Relazione

Progetto per la sessione d'esame autunnale
2023 / 2024

Automa cellulare

AUTORI

Sestri Daniele
matricola: 320713

Stoimenova Erika
matricola: 329648

Università degli Studi di Urbino Carlo Bo
Insegnamento di Architettura degli Elaboratori

Contents

1 Specifica **3**
1.1 Scopo del progetto . . . 3
1.2 Specifica funzionale . . . 3
1.2.1 Cos'è un automa cellulare . . . 3
1.2.2 Riferimenti . . . 3
1.2.3 Presentazione del nostro automa . . . 3
1.3 Specifica algoritmica . . . 4
1.4 Benchmark . . . 5

2 Prima implementazione in assembly **6**
2.1 Classi di stalli rilevati . . . 8

3 Ottimizzazione statica **8**
3.1 Versione 1 . . . 8
3.2 Versione 2 . . . 11
3.3 Versione 3 . . . 15

4 Conclusioni **19**

1 Specifica

1.1 Scopo del progetto

Si intende produrre un listato per l'esecuzione di una **singola evoluzione** di un **automa cellulare**, partendo da una griglia 4x4 gestita come array monodimensionale. Successivamente, verranno implementate diverse ottimizzazioni statiche del codice e ne verrà analizzata l'efficienza.

Per lo sviluppo e la valutazione utilizzeremo il software didattico WinMIPS64, basato sull'architettura DLX.

1.2 Specifica funzionale

1.2.1 Cos'è un automa cellulare

Un automa cellulare è un sistema che simula l'evoluzione di una popolazione di celle disposte in una griglia. Ogni cella può trovarsi in uno stato specifico, il quale evolve in funzione di regole definite, basate sullo stato delle celle adiacenti.

1.2.2 Riferimenti

L'automa cellulare, oggetto di questa relazione è stato liberamente ispirato da *"Game of Life"* creato dal matematico britannico John Conway nel 1970.

Il Game of Life è un sistema dinamico in cui le celle di una griglia bidimensionale evolvono nel tempo in base a regole semplici, ma che possono generare comportamenti complessi e imprevedibili. Ogni cella può assumere due stati: viva o morta, e il destino di ciascuna cella è determinato dallo stato delle celle adiacenti.
La popolarità di questo modello matematico risiede nella sua capacità di simulare modelli di crescita, interazione e decadimento che ricordano fenomeni reali, come la proliferazione di organismi o l'evoluzione di sistemi complessi.

1.2.3 Presentazione del nostro automa

La griglia della popolazione è rappresentata da un array monodimensionale con un numero di elementi pari, superiore a 2. In questa relazione, si utilizza una struttura predefinita composta da 16 celle, ciascuna delle quali contiene un valore di tipo float. Tali valori rappresentano diversi "gradi" di vita delle celle.

In questo contesto:

- Un valore pari a 0.0 indica che la cella è morta;
- Un valore diverso da 0.0 rappresenta una cella viva.

La singola evoluzione di questa popolazione avviene in due passaggi:

1. **Aggiornamento iniziale dello stato delle celle**
   (serve a simulare un decadimento naturale che porta alla morte.)
   - per ogni cella, il valore corrente viene decrementato di 0.1;
   - Se, dopo il decremento, il valore della cella risulta minore o uguale a 0.3, la cella muore.

2. **Applicazione delle regole di evoluzione**
   (evoluzione basata sul conteggio dei vicini vivi di ciascuna cella, limitato ai vicini orizzontali: il vicino a sinistra e il vicino a destra)

   - Se la cella viva ha zero vicini vivi, la cella muore per solitudine;
   - Se la cella morta ha esattamente due vicini vivi, la cella rinasce;
   - Se almeno un vicino esiste (non si va fuori dai limiti della griglia) e il suo valore è diverso da 0.0, allora la cella rimane in vita.

1.3 Specifica algoritmica

```c
#include <stdio.h>

int main() {
    float grid[16] = {
        1.3, 0.7, 0.5, 0.2,
        0.6, 0.9, 0.3, 0.0,
        0.8, 1.2, 0.1, 0.7,
        0.3, 0.2, 0.5, 0.1
    };

    for (int i = 0; i < 16; i++) {
        grid[i] -= 0.1;
        if (grid[i] <= 0.3)
            grid[i] = 0.0;
    }

    for (int i = 0; i < 16; i++) {
       int alive_neighbors = 0;
       int j = i - 1;
       int w = i + 1;

       if(
        j >= 0 &&
        grid[j] != 0
        )
          alive_neighbors++;

        if(
        w <= 15 &&
        grid[w] != 0
        )
          alive_neighbors++;

        if (grid[i] != 0) {
            if (alive_neighbors == 0)
                grid[i] = 0.0;
        } else {
            if (alive_neighbors == 2)
                grid[i] = 1.0;
        }
    }
```

```
42
43      return 0;
44  }
```

RISULTATO

```
1.2     0.6     0.4     1.0
0.5     0.8     0.0     0.0
0.7     1.1     1.0     0.6
0.0     0.0     0.0     0.0
```

1.4 Benchmark

L'architettura del WinMIPS64 è quella di default.

Code Address Bus	**10**
Data Address Bus	**10**
FP Addition Latency	**4**
Multiplier Latency	**7**
Division Latency	**24**
Forwarding	**yes**

Dimensione Il numero degli elementi dell'array deve essere pari e maggiore di 2. In questo contesto utilizziamo una struttura pre-impostata con 16 elementi.

Tipo dei dati I valori delle celle sono di tipo double e vanno gestiti come tali.

Vicini le regole di aggiornamento considerano solo il vicino di destra e il vicino di sinistra (se esistono).

Aggiornamento sul posto L'algoritmo aggiorna la griglia direttamente, senza l'uso di strutture temporanee.

Si mette in evidenza che:

- Le regole di funzionamento sono riportate nella sezione 1.2.3;
- Il registro **r0** è utilizzato per rappresentare zero in operazioni con interi.
- Il registro **f0** viene usato come zero per le operazioni in virgola mobile.

2 Prima implementazione in assembly

```
.data

sub:    .double 0.1   ; valore decadimento
soglia: .double 0.3   ; valore di soglia
renew:  .double 1.0   ; valore rinascita
cont:   .word 16      ; contatore / numero elementi

; valori della griglia
vett:  .double 1.3, 0.7, 0.5, 0.2, 0.6, 0.9, 0.3, 0.0, 0.8, 1.2, 0.1, 0.7
   , 0.3, 0.2, 0.5, 0.1

.code

LW r1, cont(r0)     ; carico il valore iniziale del contatore
LW r9, cont(r0)     ; carico il valore degli elementi
L.D f1, sub(r0)     ; carico il valore per la sottrazione
L.D f2, soglia(r0)  ; carico il valore di soglia
L.D f5, renew(r0)   ; carico il valore di rinascita
DADDI r2, r0, vett  ; puntatore al primo elemento dell'array

;************
; PASSAGGIO_1
;************
loopM:
  L.D f3, 0(r2)       ; leggi valore cella
  SUB.D f3, f3, f1    ; decadimento del valore della cella
  C.LE.D f3, f2       ; f3 <= 0.3
  BC1F skip_0         ; se falso salta
  MOV.D f3, f0        ; la cella corrente muore
skip_0:
  S.D f3, 0(r2)       ; salvataggio risultato
  DADDI r2, r2, 8     ; passo all'elemento successivo
  DADDI r1, r1, -1    ; decrementa contatore
BNEZ r1, loopM

;************
; PASSAGGIO_2
;************
LW r1, cont(r0)     ; carico il valore iniziale del contatore
DADDI r10, r0, 0    ; imposto il valore iniziale della posizione
DADDI r2, r0, vett  ; puntatore al primo elemento dell'array
DADDI r8, r0, 2     ; Carica il valore 2 in r8
loopE:
  DADDI r3, r0, 0    ; inizializzazione contatore vicini vivi ->
     alive_neighbors
  DADDI r4, r10, -1  ; posizione vicino di sinistra -> j
  DADDI r5, r10, 1   ; posizione vicino di destra -> w

  SLT r6, r4, r0       ; r6 = 1 se r4 < 0 altrimenti r6 = 0
  BNEZ r6, skip_1     ; se non è uguale a zero fa il salto. --> j >= 0
    DADDI r7, r2, -8  ; puntatore di sinistra
```

```
50      L.D f4, 0(r7)       ; leggi valore del vicino di sinistra -> grid[j]
51      C.EQ.D f4, f0       ; grid[j] == 0
52      BC1T skip_1         ; se vera fa il salto
53        DADDI r3, r3, 1   ; alive_neighbors++
54
55  skip_1:
56    SLT r6, r5, r9        ; r6 = 1 se r5 < 16 altrimenti r6 = 0
57    BEQZ r6, skip_2       ; se è uguale a zero fa il salto. --> w <= 15
58      DADDI r7, r2, 8     ; puntatore di destra
59      L.D f4, 0(r7)       ; leggi valore del vicino di destra -> grid[w]
60      C.EQ.D f4, f0       ; grid[w] == 0
61      BC1T skip_2         ; se vera fa il salto
62        DADDI r3, r3, 1   ; alive_neighbors++
63
64  skip_2:
65    L.D f3, 0(r2)       ; leggi valore cella corrente -> grid[i]
66    C.EQ.D f3, f0       ; grid[i] == 0
67    BC1T skip_3         ; se vera fa il salto
68    BNEZ r3, skip_4     ; se non è uguale a zero fa il salto -->
         alive_neighbors != 0
69      S.D f0, 0(r2)     ; setta la cella corrente come morta
70      J skip_4          ; salto alla fine del ciclo
71
72  skip_3:
73    BNE r3, r8, skip_4    ; Se r3 != r8, salta --> alive_neighbors != 2
74      S.D f5, 0(r2)       ; setta la cella corrente come rinata
75
76  skip_4:
77    DADDI r2, r2, 8       ; passo all'elemento successivo
78    DADDI r10, r10, 1     ; successiva posizione
79    DADDI r1, r1, -1      ; decrementa contatore
80
81  BNEZ r1, loopE
82
83  HALT ; Termina il programma
```

Execution
780 Cycles
528 Instructions
1.477 Cycles Per Instruction (CPI)

Stalls
174 RAW Stalls
0 WAW Stalls
0 WAR Stalls
16 Structural Stalls
74 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
196 Bytes

2.1 Classi di stalli rilevati

- Il codice presentato causa uno stallo di tipo RAW (Read After Write). Inoltre, introduce uno stallo strutturale, poiché la latenza dell'istruzione SUB.D è di 4 cicli di clock, il che significa che l'istruzione C.LE.D deve attendere il completamento di SUB.D prima di poter procedere alla fase di MA (Memory Access):

```
L.D. f3, 0(r2)
SUB.D f3, f3, f1    ; decadimento del valore della cella
C.LE.D f3, f2       ; f3 <= 0.3
```

- I *Branch Taken Stall* sono dovuti alle istruzione di salto *BC1F*, *BC1T*, *BNE*, *BNEZ* e *BEQZ*;

- Le coppie di istruzioni qui riportate producono uno stallo RAW:

```
DADDI r1, r1, -1    ; decrementa contatore
BNEZ r1, loopM
--------------------------
SLT r6, r4, r0       ; r6 = 1 se r4 < 0 altrimenti r6 = 0
BNEZ r6, skip_1      ; se non è uguale a zero fa il salto. --> j >= 0
--------------------------
SLT r6, r5, r9       ; r6 = 1 se r5 < 16 altrimenti r6 = 0
BEQZ r6, skip_2      ; se è uguale a zero fa il salto. --> w <= 15
--------------------------
DADDI r1, r1, -1      ; decrementa contatore
BNEZ r1, loopE
--------------------------
L.D f4, 0(r7)       ; leggi valore cella del vicino di sinistra -> grid[j]
C.EQ.D f4, f0       ; grid[j] == 0
--------------------------
L.D f3, 0(r2)       ; leggi valore cella corrente -> grid[i]
C.EQ.D f3, f0       ; grid[i] == 0
```

3 Ottimizzazione statica

3.1 Versione 1

In questa versione dell'assembly applichiamo come ottimizzazione statica il solo instruction reordering.

```
.data

sub:    .double 0.1    ; valore decadimento
soglia: .double 0.3    ; valore di soglia
renew:  .double 1.0    ; valore rinascita
cont:   .word 16       ; contatore / numero elementi

; valori della griglia
```

```
vett:  .double 1.3, 0.7, 0.5, 0.2, 0.6, 0.9, 0.3, 0.0, 0.8, 1.2, 0.1, 0.7
   , 0.3, 0.2, 0.5, 0.1

.code

LW r1, cont(r0)     ; carico il valore iniziale del contatore
LW r9, cont(r0)     ; carico il valore degli elementi
L.D f1, sub(r0)     ; carico il valore per la sottrazione
L.D f2, soglia(r0)  ; carico il valore di soglia
L.D f5, renew(r0)   ; carico il valore di rinascita
DADDI r2, r0, vett  ; puntatore al primo elemento dell'array

;************
; PASSAGGIO_1
;************
loopM:
  L.D f3, 0(r2)      ; leggi valore cella
  DADDI r1, r1, -1   ; decrementa contatore
  SUB.D f3, f3, f1   ; decadimento del valore della cella
  C.LE.D f3, f2      ; f3 <= 0.3
  BC1F skip_0        ; se falso salta
  MOV.D f3, f0       ; la cella corrente muore
skip_0:
  S.D f3, 0(r2)      ; salvataggio risultato
  DADDI r2, r2, 8    ; passo all'elemento successivo
BNEZ r1, loopM

;************
; PASSAGGIO_2
;************
LW r1, cont(r0)     ; carico il valore iniziale del contatore
DADDI r10, r0, 0    ; imposto il valore iniziale della posizione
DADDI r2, r0, vett  ; puntatore al primo elemento dell'array
DADDI r8, r0, 2     ; Carica il valore 2 in r8
loopE:
  DADDI r3, r0, 0    ; inizializzazione contatore vicini vivi ->
     alive_neighbors
  DADDI r4, r10, -1  ; posizione vicino di sinistra -> j

  SLT r6, r4, r0      ; r6 = 1 se r4 < 0 altrimenti r6 = 0
  DADDI r5, r10, 1    ; posizione vicino di destra -> w
  BNEZ r6, skip_1     ; se non è uguale a zero fa il salto. --> j >= 0
    DADDI r7, r2, -8  ; puntatore di sinistra
    L.D f4, 0(r7)     ; leggi valore del vicino di sinistra -> grid[j]
    C.EQ.D f4, f0     ; grid[j] == 0
    BC1T skip_1       ; se vera fa il salto
      DADDI r3, r3, 1   ; alive_neighbors++

skip_1:
  SLT r6, r5, r9      ; r6 = 1 se r5 < 16 altrimenti r6 = 0
  DADDI r1, r1, -1    ; decrementa contatore
  BEQZ r6, skip_2     ; se è uguale a zero fa il salto. --> w <= 15
    DADDI r7, r2, 8   ; puntatore di destra
```

```
60      L.D f4, 0(r7)      ; leggi valore del vicino di destra -> grid[w]
61      C.EQ.D f4, f0      ; grid[w] == 0
62      BC1T skip_2        ; se vera fa il salto
63        DADDI r3, r3, 1    ; alive_neighbors++
64
65  skip_2:
66    L.D f3, 0(r2)      ; leggi valore cella corrente -> grid[i]
67    C.EQ.D f3, f0      ; grid[i] == 0
68    BC1T skip_3        ; se vera fa il salto
69    BNEZ r3, skip_4    ; se non è uguale a zero fa il salto -->
         alive_neighbors != 0
70      S.D f0, 0(r2)    ; setta la cella corrente come morta
71      J skip_4         ; salto alla fine del ciclo
72
73  skip_3:
74    BNE r3, r8, skip_4    ; Se r3 != r8, salta --> alive_neighbors != 2
75      S.D f5, 0(r2)       ; setta la cella corrente come rinata
76
77  skip_4:
78    DADDI r2, r2, 8       ; passo all'elemento successivo
79    DADDI r10, r10, 1     ; successiva posizione
80
81  BNEZ r1, loopE
82
83  HALT ; Termina il programma
```

Nel passaggio 1 abbiamo spostato il decremento del contatore inserendolo tra L.D e SUB.D su f3: con questo passaggio elimino tutti gli stalli RAW su f3 in questa porzione di codice.

```
1  L.D f3, 0(r2)       ; leggi valore cella
2  DADDI r1, r1, -1    ; decrementa contatore
3  SUB.D f3, f3, f1    ; decadimento del valore della cella
```

Nel passaggio 2 abbiamo spostato il calcolo della posizione del vicino di destra e posizionato tra SLT e BNEZ per evitare uno stallo RAW su r6.

```
1  DADDI r3, r0, 0     ; inizializzazione contatore vicini vivi ->
      alive_neighbors
2  DADDI r4, r10, -1   ; posizione vicino di sinistra -> j
3  SLT r6, r4, r0        ; r6 = 1 se r4 < 0 altrimenti r6 = 0
4  DADDI r5, r10, 1    ; posizione vicino di destra -> w
5  BNEZ r6, skip_1      ; se non è uguale a zero fa il salto. --> j >= 0
```

Abbiamo spostato nello skip 1 del passaggio 2 il decremento del contatore del ciclo. Procedendo come descritto abbiamo risolto tutti gli stalli RAW di r6 in questa porzione ed inoltre abbiamo risolto gli stalli RAW del salto del loop.

```
1  SLT r6, r5, r9       ; r6 = 1 se r5 < 16 altrimenti r6 = 0
2  DADDI r1, r1, -1     ; decrementa contatore
3  BEQZ r6, skip_2      ; se è uguale a zero fa il salto. --> w <= 15
4  ----------------------------------------
5  DADDI r2, r2, 8       ; passo all'elemento successivo
6  DADDI r10, r10, 1     ; successiva posizione
7  BNEZ r1, loopE
```

Execution
700 Cycles
528 Instructions
1.326 Cycles Per Instruction (CPI)

Stalls
78 RAW Stalls
0 WAW Stalls
0 WAR Stalls
16 Structural Stalls
74 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
196 Bytes

3.2 Versione 2

In questa versione dell'assembly applichiamo come ottimizzazione statica il loop unrolling parziale, il register renaming e l'instruction reordering.

```
.data

sub:    .double 0.1    ; valore decadimento
soglia: .double 0.3    ; valore di soglia
renew:  .double 1.0    ; valore rinascita
cont:   .word 16       ; contatore / numero elementi

; valori della griglia
vett:  .double 1.3, 0.7, 0.5, 0.2, 0.6, 0.9, 0.3, 0.0, 0.8, 1.2, 0.1, 0.7
   , 0.3, 0.2, 0.5, 0.1

.code

LW r1, cont(r0)      ; carico il valore iniziale del contatore
LW r9, cont(r0)      ; carico il valore degli elementi
L.D f1, sub(r0)      ; carico il valore per la sottrazione
L.D f2, soglia(r0)   ; carico il valore di soglia
L.D f5, renew(r0)    ; carico il valore di rinascita
DADDI r2, r0, vett   ; puntatore al primo elemento dell'array

;************
; PASSAGGIO_1
;************
loopM:
  L.D f3, 0(r2)       ; leggi valore cella
  DADDI r1, r1, -1    ; decrementa contatore
  SUB.D f3, f3, f1    ; decadimento del valore della cella
  DADDI r2, r2, 8     ; passo all'elemento successivo
  L.D f6, 0(r2)       ; leggi valore cella

```

```
30    SUB.D f6, f6, f1    ; decadimento del valore della cella
31    DADDI r1, r1, -1    ; decrementa contatore
32
33    C.LE.D f3, f2       ; f3 <= 0.3
34    BC1F skip0          ; se falso salta
35    MOV.D f3, f0        ; la cella corrente muore
36  skip0:
37    S.D f3, -8(r2)       ; salvataggio risultato
38
39    C.LE.D f6, f2       ; f6 <= 0.3
40    BC1F skip1         ; se falso salta
41    MOV.D f6, f0        ; la cella corrente muore
42  skip1:
43    S.D f6, 0(r2)       ; salvataggio risultato
44    DADDI r2, r2, 8     ; passo all'elemento successivo
45
46  BNEZ r1, loopM
47
48  ;************
49  ; PASSAGGIO_2
50  ;************
51  LW r1, cont(r0)      ; carico il valore iniziale del contatore
52  DADDI r10, r0, 0     ; imposto il valore iniziale della posizione
53  DADDI r2, r0, vett   ; puntatore al primo elemento dell'array
54  DADDI r8, r0, 2      ; Carica il valore 2 in r8
55  loopE:
56    DADDI r3, r0, 0    ; inizializzazione contatore vicini vivi ->
        alive_neighbors
57
58    DADDI r4, r10, -1  ; posizione vicino di sinistra -> j
59    DADDI r5, r10, 1   ; posizione vicino di destra -> w
60    SLT r6, r4, r0     ; r6 = 1 se r4 < 0 altrimenti r6 = 0
61    SLT r15, r5, r9    ; r15 = 1 se r5 < 16 altrimenti r15 = 0
62
63    ; la posizione di sinistra dell'elemento successivo è la posizione dell
        'elemento corrente quindi DADDI r12, r10, 0, ma si pu  omettere
        perch  r10 è la posizione corrente.
64    DADDI r13, r10, 2   ; posizione vicino di destra -> w
65
66    ; set if less than immediate (online è sbagliato)
67    SLTI r11, r10, 1    ; r11 = 1 se r10 < 1 altrimenti r11 = 0
68
69    SLT r14, r13, r9    ; r14 = 1 se r13 < 16 altrimenti r14 = 0
70
71    BNEZ r6, skip2       ; se non è uguale a zero fa il salto.
72      DADDI r7, r2, -8   ; puntatore di sinistra
73      L.D f4, 0(r7)      ; leggi valore del vicino di sinistra -> grid[j]
74      C.EQ.D f4, f0      ; grid[j] == 0
75      BC1T skip2         ; se vera fa il salto
76        DADDI r3, r3, 1  ; alive_neighbors++
77
78  skip2:
79    BEQZ r15, skip3       ; se è uguale a zero fa il salto.
```

```
80      DADDI r7, r2, 8      ; puntatore di destra
81      L.D f4, 0(r7)        ; leggi valore del vicino di destra -> grid[w]
82      C.EQ.D f4, f0        ; grid[w] == 0
83      BC1T skip3           ; se vera fa il salto
84        DADDI r3, r3, 1    ; alive_neighbors++
85
86  skip3:
87    L.D f3, 0(r2)      ; leggi valore cella corrente -> grid[i]
88    DADDI r1, r1, -1   ; decrementa contatore
89    C.EQ.D f3, f0      ; grid[i] == 0
90    BC1T skip4         ; se vera fa il salto
91    BNEZ r3, skip5     ; se non è uguale a zero fa il salto -->
        alive_neighbors != 0
92      S.D f0, 0(r2)    ; setta la cella corrente come morta
93      J skip5          ; salto alla fine del ciclo
94
95  skip4:
96    BNE r3, r8, skip5     ; Se r3 != r8, salta --> alive_neighbors != 2
97      S.D f5, 0(r2)       ; setta la cella corrente come rinata
98
99  skip5:
100   DADDI r2, r2, 8       ; passo all'elemento successivo
101   DADDI r10, r10, 1     ; successiva posizione
102   DADDI r1, r1, -1      ; decrementa contatore
103
104   DADDI r3, r0, 0      ; inizializzazione contatore vicini vivi ->
        alive_neighbors
105
106   BNEZ r11, skip6        ; se non è uguale a zero fa il salto.
107     DADDI r7, r2, -8     ; puntatore di sinistra
108     L.D f4, 0(r7)        ; leggi valore del vicino di sinistra -> grid[j]
109     C.EQ.D f4, f0        ; grid[j] == 0
110     BC1T skip6           ; se vera fa il salto
111       DADDI r3, r3, 1    ; alive_neighbors++
112
113 skip6:
114   BEQZ r14, skip7       ; se è uguale a zero fa il salto. --> w <= 15
115     DADDI r7, r2, 8     ; puntatore di destra
116     L.D f4, 0(r7)       ; leggi valore del vicino di destra -> grid[w]
117     C.EQ.D f4, f0       ; grid[w] == 0
118     BC1T skip7          ; se vera fa il salto
119       DADDI r3, r3, 1   ; alive_neighbors++
120
121 skip7:
122   L.D f3, 0(r2)      ; leggi valore cella corrente -> grid[i]
123   C.EQ.D f3, f0      ; grid[i] == 0
124   BC1T skip8         ; se vera fa il salto
125   BNEZ r3, skip9     ; se non è uguale a zero fa il salto -->
        alive_neighbors != 0
126     S.D f0, 0(r2)    ; setta la cella corrente come morta
127     J skip9          ; salto alla fine del ciclo
128
129 skip8:
```

```
130   BNE r3, r8, skip9    ; Se r3 != r8, salta --> alive_neighbors != 2
131     S.D f5, 0(r2)      ; setta la cella corrente come rinata
132
133 skip9:
134   DADDI r2, r2, 8       ; passo all'elemento successivo
135   DADDI r10, r10, 1     ; successiva posizione
136
137 BNEZ r1, loopE
138 HALT ; Termina il programma
```

Nel primo passaggio è stato applicato un unrolling parziale a due unità. Successivamente, è stato utilizzato il register renaming, seguito dall'ottimizzazione tramite instruction reordering. Questi interventi hanno significativamente ridotto gli stalli di tipo RAW, che rimangono presenti solo tra

```
1 L.D f6, 0(r2)       ; leggi valore cella
2 SUB.D f6, f6, f1    ; decadimento del valore della cella
```

e hanno eliminato gran parte degli stalli strutturali. Gli unici stalli strutturali residui si verificano quando l'istruzione BC1F skip0 non esegue il salto.

```
1  ;************
2  ; PASSAGGIO_1
3  ;************
4  loopM:
5    L.D f3, 0(r2)       ; leggi valore cella
6    DADDI r1, r1, -1    ; decrementa contatore
7    SUB.D f3, f3, f1    ; decadimento del valore della cella
8    DADDI r2, r2, 8     ; passo all'elemento successivo
9    L.D f6, 0(r2)       ; leggi valore cella
10
11   SUB.D f6, f6, f1    ; decadimento del valore della cella
12   DADDI r1, r1, -1    ; decrementa contatore
13
14   C.LE.D f3, f2       ; f3 <= 0.3
15   BC1F skip0          ; se falso salta
16   MOV.D f3, f0        ; la cella corrente muore
17 skip0:
18   S.D f3, -8(r2)       ; salvataggio risultato
19
20   C.LE.D f6, f2       ; f6 <= 0.3
21   BC1F skip1         ; se falso salta
22   MOV.D f6, f0        ; la cella corrente muore
23 skip1:
24   S.D f6, 0(r2)       ; salvataggio risultato
25   DADDI r2, r2, 8     ; passo all'elemento successivo
26 BNEZ r1, loopM
```

Nel secondo passaggio è stato effettuato un unrolling a due unità. Successivamente, è stato applicato il register renaming per risolvere le dipendenze di registro delle istruzioni SLT, consentendo di raggrupparle.
Si è ottimizzato il calcolo del posizionamento del secondo elemento dell'unrolling.
Infine, il decremento del contatore è stato riposizionato al fine di evitare uno stallo di tipo RAW.

```
L.D f3, 0(r2)      ; leggi valore cella corrente -> grid[i]
DADDI r1, r1, -1   ; decrementa contatore
C.EQ.D f3, f0      ; grid[i] == 0
```

Execution
602 Cycles
499 Instructions
1.206 Cycles Per Instruction (CPI)

Stalls
45 RAW Stalls
0 WAW Stalls
0 WAR Stalls
3 Structural Stalls
59 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
336 Bytes

3.3 Versione 3

```
.data

sub:     .double 0.1    ; valore decadimento
soglia: .double 0.3    ; valore di soglia
renew:   .double 1.0    ; valore rinascita
cont:    .word 16       ; contatore / numero elementi

; valori della griglia
vett:   .double 1.3, 0.7, 0.5, 0.2, 0.6, 0.9, 0.3, 0.0, 0.8, 1.2, 0.1, 0.7
    , 0.3, 0.2, 0.5, 0.1

.code

LW r1, cont(r0)      ; carico il valore iniziale del contatore
LW r9, cont(r0)      ; carico il valore degli elementi
L.D f1, sub(r0)      ; carico il valore per la sottrazione
L.D f2, soglia(r0)   ; carico il valore di soglia
L.D f5, renew(r0)    ; carico il valore di rinascita
DADDI r2, r0, vett   ; puntatore al primo elemento dell'array

;************
; PASSAGGIO_1
;************
loopM:
  L.D f3, 0(r2)       ; leggi valore cella
  L.D f6, 8(r2)       ; leggi valore cella
  DADDI r1, r1, -1    ; decrementa contatore
  SUB.D f3, f3, f1    ; decadimento del valore della cella
```

```
  DADDI r2, r2, 16    ; passo all'elemento successivo
  DADDI r1, r1, -1    ; decrementa contatore

  SUB.D f6, f6, f1    ; decadimento del valore della cella

  C.LE.D f3, f2       ; f3 <= 0.3
  BC1F skip0          ; se falso salta
  MOV.D f3, f0        ; la cella corrente muore
skip0:
  C.LE.D f6, f2       ; f6 <= 0.3
  BC1F skip1          ; se falso salta
  MOV.D f6, f0        ; la cella corrente muore
skip1:
  S.D f6, -8(r2)      ; salvataggio risultato
  S.D f3, -16(r2)     ; salvataggio risultato
BNEZ r1, loopM

;************
; PASSAGGIO_2
;************
LW r1, cont(r0)      ; carico il valore iniziale del contatore
DADDI r10, r0, 0     ; imposto il valore iniziale della posizione
DADDI r2, r0, vett   ; puntatore al primo elemento dell'array
DADDI r8, r0, 2      ; Carica il valore 2 in r8
loopE:
  DADDI r3, r0, 0    ; inizializzazione contatore vicini vivi ->
     alive_neighbors

  DADDI r4, r10, -1  ; posizione vicino di sinistra -> j
  DADDI r5, r10, 1   ; posizione vicino di destra -> w
  SLT r6, r4, r0     ; r6 = 1 se r4 < 0 altrimenti r6 = 0
  SLT r15, r5, r9    ; r15 = 1 se r5 < 16 altrimenti r15 = 0
  DADDI r13, r10, 2  ; posizione vicino di destra -> w
  SLTI r11, r10, 1   ; r11 = 1 se r10 < 1 altrimenti r11 = 0
  SLT r14, r13, r9   ; r14 = 1 se r13 < 16 altrimenti r14 = 0

  BNEZ r6, skip2       ; se non è uguale a zero fa il salto. --> j >= 0
    L.D f4, -8(r2)     ; leggi valore del vicino di sinistra -> grid[j]
    C.EQ.D f4, f0      ; grid[j] == 0
    BC1T skip2         ; se vera fa il salto
      DADDI r3, r3, 1  ; alive_neighbors++

skip2:
  BEQZ r15, skip3       ; se è uguale a zero fa il salto. --> w <= 15
    L.D f4, 8(r2)       ; leggi valore del vicino di destra -> grid[w]
    C.EQ.D f4, f0       ; grid[w] == 0
    BC1T skip3          ; se vera fa il salto
      DADDI r3, r3, 1   ; alive_neighbors++

skip3:
  L.D f3, 0(r2)      ; leggi valore cella corrente -> grid[i]
  DADDI r1, r1, -1   ; decrementa contatore
  C.EQ.D f3, f0      ; grid[i] == 0
```

```
80    BC1T skip4          ; se vera fa il salto
81    BNEZ r3, skip5      ; se non è uguale a zero fa il salto -->
         alive_neighbors != 0
82      S.D f0, 0(r2)     ; setta la cella corrente come morta
83      J skip5           ; salto alla fine del ciclo
84
85  skip4:
86    BNE r3, r8, skip5     ; Se r3 != r8, salta --> alive_neighbors != 2
87      S.D f5, 0(r2)       ; setta la cella corrente come rinata
88
89  skip5:
90    DADDI r2, r2, 8     ; passo all'elemento successivo
91    DADDI r3, r0, 0     ; inizializzazione contatore vicini vivi ->
         alive_neighbors
92
93    BNEZ r11, skip6       ; se non è uguale a zero fa il salto. --> j >= 0
94      L.D f4, -8(r2)      ; leggi valore del vicino di sinistra -> grid[j]
95      C.EQ.D f4, f0       ; grid[j] == 0
96      BC1T skip6          ; se vera fa il salto
97        DADDI r3, r3, 1   ; alive_neighbors++
98
99  skip6:
100   BEQZ r14, skip7        ; se è uguale a zero fa il salto. --> w <= 15
101     L.D f4, 8(r2)        ; leggi valore del vicino di destra -> grid[w]
102     C.EQ.D f4, f0        ; grid[w] == 0
103     BC1T skip7           ; se vera fa il salto
104       DADDI r3, r3, 1    ; alive_neighbors++
105
106 skip7:
107   L.D f3, 0(r2)       ; leggi valore cella corrente -> grid[i]
108   DADDI r1, r1, -1    ; decrementa contatore
109   C.EQ.D f3, f0       ; grid[i] == 0
110   BC1T skip8          ; se vera fa il salto
111   BNEZ r3, skip9      ; se non è uguale a zero fa il salto -->
         alive_neighbors != 0
112     S.D f0, 0(r2)     ; setta la cella corrente come morta
113     J skip9           ; salto alla fine del ciclo
114
115 skip8:
116   BNE r3, r8, skip9     ; Se r3 != r8, salta --> alive_neighbors != 2
117     S.D f5, 0(r2)       ; setta la cella corrente come rinata
118
119 skip9:
120   DADDI r2, r2, 8       ; passo all'elemento successivo
121   DADDI r10, r10, 2     ; avanzo posizione
122
123 BNEZ r1, loopE
124 HALT ; Termina il programma
```

Nel passaggio 1 abbiamo eseguito delle ottimizzazioni spostando una lettura del valore, raggruppato il salvataggio della memoria e posizionato diversamente i calcoli del puntatore dell'elemento.

```
1  ;************
```

```
; PASSAGGIO_1
;************
loopM:
  L.D f3, 0(r2)       ; leggi valore cella
  L.D f6, 8(r2)       ; leggi valore cella
  DADDI r1, r1, -1    ; decrementa contatore
  SUB.D f3, f3, f1    ; decadimento del valore della cella
  DADDI r2, r2, 8     ; passo all'elemento successivo
  DADDI r1, r1, -1    ; decrementa contatore

  SUB.D f6, f6, f1    ; decadimento del valore della cella
  DADDI r2, r2, 8     ; passo all'elemento successivo

  C.LE.D f3, f2       ; f3 <= 0.3
  BC1F skip0       ; se falso salta
  MOV.D f3, f0        ; la cella corrente muore
skip0:
  C.LE.D f6, f2       ; f6 <= 0.3
  BC1F skip1        ; se falso salta
  MOV.D f6, f0        ; la cella corrente muore
skip1:
  S.D f6, -8(r2)       ; salvataggio risultato
  S.D f3, -16(r2)       ; salvataggio risultato

BNEZ r1, loopM
```

Nel passaggio 2 abbiamo tolto l'aggiornamento del puntatore utilizzando i valori immediati.

```
skip2:
  BEQZ r15, skip3        ; se è uguale a zero fa il salto. --> w <= 15
  L.D f4, 8(r2)        ; leggi valore del vicino di destra -> grid[w]
  C.EQ.D f4, f0        ; grid[w] == 0
```

Abbiamo spostato il decremento del contatore e posizionato in modo tale da far evitare stalli RAW in questi tipi di istruzioni

```
skip3:
  L.D f3, 0(r2)      ; leggi valore cella corrente -> grid[i]
  DADDI r1, r1, -1   ; decrementa contatore
  C.EQ.D f3, f0      ; grid[i] == 0
```

Abbiamo eliminato un avanzo della posizione e l'abbiamo accorpato con l'altro avanzo della posizione

```
skip9:
  DADDI r2, r2, 8      ; passo all'elemento successivo
  DADDI r10, r10, 2    ; avanzo posizione
```

Execution
549 Cycles
454 Instructions
1.209 Cycles Per Instruction (CPI)

Stalls
29 RAW Stalls

0 WAW Stalls
0 WAR Stalls
3 Structural Stalls
59 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
312 Bytes

4 Conclusioni

La restrizione di dover avere una dimensione dell'array pari e superiore a 2 impedisce l'applicazione di un unrolling superiore a due elementi. Pertanto, si può concludere che la versione v3 rappresenta la soluzione ottimale.

Versione	v0	v1	v2	v3
CPUC	780	700	602	549
Instruction	528	528	499	454
CPI	1.477	1.326	1.206	1.209
RAW Stalls	174	78	45	29
WAW Stalls	0	0	0	0
WAR Stalls	0	0	0	0
Structural Stalls	16	16	3	3
Branch Taken Stalls	74	74	59	59
Branch Misprediction Stalls	0	0	0	0
Code size	196 byte	196 byte	336 byte	312 byte

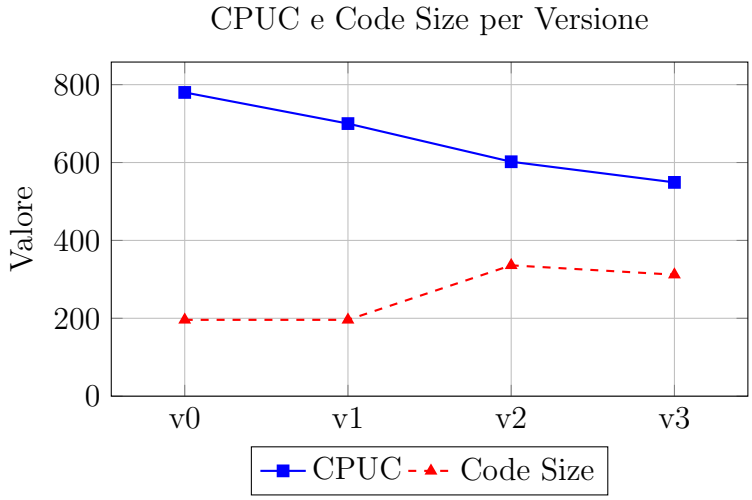

Figure 1: Confronto tra CPUC e Code Size per ciascuna versione.