

Progetto Backtracking

Struttura del codice

Sono stati prodotti 3 codici, uno che lavora in maniera lineare in singolo thread (`backtracking.cpp`) e altri due codici che lavorano in multithreading, (`backtracking_threaded.cpp` e `backtracking_list_threaded.cpp`).

Nel blocco del main dei programmi, mi costruisco un sudoku con delle celle vuote (gli zeri sono le celle vuote). La struttura in cui è messo il sudoku è una lista di liste (array). Questa struttura usa un indice per le colonne e uno per le righe, ho ricreato lo stesso codice usando anche un'unica grande lista per vedere anche il problema delle cache.

Una volta creato la struttura per il sudoku, lo passo alla funzione `backtracking` per iniziare a risolverlo.

Questa funzione usa la ricorsione. Cerca la posizione della prima cella vuota (contenente uno zero) chiamando la funzione `"next_pos_vuoto"`, questa in maniera ricorsiva cerca il primo zero presente nel sudoku. Tramite OpenMP, faccio eseguire in maniera parallela il ciclo for definendo il numero di iterazioni per thread definito dall'utente ad inizio esecuzione del programma parallelo.

Ottenuti gli indici (o l'indice per l'esecuzione con la lista), la funzione ricorsiva inizia a verificare i numeri, da 1 a 9, sulla posizione vuota, chiamando la funzione `"check_space"`. Questa funzione controlla se sulla riga, colonna o cella in cui è presente la posizione libera, esiste o meno il numero che si sta tentando di inserire.

Nel verificare la presenza del numero, in tutti e 3 i casi uso il parallelismo per controllare questa posizione.

La funzione restituisce un esito positivo se il numero non è presente, restituisce un esito negativo se è già presente il numero in una riga, colonna o nel quadrato.

Se da esito positivo, si inserisce il numero nella cella e si richiama ricorsivamente la funzione, questa troverà un nuovo numero libero, diverso da quello precedente.

Quando il ciclo per tentare i numeri da inserire finisce, avendo ottenuto tutti casi negativi nella ricorsione, quindi nessun numero può essere inserito, la funzione fallisce e torna alla ricorsione precedente sostituendo il numero già messo con uno nuovo.

Quando la funzione che cerca una cella libera termina non trovando più celle vuote, significa che il sudoku è completo e quindi deve completare tutte le ricorsioni, terminando così la funzione e restituendo un sudoku completo.

Limitazioni riscontrate

Avendo usato un algoritmo ricorsivo, non si è potuto usare una parallelizzazione sul ciclo per tentare i numeri da inserire, tuttavia sarebbe stato possibile usare un ciclo solo per tentare i numeri possibili e quindi poter parallelizzare la verifica e un secondo ciclo che usava i numeri trovati e ricorsivamente tentarli.

Un'altra limitazione è nella funzione che cerca la posizione vuota, questa viene chiamata molte volte e ogni volta inizia a cercare nuove posizioni sempre dall'inizio del sudoku.

Risultati delle prestazioni

| Approccio | Tempo (ms) | Cicli (milioni) | Page Fault | Context Switches |
|---|-------------------|------------------------|-------------------|-------------------------|
| <i>Singolo Thread (Array)</i> | 3.55 | 9 | 119 | 1 |
| <i>Multi Thread (Array) [2 thread]</i> | 59.67 | 153 | 147 | 3 |
| <i>Multi Thread (Array) [5 thread]</i> | 82.67 | 201 | 147 | 27 |
| <i>Multi Thread (Lista) [2 thread]</i> | 64.28 | 156 | 147 | 7 |
| <i>Multi Thread (Lista) [5 thread]</i> | 40.97 | 105 | 148 | 2 |
| <i>Multi Thread (Array ottimizzato)</i> | 281.71 | 658 | 148 | 42 |

Singolo Thread (Array)

Le prestazioni del codice a singolo thread risultano essere molto performanti:

```
4 2 5 1 8 9 3 6 7
6 7 9 2 3 4 5 1 8
3 1 8 6 5 7 2 4 9
1 6 4 3 7 2 9 8 5
8 9 3 5 4 1 6 7 2
2 5 7 8 9 6 1 3 4
5 4 2 7 6 3 8 9 1
9 3 1 4 2 8 7 5 6
7 8 6 9 1 5 4 2 3

Performance counter stats for './backtracking_run':

      3,55 msec task-clock                #    0,513 CPUs utilized
         1      context-switches         #    0,282 K/sec
         0      cpu-migrations            #    0,000 K/sec
        119     page-faults              #    0,034 M/sec
    9.491.871    cycles                   #    2,673 GHz
         0      stalled-cycles-frontend
         0      stalled-cycles-backend   #    0,00% backend cycles idle
<not counted>  instructions
<not counted>  branches
<not counted>  branch-misses

    0,006919836 seconds time elapsed

    0,005441000 seconds user
    0,000000000 seconds sys
```

Riuscendo a risolvere il sudoku in 3.55 millisecondi, con 9 milioni di cicli e 119 page faults

Multi Thread (Array)

D'altro canto, il codice parallelo risulta essere meno performante

```
sudo perf stat ./backtracking_threaded_run 2
4 2 5 1 8 9 3 6 7
6 7 9 2 3 4 5 1 8
3 1 8 6 5 7 2 4 9
1 6 4 3 7 2 9 8 5
8 9 3 5 4 1 6 7 2
2 5 7 8 9 6 1 3 4
5 4 2 7 6 3 8 9 1
9 3 1 4 2 8 7 5 6
7 8 6 9 1 5 4 2 3

Performance counter stats for './backtracking_threaded_run 2':

      59,67 msec task-clock              #    2,490 CPUs utilized
           3      context-switches       #    0,050 K/sec
           0      cpu-migrations          #    0,000 K/sec
          147      page-faults            #    0,002 M/sec
    153.020.900    cycles                  #    2,565 GHz              (6,47%)
           0      stalled-cycles-frontend #    0,00% backend cycles idle (47,20%)
           0      stalled-cycles-backend  #    0,00% backend cycles idle (87,71%)
           0      instructions            #    0,00 insn per cycle
                (93,53%)
           0      branches                 #    0,000 K/sec              (52,80%)
           0      branch-misses            #    0,00% of all branches   (12,29%)

    0,023966065 seconds time elapsed

    0,048437000 seconds user
    0,013839000 seconds sys
```

Riuscendo a risolvere il sudoku in 59.67 millisecondi, con 153 milioni di cicli e 147 page fault, incrementando il numero di thread per la ricerca di un nuovo spazio, il tempo peggiora.

Usando 5 thread

```
4 2 5 1 8 9 3 6 7
6 7 9 2 3 4 5 1 8
3 1 8 6 5 7 2 4 9
1 6 4 3 7 2 9 8 5
8 9 3 5 4 1 6 7 2
2 5 7 8 9 6 1 3 4
5 4 2 7 6 3 8 9 1
9 3 1 4 2 8 7 5 6
7 8 6 9 1 5 4 2 3

Performance counter stats for './backtracking_threaded_run 5':

      82,67 msec task-clock              #    2,460 CPUs utilized
          27      context-switches       #    0,327 K/sec
           3      cpu-migrations          #    0,036 K/sec
          147      page-faults            #    0,002 M/sec
    201.292.581    cycles                  #    2,435 GHz              (45,11%)
           0      stalled-cycles-frontend #    0,00% backend cycles idle (96,36%)
           0      stalled-cycles-backend  #    0,00% backend cycles idle
           0      instructions            #    0,00 insn per cycle
                (54,89%)
           0      branches                 #    0,000 K/sec              (3,64%)
<not counted>    branch-misses            #    0,00% of all branches   (0,00%)

    0,033601620 seconds time elapsed

    0,053161000 seconds user
    0,021264000 seconds sys
```

Multi Thread (Lista)

L'approccio con la lista sembra essere di poco più lento rispetto al test con l'array

```

sudo perf stat ./backtracking_threaded_list_run 2
4 2 5 1 8 9 3 6 7
6 7 9 2 3 4 5 1 8
3 1 8 6 5 7 2 4 9
1 6 4 3 7 2 9 8 5
8 9 3 5 4 1 6 7 2
2 5 7 8 9 6 1 3 4
5 4 2 7 6 3 8 9 1
9 3 1 4 2 8 7 5 6
7 8 6 9 1 5 4 2 3

Performance counter stats for './backtracking_threaded_list_run 2':

      64,28 msec task-clock                #    2,276 CPUs utilized
           7      context-switches        #    0,109 K/sec
           3      cpu-migrations          #    0,047 K/sec
          147      page-faults            #    0,002 M/sec
    156.653.676    cycles                  #    2,437 GHz                    (35,55%)
           0      stalled-cycles-frontend #                   (71,16%)
           0      stalled-cycles-backend  #    0,00% backend cycles idle    (97,88%)
           0      instructions             #    0,00 insn per cycle
                (64,45%)
           0      branches                 #    0,000 K/sec                    (28,84%)
           0      branch-misses            #    0,00% of all branches        (2,12%)

    0,028241046 seconds time elapsed

    0,047169000 seconds user
    0,017152000 seconds sys

```

Riuscendo a risolvere il problema in 64.28 millisecondi, con 156 milioni di cicli e 147 page fault

Usando 5 thread

```

4 2 5 1 8 9 3 6 7
6 7 9 2 3 4 5 1 8
3 1 8 6 5 7 2 4 9
1 6 4 3 7 2 9 8 5
8 9 3 5 4 1 6 7 2
2 5 7 8 9 6 1 3 4
5 4 2 7 6 3 8 9 1
9 3 1 4 2 8 7 5 6
7 8 6 9 1 5 4 2 3

Performance counter stats for './backtracking_threaded_list_run 5':

      40,97 msec task-clock                #    2,137 CPUs utilized
           2      context-switches        #    0,049 K/sec
           0      cpu-migrations          #    0,000 K/sec
          148      page-faults            #    0,004 M/sec
    105.404.573    cycles                  #    2,572 GHz                    (36,42%)
           0      stalled-cycles-frontend #                   (76,23%)
           0      stalled-cycles-backend  #    0,00% backend cycles idle    (98,20%)
           0      instructions             #    0,00 insn per cycle
                (63,58%)
           0      branches                 #    0,000 K/sec                    (23,77%)
           0      branch-misses            #    0,00% of all branches        (1,80%)

    0,019170888 seconds time elapsed

    0,032779000 seconds user
    0,010926000 seconds sys

```

Aumentando i thread del programma usando una lista, in base alla cpu, il programma può risultare più performante avendo bassi i “context-switches” e i “cpu-migrations”.

Multi Thread (Array ottimizzato)

Ho voluto tentare di compilare il programma multithread con il flag di ottimizzazione -O3

```
sudo perf stat ./backtracking_threaded_optimized_run 2
4 2 5 1 8 9 3 6 7
6 7 9 2 3 4 5 1 8
3 1 8 6 5 7 2 4 9
1 6 4 3 7 2 9 8 5
8 9 3 5 4 1 6 7 2
2 5 7 8 9 6 1 3 4
5 4 2 7 6 3 8 9 1
9 3 1 4 2 8 7 5 6
7 8 6 9 1 5 4 2 3

Performance counter stats for './backtracking_threaded_optimized_run 2':

      281,71 msec task-clock                #    3,047 CPUs utilized
         42      context-switches          #    0,149 K/sec
          0      cpu-migrations            #    0,000 K/sec
        148      page-faults              #    0,525 K/sec
    658.723.754 cycles                     #    2,338 GHz                    (47,30%)
          0      stalled-cycles-frontend   #                   (53,43%)
          0      stalled-cycles-backend    #    0,00% backend cycles idle   (52,54%)
          0      instructions              #    0,00  insn per cycle
                (52,70%)
          0      branches                  #    0,000 K/sec                (46,57%)
          0      branch-misses             #    0,00% of all branches      (47,46%)

    0,092455205 seconds time elapsed

    0,245673000 seconds user
    0,021550000 seconds sys
```

Il risultato è il più lento di quelli precedenti, avendo un numero elevato di cicli (658 milioni) e un numero alto di “context-switches” (42)