Relazione Homework 1: Sudoku

A cura di Sadman Sakib Rahman e Tiziano Natali

Matricole: 1632174, 1554933

L'IDEA

Essendo quello della ricerca della soluzione del sudoku un problema di tipo NP-completo, l'approccio algoritmico che abbiamo scelto di utilizzare è il backtracking, in quanto si presta bene a risolvere in tempo ottimale questo genere di problemi.

Per parallelizzare abbiamo semplicemente deciso di creare un processo parallelo per ogni nodo nell'albero del backtracking, in maniera ricorsiva.

L'IMPLEMENTAZIONE

Per implementare il backtracking abbiamo adottato, a grandi linee, il seguente modus operandi, applicato inizialmente ad un programma sequenziale, da poi "convertire" in un programma parallelo.

Preso in input la griglia di un sudoku sotto forma di matrice di interi, l'algoritmo esplora le celle della griglia dall'alto verso il basso, e successivamente da sinistra verso destra.

Per ogni cella, verifica, in caso sia vuota, i possibili valori che possono essere inseriti: se un valore è legale, esso viene inserito e viene effettuata una nuova chiamata al metodo. Se viene raggiunta l'ultima cella in basso a destra, il programma aumenta il numero delle soluzioni, altrimenti ogni cella esplorata viene resettata a zero.

In seguito, durante il passaggio alla versione parallela, abbiamo aggiunto alcune modifiche: innanzitutto, abbiamo fatto in modo che ogni singolo processo avesse in input una copia della matrice di input, in maniera tale da non modificare la stessa griglia con un accesso contemporaneo. Abbiamo quindi omesso il reset delle caselle, in quanto non era più necessario.

Dopo i primi test di esecuzione del codice parallelo abbiamo constatato un tempo di esecuzione maggiore rispetto all'esecuzione sequenziale, dovuta all'assenza di cutoff. Per ottimizzare abbiamo deciso di eseguire sequenzialmente il calcolo delle soluzioni dopo aver trovato almeno dieci caselle piene o riempite, dimezzando di conseguenza il tempo di esecuzione rispetto al programma sequenziale, nel caso vi fosse un basso fattore di riempimento della griglia. Nel caso in cui lo spazio delle soluzioni fosse piccolo, tuttavia, risulta più efficace il calcolo sequenziale.

PIATTAFORME

MacBook Pro di S. Sakib Rahman

Riepilogo hardware:

Nome modello:	MacBook Pro
Identificatore modello:	MacBookPro11,1
Nome processore:	Intel Core i5
Velocità processore:	2,4 GHz
Numero di processori:	1
Numero totale di Core:	2
Cache L2 (per Core):	256 KB
Cache L3:	3 MB
Memoria:	4 GB
Versione Boot ROM:	MBP111.0138.B18
Versione SMC (sistema):	2.16f68
Numero di serie (sistema):	C02LJ706FGYY
Hardware UUID:	CD0FF4A2-6E7C-54C8-AA40-8CD0122AFCEF

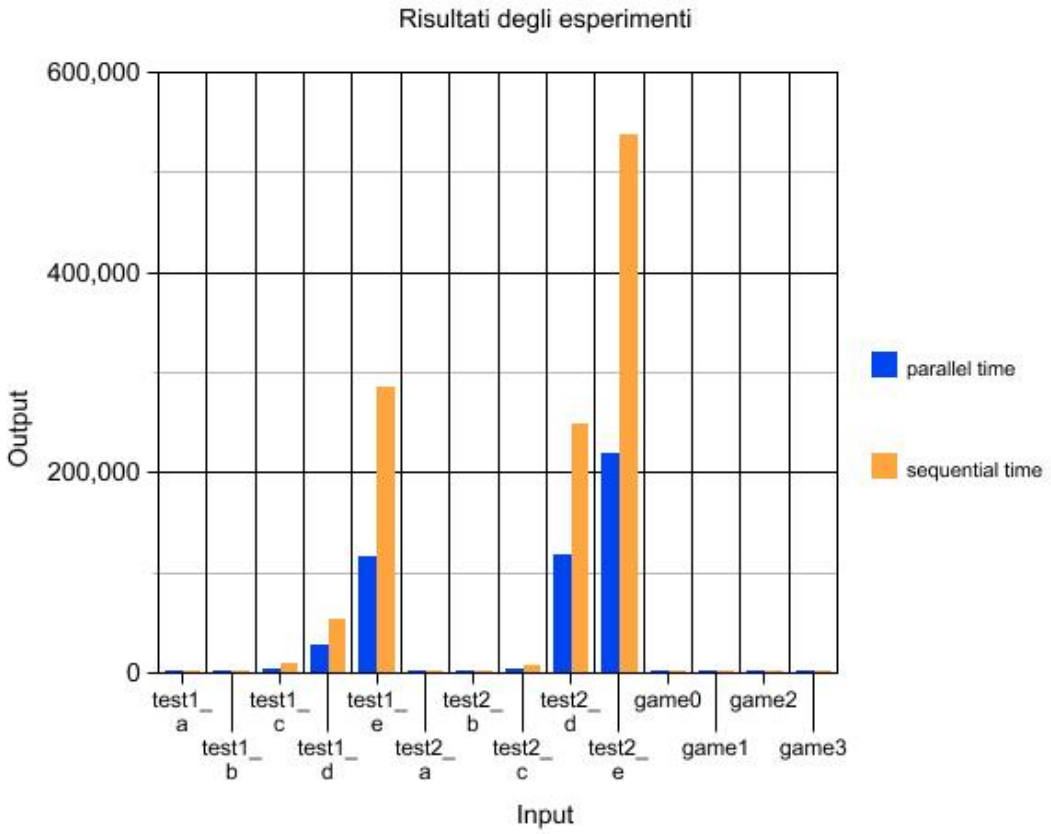
MacBook Pro di Tiziano Natali

Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro11,1
Processor Name:	Intel Core i5
Processor Speed:	2.6 GHz
Number of Processors:	1
Total Number of Cores:	2
L2 Cache (per Core):	256 KB
L3 Cache:	3 MB
Memory:	8 GB
Boot ROM Version:	MBP111.0138.B18
SMC Version (system):	2.16f68
Serial Number (system):	C02LT173FH01
Hardware UUID:	8C82C198-0439-5CE9-8DEC-24F9AE1ADBDE

DATI SUGLI ESPERIMENTI

Paragone tra tempo di esecuzione parallela e sequenziale su input diversi



Confronto dei dati di output

INPUT FILE NAME	EMPTY CELLS	FILL FACT.	SOLUTIONS	PAR. TIME	SEQ. TIME	SPEEDUP
test1_a	53	34%	1	143	5	0.03
test1_b	59	27%	4715	323	356	1.10
test1_c	61	24%	132271	3891	8390	2.15
test1_d	62	23%	587264	26948	54115	2.00
test1_e	63	22%	3151964	116858	285944	2.44
test2_a	58	28%	1	196	74	0.37
test2_b	60	25%	276	323	130	0.40
test2_c	62	23%	32128	2852	6540	2.29
test2_d	64	20%	1014785	118315	248193	2.09
test2_e	65	19%	7388360	218643	538311	2.46
game0	5	93%	1	8	1	0
game1	45	44%	1	96	1	0
game2	49	39%	1	93	3	0.03
game3	59	27%	50142	569	1038	1.8

CONSIDERAZIONI FINALI

Come si può evincere dai dati ottenuti dagli esperimenti, lo speedup non è sempre maggiore di 1.

La ragione si comprende paragonando i casi in cui lo speedup è maggiore con quelli in cui è minore. Maggiore è lo spazio di ricerca delle soluzioni, più risulta efficace calcolare le soluzioni in parallelo. Nel caso in cui invece lo spazio di ricerca è piccolo, è più veloce calcolare sequenzialmente le soluzioni.

Le istanze che richiedono più tempo sono generalmente quelle con un fattore di riempimento basso, e di conseguenza con un maggior numero di celle libere.

Esiste quindi una correlazione tra fattore di riempimento, spazio delle soluzioni e tempo di esecuzione: minore è il fattore di riempimento, maggiore è lo spazio delle soluzioni, e di conseguenza il tempo di esecuzione, e migliore è quindi la performance dell'esecuzione parallela rispetto a quella sequenziale, e viceversa nel caso opposto.

ISTRUZIONI PER ESECUZIONE DA TERMINALE

cd DIRECTORY/FILE/SRC

javac Main.java; java Main PATH/TO/TEXTFILE.txt

Esempio di esecuzione:

```
MacBook-Pro-di-Ridam:~ Ridam$ cd Docu*/work*/Sudo*/src
MacBook-Pro-di-Ridam:src Ridam$ javac Main.java; java Main /Users/Ridam/Documents/workspace/Sudoku/debugInstances/game3.txt
empty cells: 59
fill factor: 27%
search space before elimination: 230443315200000000000000000000000000000 branches

solving in parallel...
done in: 569.0 ms
solutions: 50142

solving sequentially...
done in: 1228.0 ms
solutions: 50142

speedup: 2.15817223198594
MacBook-Pro-di-Ridam:src Ridam$
```

P.S. Per eseguire correttamente il programma occorre che il pacchetto Sudoku sia nel workspace di Java