i]
41       L.D     f19, 88(r3)     ; leggi b[i]
42       L.D     f20, 120(r3)    ; leggi b[i]
43
44       MUL.D   f21, f1, f2     ; a[i]*b[i]
45         MUL.D   f22, f3, f4     ; a[i]*b[i]
```

```
46          MUL.D   f23, f5, f6      ; a[i]*b[i]
47          MUL.D   f24, f7, f8      ; a[i]*b[i]
48          MUL.D   f25, f1, f9      ; a[i]*b[i]
49          MUL.D   f26, f3, f10     ; a[i]*b[i]
50          MUL.D   f27, f5, f11     ; a[i]*b[i]
51          MUL.D   f28, f7, f12     ; a[i]*b[i]
52          MUL.D   f2, f1, f13      ; a[i]*b[i]
53          MUL.D   f4, f3, f14      ; a[i]*b[i]
54          MUL.D   f6, f5, f15      ; a[i]*b[i]
55          MUL.D   f8, f7, f16      ; a[i]*b[i]
56          MUL.D   f9, f1, f17      ; a[i]*b[i]
57          MUL.D   f10, f3, f18     ; a[i]*b[i]
58          MUL.D   f11, f5, f19     ; a[i]*b[i]
59          MUL.D   f12, f7, f20     ; a[i]*b[i]
60
61          ADD.D   f25, f25, f26    ; p = p + a[i]*b[i] (
    risultato parziale)
62        ADD.D   f27, f27, f28    ; p = p + a[i]*b[i] (risultato
    parziale)
63        ADD.D   f21, f21, f22    ; p = p + a[i]*b[i] (risultato
    parziale)
64        ADD.D   f23, f23, f24    ; p = p + a[i]*b[i] (risultato
    parziale)
65          ADD.D   f9, f9, f10      ; p = p + a[i]*b[i] (
    risultato parziale)
66          ADD.D   f2, f2, f4       ; p = p + a[i]*b[i] (
    risultato parziale)
67          ADD.D   f6, f6, f8       ; p = p + a[i]*b[i] (
    risultato parziale)
68          ADD.D   f11, f11, f12    ; p = p + a[i]*b[i] (
    risultato parziale)
69
70        ADD.D   f21, f21, f23    ; risulato finale 1
71        ADD.D   f25, f25, f27    ; risultato finale 2
72        ADD.D   f2, f2, f6       ; risultato finale 3
73        ADD.D   f9, f9, f11      ; risultato finale 4
74
75     DADDI r6, r6, -1     ; decrementa contatore righe a
76          DADDI   r4, r4, 32       ; scorri registro risultati
77        DADDI   r2, r2, 32       ; riporta puntatore a alla
    seconda riga
78
79          S.D     f21, 0(r4)       ; salva il risultato
80        S.D     f25, 8(r4)       ; salva il risultato
81        S.D     f2, 16(r4)       ; salva il risultato
82        S.D     f9, 24(r4)       ; salva il risultato
83     BNEZ    r6, loop     ; ripeti se contatore right a diverso
     da 0
84 end:  HALT
```

Nella sesta versione ho effettuato *Instruction Reordering* e *Register Renaming* al fine di eliminare gli ultimi stalli, rimanendo con un *Branch Taken Stall*[3] dovuto all'istruzione di ripetizione del loop, mentre i 12 *Stalli Strutturali*[4] sono dovuti alla falsa dipendenza tra gli ultimi MUL e gli ADD. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	135
Instructions	118
CPI	1.144
Stalls	
Raw Stalls	0
WAW Stalls	0
WAR Stalls	0
Structural Stalls	12
Branch Taken Stalls	1
Branch Misprediction Stalls	0
Code Size	
Bytes	248

Tabella 9: Statistiche versione attuale

Execution	
Cycles	-88
Instructions	-9
CPI	-0.61
Stalls	
Raw Stalls	-106
WAW Stalls	0
WAR Stalls	-24
Structural Stalls	+3
Branch Taken Stalls	0
Branch Misprediction Stalls	0
Code Size	
Bytes	-20

Tabella 10: Confronto versione precedente

[3]Branch Taken Stall avviene quando si esegue un branch (vai a un'istruzione diversa dalla successiva), l'istruzione sull'indirizzo di destinazione non è ancora stata decodificata.

[4]Stallo Strutturale è il tentativo di usare la stessa risorsa hardware da parte di diverse istruzioni in modi diversi nello stesso ciclo di clock.

3.2.7 Versione Dinamica

```
1         .data
2         ;MATRICE A
3 a:      .double 4.3,2.1,3.3,5.2,4.8,9.3,2.3,0.5
4         ;MATRICE B
5 b:      .double 2.3,1.1,4.3,6.2,5.3,5.5,3.5,7.3,4.9,2.3,4.3,2
      .1,2.3,0.5,1.8,1.1
6         ;MATRICE C
7 c:      .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8         ;PARZIALE
9 p:      .double 0
10        ;COLONNE A
11 ca:     .word   4
12        ;COLONNE B
13 cb:     .word   4
14        ;RIGHE A
15 ra:     .word   2
16        ;RIGHE B
17 rb:     .word   4
18        ;GRANDEZZA IN BIT
19 inc:    .word   8
20        ;INDICE 1
21 i:      .word 0
22        ;INDICE 2
23 j:      .word 0
24        ;INDICE 3
25 k:      .word 0
26        ;SETS
27 sets:   .word 1
28        ;ONE
29 one:    .word 1
30
31    .text
32 start: LW   r1, ca(r0)              ; puntatore colonne a
33        LW   r2, ra(r0)              ; puntatore righe a
34        LW   r3, cb(r0)              ; puntatore colonne b
35        LW   r4, rb(r0)              ; puntatore righe b
36        DADDI   r5, r0, a            ; puntatore a primo
      elemento a
37        DADDI   r6, r0, b            ; puntatore a primo
      elemento b
38        DADDI   r7, r0, c            ; puntatore a primo
      elemento c
39        LW   r8, inc(r0)             ; puntatore a grandezza
      in bit
40        LW   r9, i(r0)               ; puntatore indice i
41        LW   r10, j(r0)              ; puntatore indice j
42        LW   r11, k(r0)              ; puntatore indice k
```

```
43          LW   r12, sets(r0)              ; puntatore a set
44          LW   r16, one(r0)               ; puntatore a numero 1
45          DADDI    r17, r6, 0             ; puntatore a primo
    elemento b
46          DADDI    r18, r7, 0             ; puntatore a primo
    elemento c
47          L.D  f3, p(r0)                  ; inizializzazione
    parziale
48  validate:  SLT r12, r1, r16             ; controlla se colonne a
    maggiore di 0
49          BNEZ r12, end                   ; se < 1 termina
50          SLT r12, r2, r16                ; controlla se righe a
    maggiore di 0
51          BNEZ r12, end                   ; se < 1 termina
52          SLT r12, r3, r16                ; controlla se colonne b
    maggiore di 0
53          BNEZ r12, end                   ; se < 1 termina
54          SLT r12, r4, r16                ; controlla se righe b
    maggiore di 0
55          BNEZ r12, end                   ; se < 1 termina
56          BNE  r1, r4, end                ; controlla se CA = RB
57  loop_i: LW r10, j(r0)                   ; resetta j
58  loop_j: LW r11, k(r0)                   ; resetta k
59  loop_k: SLT r12, r11, r16               ; controlla se k < 1
60      if_ge2:   DMUL r15, r9, r1          ; i * CA
61          DMUL r19, r11, r3               ; k * CB
62          DADD r15, r15, r11              ; (i * CA) + k
63          DADD r19, r19, r10              ; (k * CB) + j
64          DMUL r15, r15, r8               ; ((i * CA) + k) * 8
65          DMUL r19, r19, r8               ; ((k * CB) + j) * 8
66          DADD r19, r17, r19              ; primo elemento b +
    indicizzazione
67          DADDI r5, r15, 0                ; aggiorna puntatore ad
    elemento a
68          DADDI r6, r19, 0                ; aggiorna puntatore ad
    elemento b
69          L.D f0, 0(r5)                   ; leggi elemento in A
70          L.D f1, 0(r6)                   ; leggi elemento in B
71          MUL.D f0, f0, f1                ; moltiplicare elementi
    di A e B
72          ADD.D f3, f3, f0                ; sommare risultato
    parziale
73          DADDI r11, r11, 1               ; incrementare k
74          BEQZ r12, pointers              ; salta singola
    ripetizione se k > 1
75        if_lt2: DMUL r15, r9, r1          ; i * CA
76          DMUL r19, r11, r3               ; k * CB
77          DADD r15, r15, r11              ; (i * CA) + k
78          DADD r19, r19, r10              ; (k * CB) + j
```

```
79          DMUL r15, r15, r8           ; ((i * CA) + k) * 8
80          DMUL r19, r19, r8           ; ((k * CB) + j) * 8
81          DADD r19, r17, r19          ; primo elemento b +
    indicizzazione
82          DADDI r5, r15, 0            ; aggiorna puntatore ad
    elemento a
83          DADDI r6, r19, 0            ; aggiorna puntatore ad
    elemento b
84          L.D f0, 0(r5)               ; leggi elemento in A
85          L.D f1, 0(r6)               ; leggi elemento in B
86          MUL.D f0, f0, f1            ; moltiplicare elementi
    di A e B
87          ADD.D f3, f3, f0            ; sommare risultato
    parziale
88          DADDI r11, r11, 1           ; incrementare k
89     pointers:  BNE r11, r1, loop_k   ; saltare se k non
    uguale a colonne_A
90          S.D f3, 0(r7)               ; scrivere risultato in C
91          MUL.D f3, f3, f4            ; resetta risultato
    parziale
92          DADDI r7, r7, 8             ; scorri risultati
93          DADDI r10, r10, 1           ; incrementare j
94          BNE r10, r3, loop_j         ; saltare se j non
    uguale a colonne_B
95          DADDI r9, r9, 1             ; incrementare i
96          BNE r9, r2, loop_i          ; saltare se i non
    uguale a righe_A
97 end: HALT
```

Nella settima versione la specifica è stata trattata in modo più adattabile possibile, come conseguenza, l'indicizzazione della matrice non può più essere trattata in modo Statico come nelle precedenti versioni bensì in modo Dinamico, causando molti stalli. Per arrivare ad una versione accettabile, ho effettuato un *Unroll Parziale* che mantiene l'adattabilità del codice e ne riduce il numero di cicli di Clock. In seguito sono riportate le statistiche ottenute:

Execution	
Cycles	1132
Instructions	602
CPI	1.884
Stalls	
Raw Stalls	546
WAW Stalls	0
WAR Stalls	24
Structural Stalls	198
Branch Taken Stalls	39
Branch Misprediction Stalls	0
Code Size	
Bytes	268

Tabella 11: Statistiche versione attuale

Execution	
Cycles	+997
Instructions	+493
CPI	+0.77
Stalls	
Raw Stalls	+546
WAW Stalls	0
WAR Stalls	+24
Structural Stalls	+186
Branch Taken Stalls	+38
Branch Misprediction Stalls	0
Code Size	
Bytes	+20

Tabella 12: Confronto versione precedente

4 Conclusione

Dalla prima alla sesta versione vi è un decremento di 334 Cicli, in cui le variazioni più importanti sono avvenute grazie a ottimizzazioni come il Register Renaming, che eliminano molte delle dipendenze. Con la Versione Dinamica del codice si è arrivati ad un compromesso accettabile tra adattabilità e Cicli, rendendola la versione più utilizzabile universalmente ma chiaramente quella che richiede un tempo maggiore.