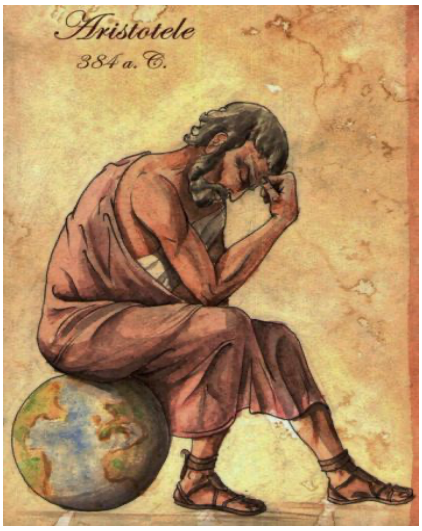
Aristotle's Number Puzzle

Progetto per l'esame di Parallel Algorithms Prof. Massimo Cafaro



Panico Andrea

A.A 2021/2022

Sommario

Aristotle's Number Puzzle

1. Il problema

1.1 L'esagono magico di ordine 3.

L'esagono magico di Aristotele è un rompicapo rientrante nella categoria dei puzzle. Si hanno a disposizione i primi diciannove numeri naturali {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19} e bisogna usarli per riempire le diciannove caselle vuote del seguente esagono:

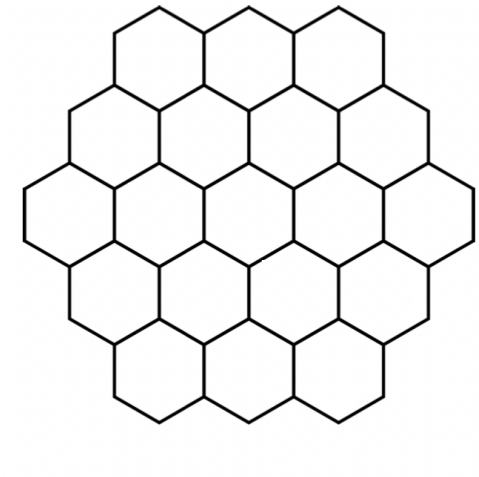
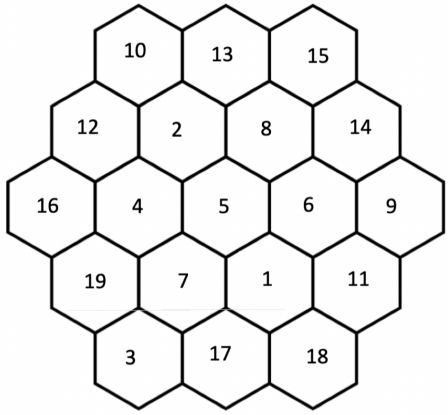


Figura 1 esagono vuoto



esempio di soluzione

Regole

1. Ogni numero deve essere impiegato una ed una sola volta.
2. $\forall\, riga\ j\ \in \{1,5\} \rightarrow \sum_{i=1}^{l} a_i = 38$. (38 viene chiamata costante magica).
   Ovvero, per ognuna delle 5 righe orizzontali, la sommatoria di tutti gli elementi a_i della riga deve essere pari a 38, con i che varia dalla posizione 1 alla posizione l (dove l è il numero di elementi per riga).
3. La sommatoria di tutte le diagonali destre deve essere pari a 38.
4. La sommatoria di tutte le diagonali sinistre deve essere pari a 38.

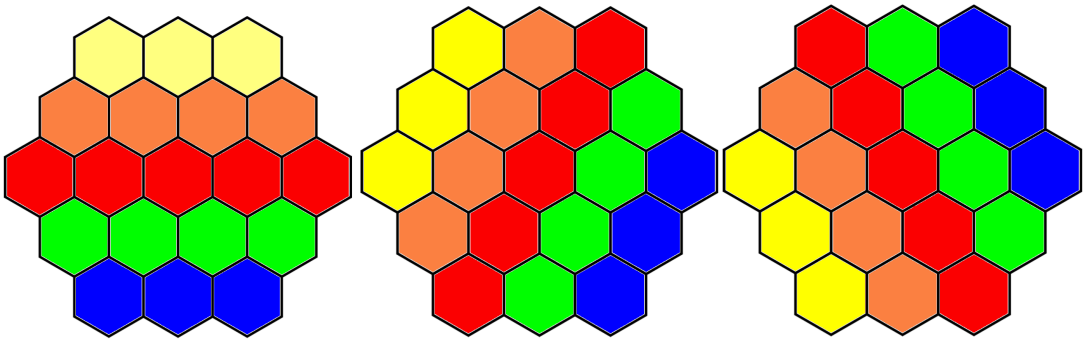


Figura 2 *Regola #2* *Regola #3* *Regola #4*

La soluzione non è unica.

1.2 Il problema dell'esagono magico in generale

Un esagono magico di ordine n è una disposizione di numeri tra loro distinti in una tabella esagonale composta da n celle per ogni lato dell'esagono, in modo che la somma dei numeri di ogni riga, di ogni diagonale sinistra e ogni diagonale destra abbia come somma la stessa costante magica. Un esagono magico **normale** ha il vincolo ulteriore di usare gli interi consecutivi da 1 a $3n^2 - 3n + 1$. In letteratura esiste la dimostrazione che esistono esagoni magici normali solo per $n = 1$ (caso banale) ed $n = 3$.[1]
Per il caso n = 3 gli interi da usare sono nel range [1, $(3(3^2) - 9 + 1)$] = [1,19]
Per il caso n = 1 gli interi da usare sono nel range [1, (1-1+1)] = {1}
Il caso in esame nel progetto viene chiamato esagono magico di ordine 3 poiché tutti i lati dell'esagono contano 3 caselle.

2. Tecniche risolutive e algoritmo sequenziale

2.0 Approccio #0 Naive

La prima idea è ovviamente l'approccio brute force che consisterebbe nell'andare a provare tutte le permutazioni dei 19 numeri fino all'ottenimento di una soluzione.

```
Algoritmo naïve Approccio 0
Input:
      A //lista di numeri naturali da 1 a 19
      n //n = 19
for i = 1 to 19!
      x = generated permutation of A;
      if(x respects all the constraints)
            return true;
      endif
endfor
```

Il numero di permutazioni è banalmente ricavabile ed è:
#permutazioni = $19! = 121645100408832000 = 1{,}21645100408832 \cdot 10^{17}$
Anche con il migliore dei processori, per il quale il tempo necessario a generare la permutazione e verificare la validità delle regole sia nell'ordine dei µs, occorrerebbero comunque $1{,}21645100408832 \cdot 10^{11} s$ per verificare tutte le soluzioni ($\cong 3800\ anni$). Per ovvie ragioni questo approccio è totalmente sbagliato; l'algoritmo avrebbe complessità computazionale esponenziale al crescere dell'input.

[1] https://it.wikipedia.org/wiki/Esagono_magico

$$T(n) = O(n!) = O(n^n) \quad usando\ l'approssimazione\ di\ Striling$$

Come visto numericamente, già per un n piuttosto piccolo come 19, il tempo necessario al calcolo è drammaticamente alto.
Prima di provare qualche approccio risolutivo, si cerca preliminarmente di matematizzare il problema.

2.1 Modello matematico

Le regole 2, 3 e 4 di 1.2 ci definiscono chiaramente delle equazioni dove le incognite sono le 19 celle dell'esagono. Si procede dunque a chiamare ognuna delle celle con una lettera dell'alfabeto creando la seguente associazione:

Figura 3 Assegnazione variabili

Le precedenti regole equivalgono alle equazioni:

$$\begin{aligned} a + b + c &= 38 \\ b + e + i + m &= 38 \\ h + i + j + k + l &= 38 \\ m + n + o + p &= 38 \\ q + r + s &= 38 \end{aligned}$$

$$\begin{aligned} a + d + h &= 38 \\ b + e + i + m &= 38 \\ c + f + j + n + q &= 38 \\ g + k + o + r &= 38 \\ l + p + s &= 38 \end{aligned}$$

$$\begin{aligned} c + g + l &= 38 \\ b + f + k + p &= 38 \\ a + e + j + o + s &= 38 \\ d + i + n + r &= 38 \\ h + m + q &= 38 \end{aligned}$$

La situazione attuale presenta un sistema di 15 equazioni e 19 incognite.
Le 15 equazioni non sono però linearmente indipendenti, ma utilizzando l'eliminazione di Gauss-Jordan si raggiunge un sistema di 12 equazioni linearmente indipendenti e 19 incognite.

```
a = 76 - j - k - n - 2o - p - r - s
b = j + n + o
c = -38 + k + o + p + r + s
d = j + k + o
e = -38 + k + n + o + p + r
f = 38 - j - k - n - o - p
g = 38 - k - o - r
h = -38 + n + o + p + r + s
i = 38 - j - k - n - o - r
l = 38 - p - s
m = 38 - n - o - p
q = 38 - r - s
```

Per cui il numero di incognite che rimangono parametriche è pari a 12 e il numero di incognite da assegnare arbitrariamente è:
incognite - # equazioni linearmente indipendenti = 19 -12 = 7.
In figura 4 vengono evidenziate le incognite che dovranno essere assegnate arbitrariamente con tecnica brute force.

Figura 4 Variabili decisionali

2.2 Algoritmo sequenziale

La situazione che ha scaturito l'elaborazione matematica permette di abbattere il numero di possibili permutazioni generate per completare l'esagono.
Il numero di permutazioni si ottiene generando tutte le possibili disposizioni di 7 elementi distinti estratti dall'insieme originario (i numeri da 1 a 19).
È stato usato il termine disposizione con cognizione di causa, in quanto è necessario estrarre k elementi da un insieme di n elementi tenendo conto dell'ordine.

$$\#disposizioni = \frac{n!}{(n-k)!} = \frac{19!}{12!} = 25395520$$

Un numero decisamente inferiore al caso naive, ma comunque esponenziale rispetto all'input. In questo caso la dipendenza non è rispetto al numero di elementi di partenza (n, nel caso studiato n =19), ma:

$$T(n,k) = O\left(\frac{n!}{(n-k)!}\right) = O\left(\frac{n^n}{(n-k)^{n-k}}\right).$$

La complessità computazionale è influenzata da due elementi di input: k ed n.
Questa equazione ci fa apprezzare il motivo per cui questo problema, così come molti rientranti nella categoria dei puzzle, sono teoricamente risolvibili solo in tempo esponenziale, ma lo sono in funzione della dimensione dell'input. Per cui, per input piccoli (come l'n = 19 analizzato) il problema può essere considerato risolvibile in tempo costante: $T(n,k) = O(1)$.
Questo risultato è ottenibile matematicamente prendendo un k molto piccolo (ovvero la differenza tra incognite ed equazioni linearmente indipendenti).

Algoritmo Sequenziale

```
Input:
      A //lista di numeri naturali da 1 a 19
      n //n = 19

matrix = generate initial matrix 15*20;
Gauss-Jordan(matrix);
n_dispositions = 19!/12!;
for i = 1 to n_dispositions
      disp = generate_dispositions(A,7) //7 elements from original list A
      for j = 1 to 7
            x[j] = disp[j];
            add x[j] to matrix;
      endfor
      solution = back_substitution(matrix); //result is an array
      result = validation(result);
      if (result)
            return solution;
      endif
endfor
```

L'analisi precedente tiene conto solo della complessità introdotta dalla generazione e dall'analisi delle disposizioni poiché questa è la parte che richiede più risorse di calcolo, tuttavia non è l'unica; si esplicita il flusso logico dell'algoritmo:

1. A partire dalle equazioni iniziali, si scrive il sistema in forma matriciale;
2. Si Applica Gauss-Jordan al sistema;
3. Si genera una disposizione di 7 elementi;
4. Questi 7 elementi sono 7 soluzioni del sistema e si vanno ad aggiungere alle altre 12 equazioni linearmente indipendenti;
5. Si risolve il sistema con il metodo di back-substitution;
6. Si estrae dalla matrice rappresentante il sistema la colonna dei termini noti e si verificano i vincoli di dominio (ogni valore deve essere unico è compreso tra 1 e 19.

2.3 Miglioramenti

1. Figura 4 evidenzia quelle che sono le variabili a cui si assegnano i valori arbitrariamente per poter generare una soluzione completa. Si noti che si può fare pruning di alcuni casi che non soddisfano i vincoli in partenza:
   - La somma della riga n-o-p potrebbe essere > 38;
   - La somma della diagonale sinistra r-o-k potrebbe essere > 38
   - La somma della diagonale destra j-o-s potrebbe essere > 38

   Basta violare uno di questi 3 vincoli per considerare la possibile soluzione non valida e passare alla prossima. Non vengono fatti controlli laddove vi sono soltanto due elementi in quanto prendendo i due elementi massimi dell'insieme (19 e 18), vi è la possibilità di ottenere una combinazione con un terzo elemento che mi permette la validità: 19 – 18 – 1 (anche in ordine diverso).

2. La funzione back-substitution produce come soluzione un vettore completo delle soluzioni per le 19 incognite. Si può evitare di fare il controllo di validazione a valle per tutte quelle disposizioni che mi generano dei termini noti > 38 già nelle iterazioni della back-substitution e interrompere invece immediatamente la generazione della soluzione ritenendola non valida.
   Esempio:
   si prenda la disposizione che mi associa all'insieme [j, k, n, o, p, r, s] i valori [1, 2, 15, 16, 17, 18, 19].

```
a = 76 - j - k - n - 2o - p - r - s
b = j + n + o
c = -38 + k + o + p + r + s
d = j + k + o
e = -38 + k + n + o + p + r
f = 38 - j - k - n - o - p
g = 38 - k - o - r
h = -38 + n + o + p + r + s
i = 38 - j - k - n - o - r
l = 38 - p - s
m = 38 - n - o - p
q = 38 - 18 - 19  -> q = 1
```

```
a = 76 - j - k - n - 2o - p - r - s
b = j + n + o
c = -38 + k + o + p + r + s
d = j + k + o
e = -38 + k + n + o + p + r
f = 38 - j - k - n - o - p
g = 38 - k - o - r
h = -38 + 15 + 16 + 17 + 19 + 18 -> h = 47 > 38 ! violazione
i = 38 - j - k - n - o - r
l = 38 - p - s
m = 38 - n - o - p
q = 1
```

3. Nel momento in cui la richiesta del problema non fosse trovare **una** soluzione, ma tutte le possibili soluzioni, allora può essere apportato un accorgimento sull'approccio adottato per trovare tutte le soluzioni a partire da una, abbattendo così i tempi di ricerca della versione di base che analizzerebbe tutto il dominio delle disposizioni.

Il problema ha un'evidente simmetria, e conta dodici possibili soluzioni, le quali però sono tutte riflessioni o rotazioni di una soluzione di partenza. In particolare, ci sono sei rotazioni (una per ogni lato dell'esagono) e per ognuna la rispettiva riflessione.

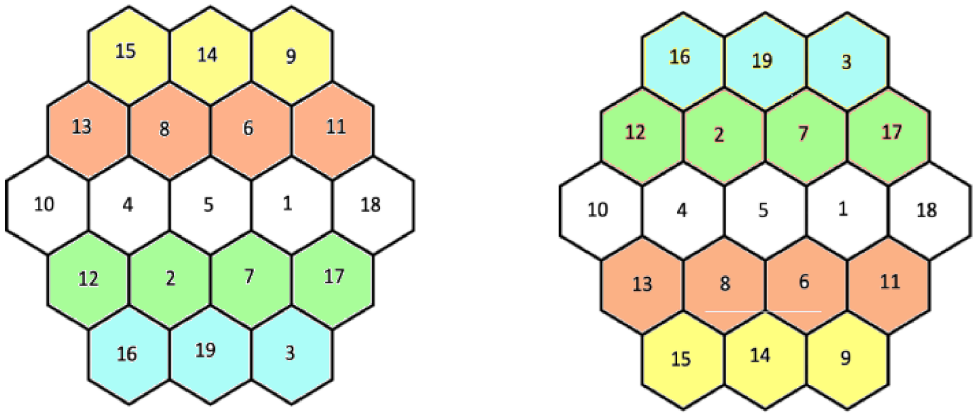


Figura 5 Possibile prima soluzione trovata e sua riflessa (anch'essa soluzione valida)

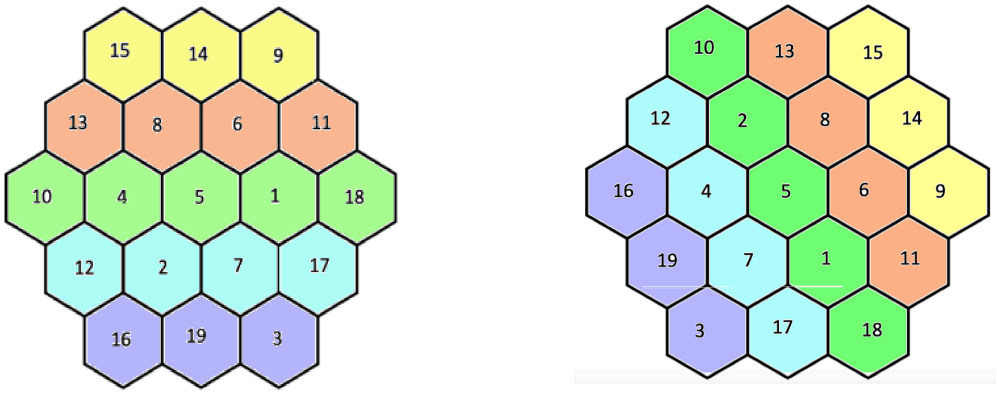


Figura 6 Possibile soluzione iniziale e sua rotazione (anch'essa soluzione valida)

Per effettuare una rotazione la strategia implementativa adottata consiste nel cambiamento della rappresentazione da un esagono ad un array 1-D di 19 elementi contente la successione degli elementi nell'ordine evidenziato in figura. (n.b. i numeri all'interno delle celle non sono i valori, ma gli indici).

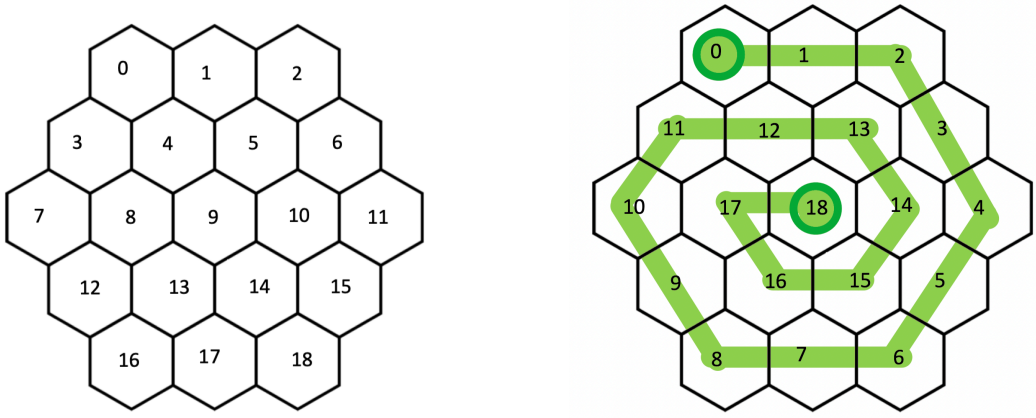


Figura 7 "Snake" representation change

3. Progettazione parallela

3.1 Progettazione

Partendo dalla soluzione sequenziale proposta nella sezione 2 si vuole parallelizzare l'algoritmo utilizzando la libreria MPI.
Il lavoro principale è la divisione delle disposizioni da generare e validare; quindi, anche se l'algoritmo consta di altre parti, queste impiegano una percentuale bassissima del tempo di calcolo ed è inoltre preferibile replicare il calcolo piuttosto che comunicare i dati.

3.1.1 Partitioning

Domain decomposition:
viene associato un task primitivo ad ogni disposizione per ottenere la grana quanto più fine possibile. Il lavoro associato ad ogni task potrebbe essere diverso causa violazione di qualche vincolo e troncamento dell'iterazione. Per questo motivo si decide di assegnare in modo round-robin le disposizioni ai task per bilanciare il carico di lavoro.

3.1.2 Communication

Nel momento in cui un processo trova una soluzione deve comunicarlo a tutti gli altri processi, i quali devono terminare a loro volta la ricerca.
Per comunicare la fine della ricerca si opta per cui per un sistema di comunicazioni di tipo ring-network avviate dal processo *id* che per primo trova una soluzione.
Si indichi con p il numero di processi.
Il processo *id* invia con la funzione MPI_Send il token che contiene un solo valore (l'intero del processo che ha trovato la soluzione e quindi *id* stesso) al successore nella ring-network.
All'inizio di ogni iterazione ogni processo verifica se ci sono messaggi in entrata con una funzione non bloccante: la MPI_Iprobe; nel caso ci siano, riceve il messaggio che contiene *id* e lo invia al suo successore nella ring network.
Nel momento in cui il messaggio arriva al processo $(id - 1)\%p$ (il predecessore di *id* nella ring-network) la comunicazione si interrompe poiché tutti i processi hanno interrotto la loro ricerca. Intanto *id* stampa a video le soluzioni per poi terminare.

Quindi le comunicazioni sono:

- Local (point-to-point)
- Structured (struttura regolare del messaggio: il token)

Nell'implementazione dell'algoritmo è emerso che l'utilizzo della funzione MPI_Iprobe, seppur non bloccante, andava ad aumentare il tempo di esecuzione del programma causa il controllo fatto all'inizio di ogni iterazione.
Si è cercato un valore soglia *threshold* che minimizzasse il running time del programma effettuando l'Iprobe ogni *threshold* iterazioni:
con $threshold = 500$ il running time peggiora dello 0,1% (con $threshold = 50$ *il running time peggiora del 3%*).

3.2.3 Mapping

Al termine delle considerazioni fatte per partitioning e communication si è seguita la via di assegnare ciclicamente le attività ai processori seguendo il mapping decision tree nel caso di numero statico di tasks, comunicazioni strutturate e tempo di computation variabile per ogni task.

3.2 Algoritmo parallelo

```
Algoritmo Parallelo
Input:
      A //lista di numeri naturali da 1 a 19
      n //n = 19

matrix = generate initial matrix 15*20;
Gauss-Jordan(matrix);
n_dispositions = 19!/12!;

founded = 0;

iterations = 0;
analyzed = - p + id;

while (!founded)
      if(iterations == analyzed + p)
            analyzed = iterations;
            flag = 0;
            if(!iterations % 500)
                  MPI_Iprobe(previous, 0, MPI_COMM_WORLD, &flag, &status);
                  if (flag)
                        MPI_Recv(&founder,   1,   ;MPI_INT,   previous,   0,
                        MPI_COMM_WORLD);
                        Break;
                  endif
            endif;

            disp = generate_dispositions(A,7)
            for j = 1 to 7
                  x[j] = disp[j];
                  add x[j] to matrix;
            endfor
            solution = back_substitution(matrix); //result is an array
            result = validation(result);
            if (result)
                  founded = 1;
                  MPI_Send(&id,1,MPI_INT, next, 0 ,MPI_COMM_WORLD)
                  print_solutions();
            endif
```

```
        endif
        iterations ++

endwhile
```

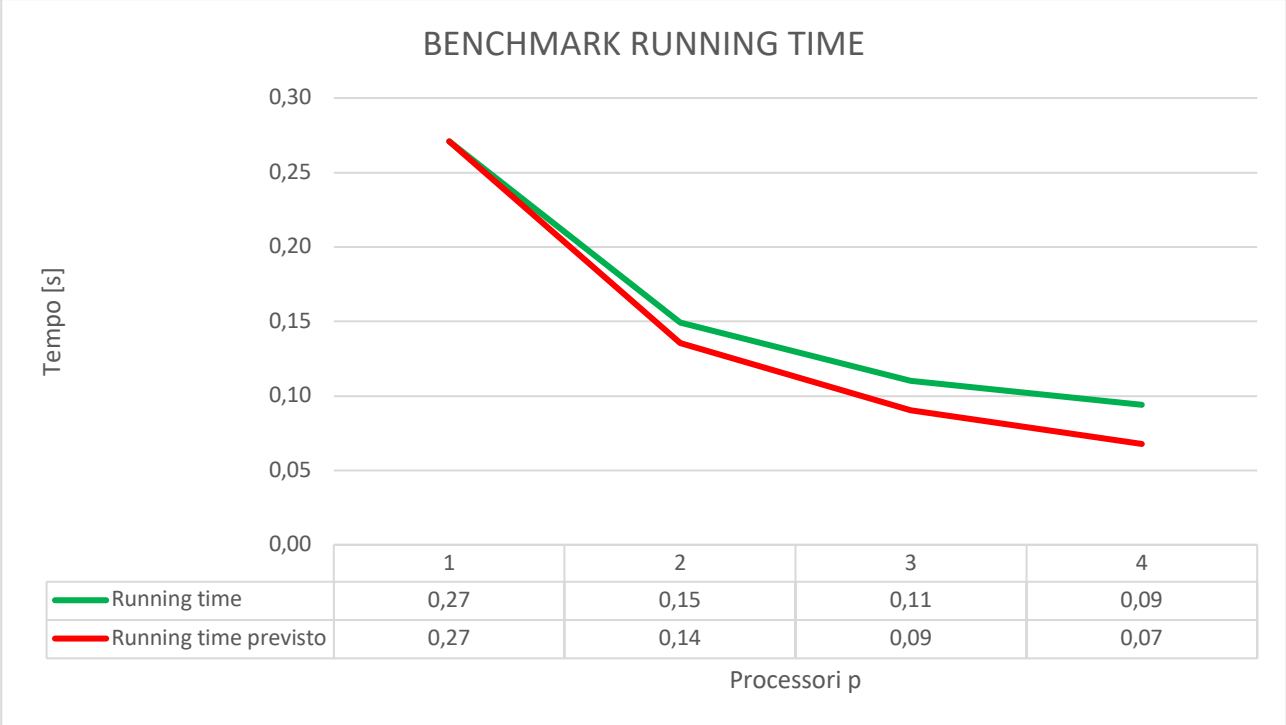
4. Benchmark

4.1 Running Time

Test effettuati su processore Apple M1 chip:

- 8-core CPU con 4 performance cores e 4 efficiency cores
- Frequenza di clock: tra i 0,6 ed i 3,204 GHz
- RAM 4,266 MT/s LPDDR4X SDRAM

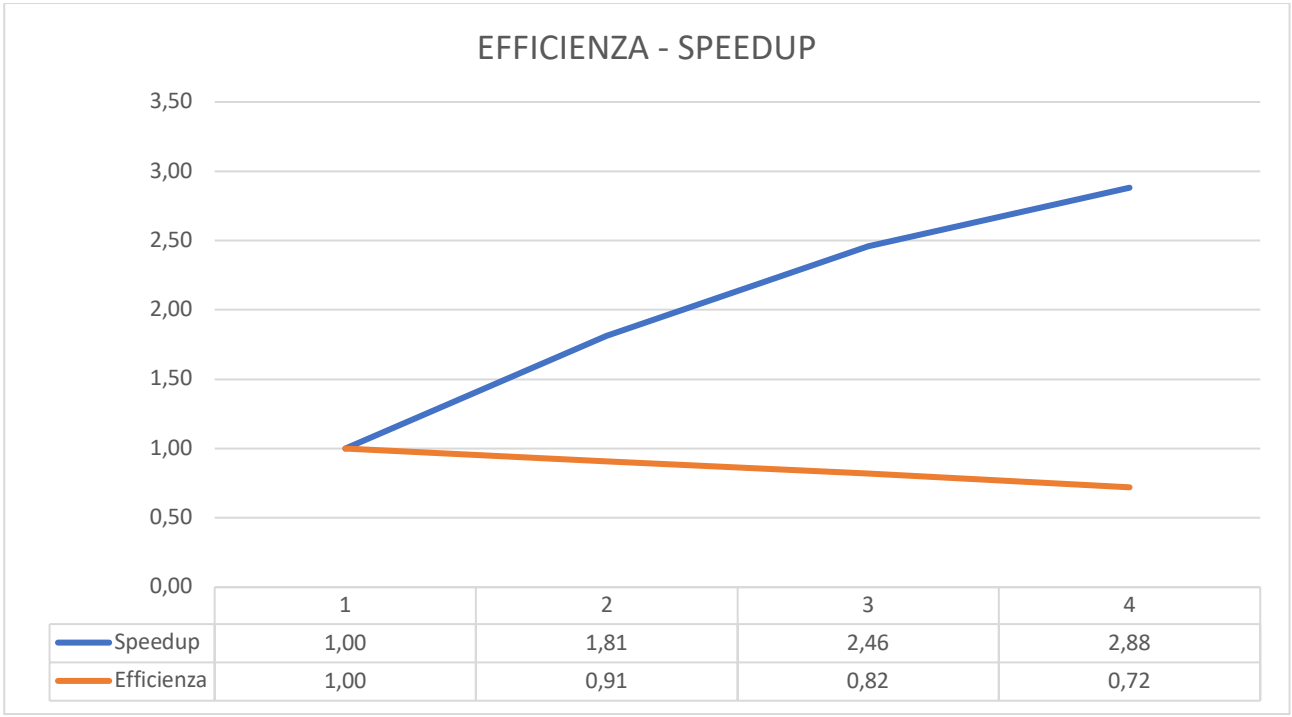
Per ogni analisi effettuata su un numero definito di processori, il valore è stato mediato su 1000 test.



	1	2	3	4
Running time	0,27	0,15	0,11	0,09
Running time previsto	0,27	0,14	0,09	0,07

Non ha senso andare ad esplicitare la forma del modello di calcolo, in quanto la dimensione m, intesa come iterazioni del main loop, sarebbe il numero di disposizioni. L' input n è funzione esponenziale della grandezza m come esposto nella sezione 2.2.
Si fornisce un confronto tra il tempo stimato $\left(\frac{tempo\ sequenziale}{n°\ di\ processori}\right)$ e il tempo misurato sulla macchina con chip M1.
Prima di tutto è stato eseguito il programma su un solo processore e poi, per fornire una stima del tempo di esecuzione, è stato diviso di volta in volta per il numero di processori.
Questa "approssimazione" è giustificata dal fatto che il tempo di calcolo (che, come parte dominante, deve risolvere un sistema con back-substitution) sovrasta il tempo delle comunicazioni (un controllo di tempo costante per ogni disposizione).

4.2 Efficienza e Speedup



	1	2	3	4
Speedup	1,00	1,81	2,46	2,88
Efficienza	1,00	0,91	0,82	0,72

Sitografia

[1]https://it.wikipedia.org/wiki/Esagono_magico pagina

[2]https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784394004/1/ch01lvl1sec08/aristotle-s-number-puzzle

[3] http://hwiechers.blogspot.com/2013/03/solving-artitotles-number-puzzle.html

[4] https://jtp.io/2017/01/12/aristotle-number-puzzle.html

Per tutti i riferimenti vale la data di consultazione: 01/02/2022