MANDELBROT SET

Riccardo Fontanini

Consegna

Write a "C" program determining the Mandelbrot set.

The program is going act as baseline version.

All subsequent algorithmic improvements will be compared to its figures.

The program must be coded, tested, profiled in terms of execution time, and a brief report - yet exhaustive - will illustrate all aspects of the derivation, will present the adopted coding solutions and choices, and will show the obtained result to demonstrate the correctness of the code.

Mandelbrot set

È l'insieme dei numeri complessi c per i quali la successione definita da:

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

è limitata.

Si nota che il contorno della figura complessa espressa dall'insieme di Mandelbrot è frattale

Notazione

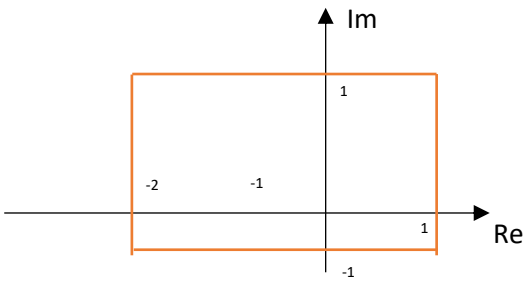
Nella seguente relazione saranno presenti riferimenti a due concetti chiave per il Mandelbrot set:

Numero massimo di iterazioni per punto: cioè il massimo numero di iterazioni che l'algoritmo deve eseguire per definire che un determinato punto sul piano complesso appartenga o meno al Mandelbrot set.

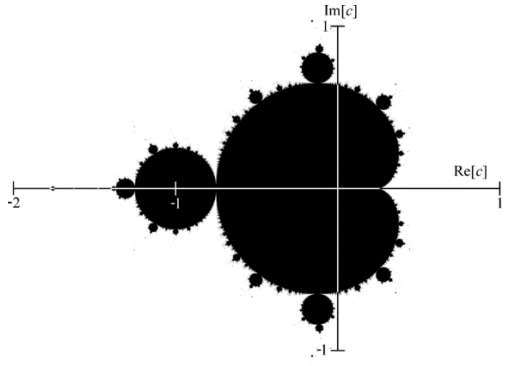
Risoluzione: di seguito ci si riferisce al concetto di risoluzione come al numero di punti che il programma deve elaborare tra un intero e l'altro del piano complesso di riferimento. Per esempio: se ci si riferisce ad una risoluzione di 2500 punti, vuol dire che sull'asse reale del piano complesso, tra il punto $0 + i0$ e il punto $1 + i0$ verranno elaborati 2500 punti, quindi il passo tra un punto e il successivo sarà di $\frac{1}{2500} = 0.0004$.

Immagine e mapping su piano complesso

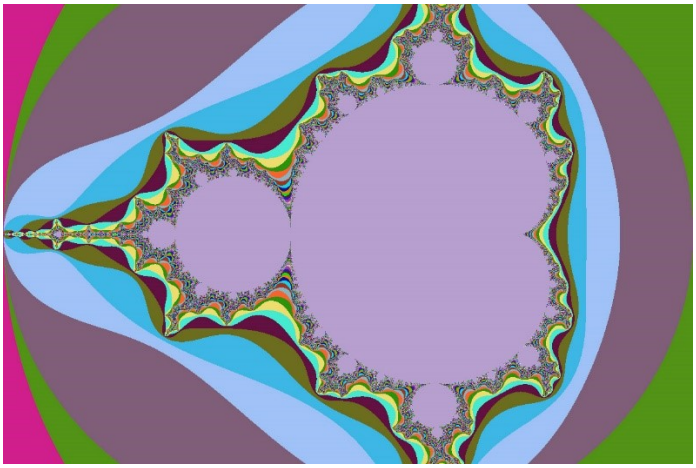
L'algoritmo è stato creato con posizionamento statico sul piano complesso con limiti sull'asse reale da $-2 + i0$ a $1 + i0$ mentre sull'asse complesso da $1i$ a $-1i$.



Impostando una risoluzione posso determinare il passo all'interno del piano complesso e quindi mappare il Mandelbrot set all'interno dei limiti stabiliti.



Da cui si possono ottenere la larghezza e l'altezza dell'immagine che verrà generata moltiplicando la risoluzione per le ampiezze del campo complesso: cioè $larghezza = risoluzione \times 3$ e $altezza = risoluzione \times 2$.

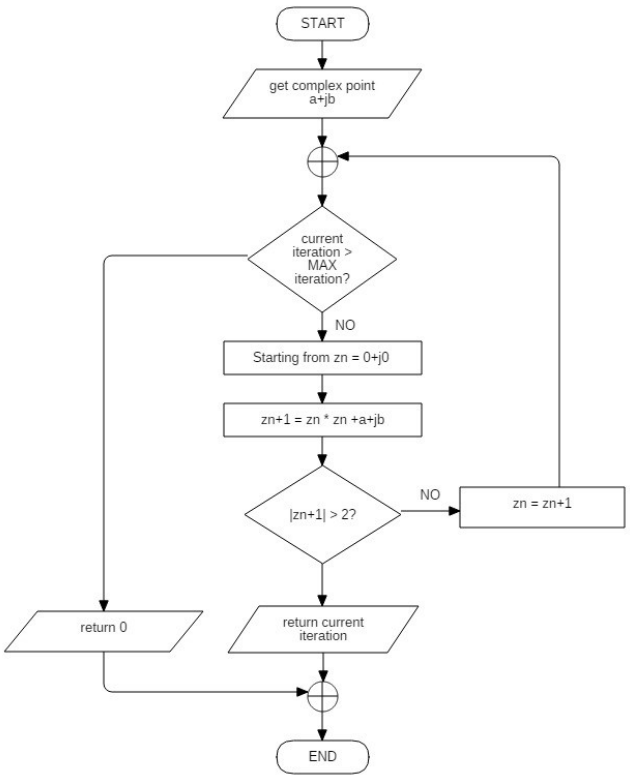


Sistema per il test

I test sono stati eseguiti, per la parte di elaborazione su CPU su un Intel Core i7-3630QM e per il confronto in GPU su una macchina server che sfrutta una scheda grafica NVIDIA K40.

Codice e algoritmi

Il codice per l'implementazione del Mandelbrot set è il seguente:



Identificato un punto sul piano complesso, si entra in un ciclo che itera fino a quando il numero massimo di iterazioni è raggiunto o $|z_{n+1}| > 2$. Nel primo caso il punto in questione appartiene al Mandelbrot set, altrimenti non appartiene. Successivamente ad ogni punto associo, in accordo con una prefissata palette, un determinato colore in base all'iterazione raggiunta nel ciclo.

Compilazione

Per la compilazione (in sistemi linux) è stato creato un makefile, una volta invocato il comando:

make mandelbrot

permette la compilazione di tutte le librerie utilizzate e la creazione dell'eseguibile nella cartella:

./build/linux

La compilazione avviene sfruttando lo standard C99.

Esecuzione e parametri

Il programma può essere eseguito su linea di comando semplicemente digitando:

./mandelbrot

Tale comando sfrutta i valori di configurazione di default e crea un file test.bmp nella cartella corrente.

I parametri del programma da passare in linea di comando sono:

- *-h* mostra la guida
- *-n <name>* modifica il nome del file in uscita (standard: ./test.bmp)
- *-r <resolution>* modifica la risoluzione del file in uscita (standard: 400)
- *-i <maxiteration>* modifica il numero massimo (standard: 100)

Esempio

./mandelbrot -n nuovonome -r 500

Creerà un file nuovonome.ppm con dimensione 1500x1000 contenente la visualizzazione del Mandelbrot set.

Si ricorda che se in fase di compilazione è stata abilitata la macro DEBUG verrà visualizzato anche il debug direttamente in console.

Per eseguire delle simulazioni ricorrenti, cioè poter invocare il programma ciclicamente per generare dei file di log e quindi poterli analizzare successivamente, si è utilizzato uno script python presente nella cartella *"<root>/build/linux/iterate.py"*.

Problematiche

Heap & Stack

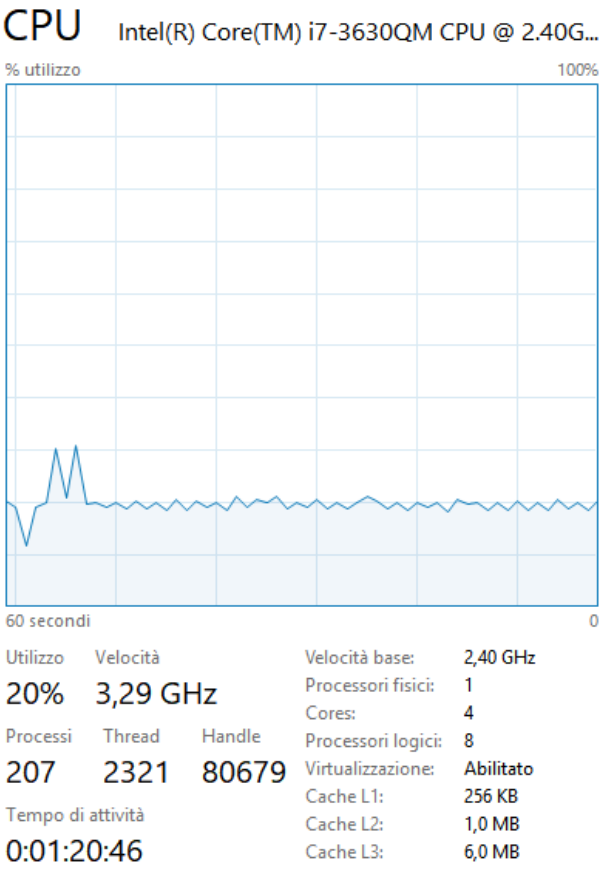
Durante lo sviluppo del programma, si è verificato un problema di allocazione di memoria nello stack assegnato al programma: in particolare non poteva essere allocata una quantità troppo elevata di risorse nello stack (in questo caso veniva generato un errore durante l'allocazione della matrice di interi nel main).

Tale problema è stato risolto introducendo l'allocazione dinamica nel flusso del programma, permettendo quindi al programma stesso di allocare la memoria all'interno dell'heap e liberando le risorse all'uscita dal programma.

Analisi del sistema

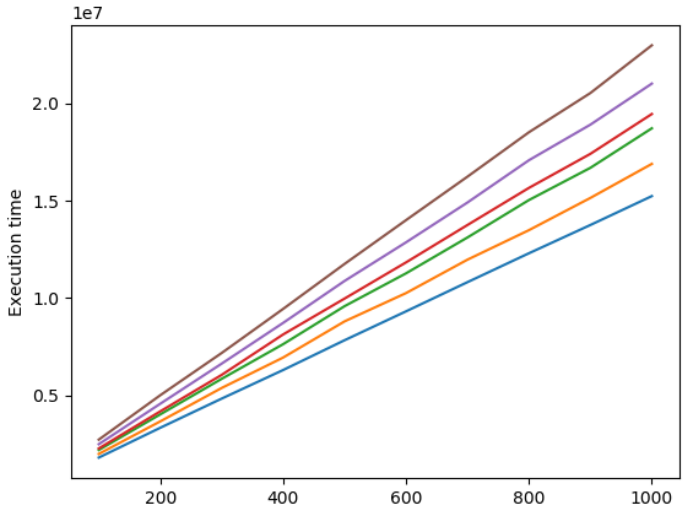
Single thread

L'analisi seguente consiste nel confronto dei tempi di elaborazione e salvataggio di un mandelbrot set lanciato su un thread singolo. Grazie ad un semplice script python, che esegue in successione il programma C per la creazione del mandelbrot set, modificando ogni volta i parametri (risoluzione e numero massimo di iterazioni), si nota subito che la CPU viene sfruttata non a pieno delle sue capacità. L'immagine successiva mostra l'utilizzo della CPU durante l'elaborazione del set, come si vede solamente un core logico del processore è impegnato.



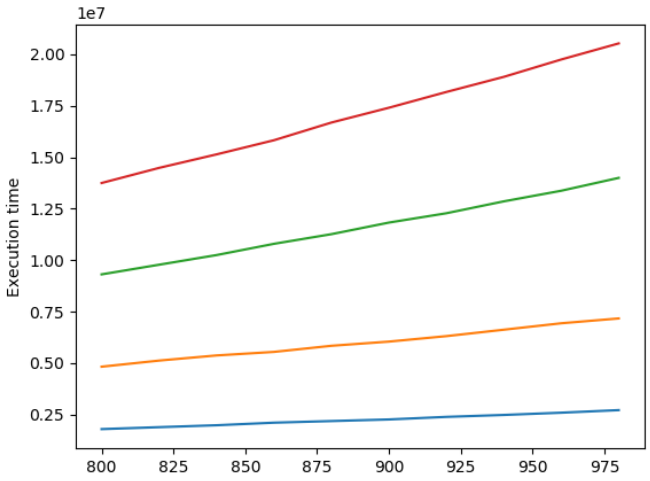
Se andiamo ad analizzare più nello specifico i tempi possiamo notare come al crescere della risoluzione e del numero di iterazioni massime i tempi di elaborazione del mandelbrot set si alzino. Rimangano pressoché invariati quelli di salvataggio. Prendiamo in considerazione il grafico seguente: troviamo rappresentati il tempo di elaborazione in relazione al numero di iterazioni massime per punto, analizzato per differenti risoluzioni.

Si nota che fornendo delle variazioni costanti del numero massimo di iterazioni, otteniamo delle leggi pressoché lineari, che ci suggeriscono una capacità di elaborazione del mandelbrot set linearmente dipendente dal tempo di esecuzione (a parità di risoluzione).



Tempi espressi in microsecondi

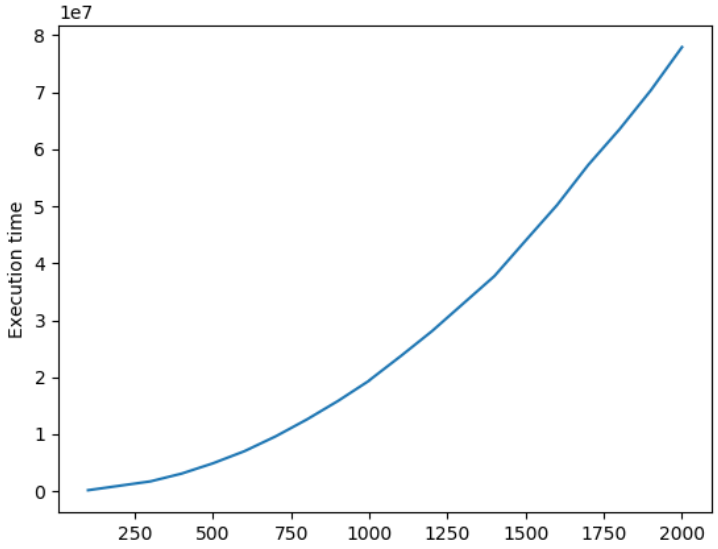
Si nota che tutte hanno un andamento lineare, ma con coefficienti di proporzionalità differenti l'una dall'altra. Se invece si fa tendere a 0 il numero massimo di iterazioni (estrapolazione), si nota una convergenza del fascio di caratteristiche. Possiamo interpretare tale dato come il tempo di overhead necessario alla strutturazione del problema, non prettamente coinvolto nella risoluzione del mandelbrot set. La traslazione di tale punto asintotico verso lo 0 fa sì che l'impatto dell'overhead sul tempo di elaborazione totale migliori: se confrontato in termini assoluti con le varie caratteristiche, ha un impatto maggiore per elaborazioni con un numero massimo di iterazioni basso, mentre per un numero massimo di iterazioni nell'ordine di qualche centinaio, risulta pressoché ininfluente.



Tempi espressi in microsecondi

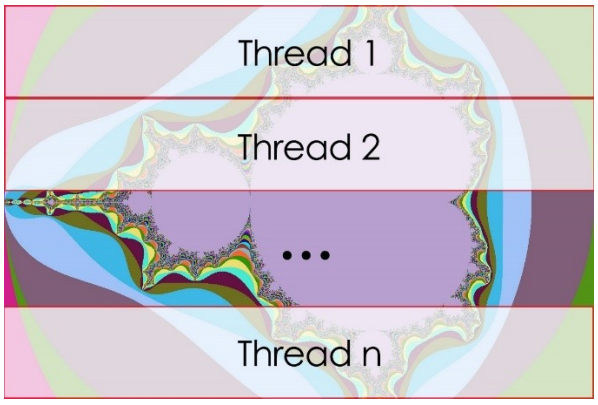
Tale figura mostra il tempo di esecuzione dell'algoritmo in relazione alla risoluzione delle immagini da generare per quattro differenti massime iterazioni per punto (100, 300, 600, 1000). Si nota come la caratteristica blu (massime iterazioni pari a 100) sia pressoché costante al variare della risoluzione impiegata. Mentre per la caratteristica rossa (massime iterazioni pari a 1000), oltre ad essere in valore assoluto molto più elevata rispetto alla blu, ha una variabilità molto maggiore, quindi non può essere ritenuta costate rispetto al parametro del numero massimo di iterazioni. Se estendiamo questo grafico per una sola caratteristica (con numero massimo di iterazioni fissa pari a 800 per punto) in un range di risoluzione che

varia da 100 a 2500 (più ampio delle precedenti), possiamo notare come la curva tenda ad impennarsi all'aumentare della risoluzione con un andamento quasi quadratico.

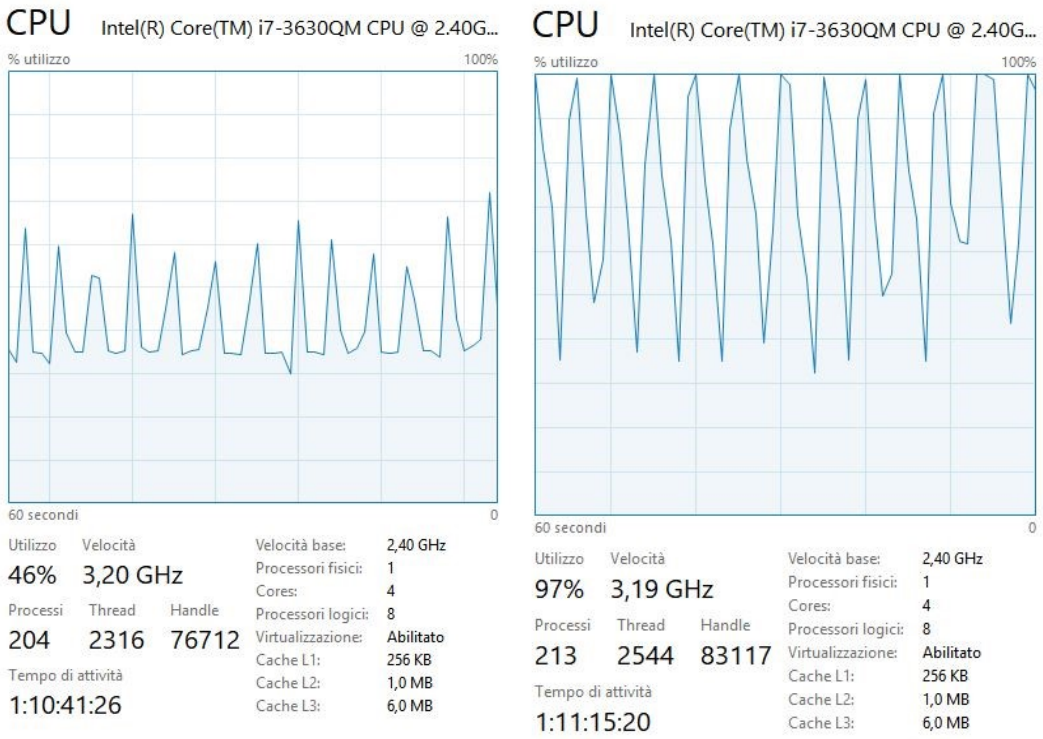


Multi-thread

Il mapping per suddividere il lavoro sugli n thread previsti dal programma, ha portato ad uno slicing orizzontale del piano complesso: cioè ogni thread si occupa di elaborare una fetta orizzontale della figura. Lo spazio è suddiviso in parti uguali, così da suddividere più omogeneamente possibile il carico di lavoro sui vari core del processore.

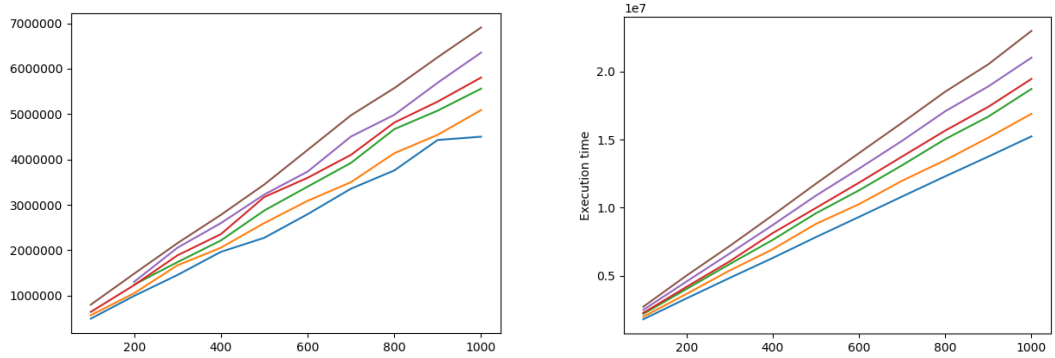


In prima analisi consideriamo l'utilizzo del processore: sfruttando una architettura Intel i7 quad core, tramite l'hyperthreading è possibile sfruttare fino a 4 core in parallelo, 8 core virtuali. I due grafici seguenti mostrano molto chiaramente l'elaborazione di 4 core a sinistra (che sfruttano circa il 50% delle risorse computazionali del processore) mentre gli 8 core virtuali a destra.

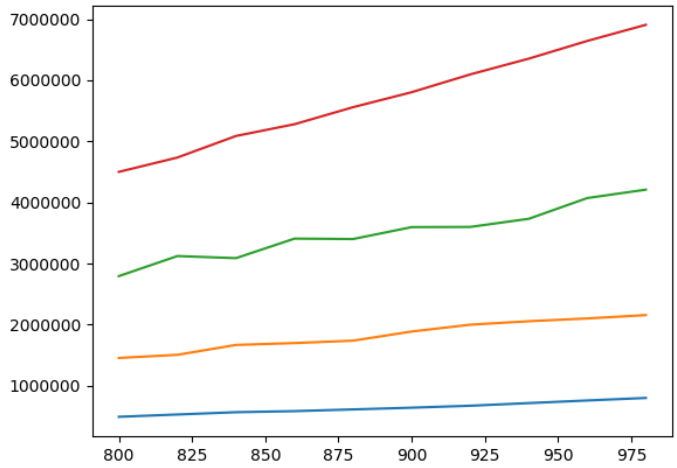


Entrambi i test sono stati eseguiti lanciando il programma per generare il Mandelbrot set in loop così da poter apprezzarne l'utilizzo del sistema.

Di seguito riportiamo l'andamento delle curve per la creazione del Mandelbrot set in due diversi casi: a sinistra vediamo la versione multi-thread mentre a destra l'elaborazione mono core. Ogni curva rappresenta l'elaborazione per una risoluzione diversa. Nonostante siano piuttosto simili nell'andamento, la versione multi-thread è scalata di circa un ordine di grandezza rispetto a quella che utilizza un solo thread.



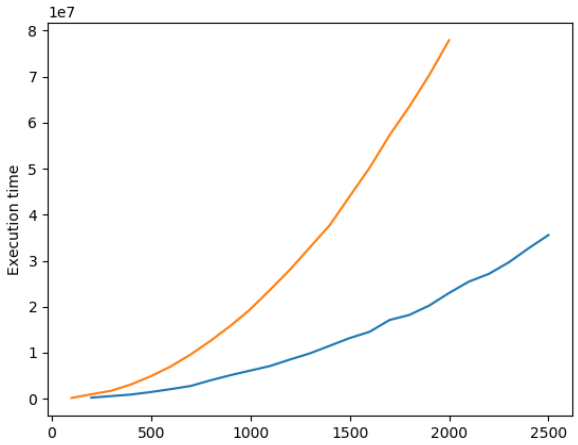
Il grafico seguente mostra l'andamento del tempo di elaborazione in relazione alla variazione della risoluzione dell'immagine. Ogni curva ha un numero massimo di cicli di elaborazioni per punto differente l'una dall'altra: rispettivamente 1000, 600, 300, 100. L'analisi di questi dati è consistente con quella della versione single-thread, quindi si invita il lettore a far riferimento a tale paragrafo nella relazione dedicata.



La figura seguente mostra un raffronto tra la versione single-thread e multi-thread del tempo di elaborazione al variare della risoluzione. Entrambe le elaborazioni sono state fatte con un numero massimo di iterazioni per punto di 800. Si nota subito che le due curve, per passo elevato, differiscono di molto ed il gap tende a crescere al incrementarsi della risoluzione. Citando la legge di Amdahl:

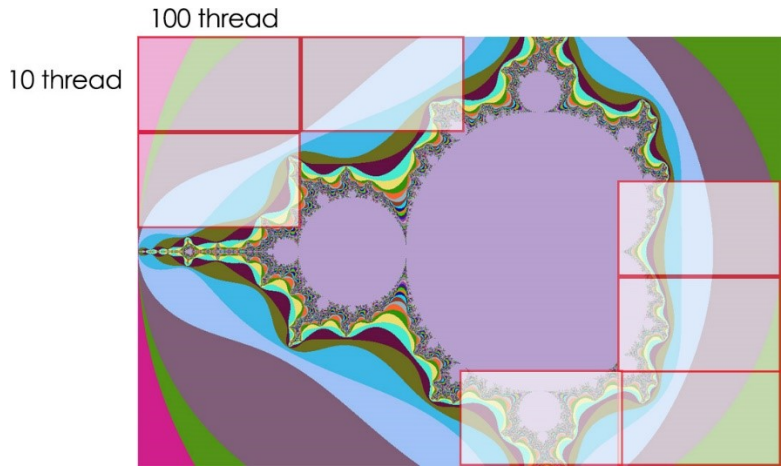
$$speedup = \frac{1}{S + \frac{1 - S}{P}}$$

che mette in relazione il miglioramento complessivo delle prestazioni di un sistema informatico in base ai miglioramenti delle singole parti del sistema, è possibile comprendere il perché dell'incremento prestazionale. Di fatto partizionando il lavoro su più core, eseguiamo un miglioramento prestazionale su una parte del sistema (la parte più onerosa), migliorando quindi le prestazioni totali del sistema.



Cuda

Per quanto riguarda la versione Cuda del Mandelbrot set, il carico di lavoro è stato mappato sulla griglia della GPU dividendo lo spazio in blocchi 100 x 10 thread. Ogni blocco elabora 1000 pixel completamente in parallelo l'uno rispetto all'altro. Quindi la griglia è formata da: $\frac{larghezza_{inPixel}}{100} \times \frac{altezza_{inPixel}}{10}$ blocchi.



Grazie al tool Nvprof è stato possibile ottenere il tempo effettivo di elaborazione del kernel

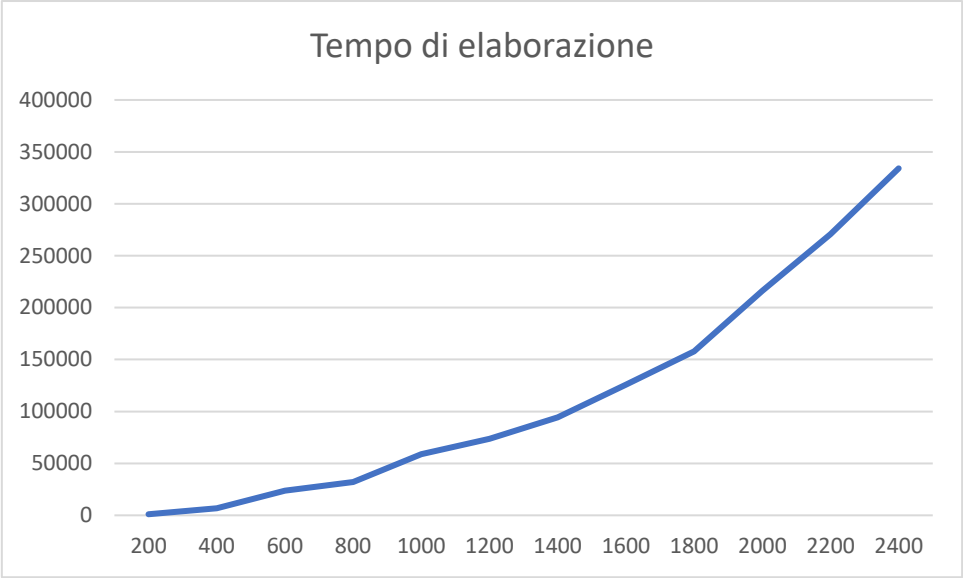
```
==23635== Profiling application: ./mandelbrot -i 800 -r 2400
==23635== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  334.10ms         1  334.10ms  334.10ms  334.10ms  cuda_elaboration(int*, int, int, int)
      API calls:   60.87%  524.24ms         1  524.24ms  524.24ms  524.24ms  cudaMallocManaged
                   38.88%  334.89ms         2  167.45ms  14.728us  334.88ms  cudaDeviceSynchronize
                    0.16%  1.4184ms       188  7.5440us     280ns  371.76us  cuDeviceGetAttribute
                    0.05%  402.61us         1  402.61us  402.61us  402.61us  cudaLaunch
                    0.02%  191.63us         2  95.814us  79.346us  112.28us  cuDeviceTotalMem
                    0.01%  115.27us         2  57.634us  55.036us  60.232us  cuDeviceGetName
                    0.00%  7.1010us         4  1.7750us     232ns  5.5520us  cudaSetupArgument
                    0.00%  3.4770us         3  1.1590us     330ns  2.4950us  cuDeviceGetCount
                    0.00%  3.2540us         4     813ns     348ns  1.4620us  cuDeviceGet
                    0.00%  1.8280us         1  1.8280us  1.8280us  1.8280us  cudaConfigureCall

==23635== Unified Memory profiling result:
```

I risultati per un set con 800 massime iterazioni sono:

Risoluzione	Tempo di elaborazione
200	1070
400	6790
600	23720
800	32080
1000	59060
1200	73560
1400	94230
1600	125840
1800	158000
2000	215990
2200	270590
2400	334100

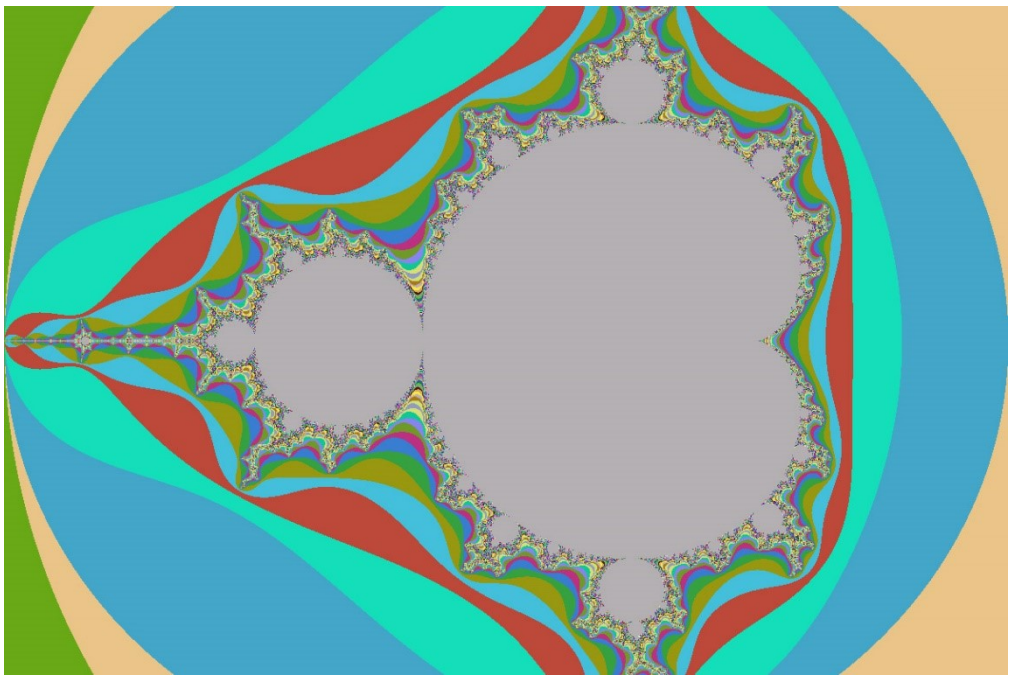
Il seguente grafico mostra l'andamento del tempo di elaborazione (in microsecondi) in relazione alla risoluzione.



Come si può vedere i tempi sono estremamente più rapidi in termini assoluti rispetto alle versioni single e multi thread. Per esempio, per una risoluzione pari a 1000 e massime iterazioni per punto pari a 800, il tempo di elaborazione dell'algoritmo single thread è di circa 20 secondi, quello multi thread (8 thread) di circa 3 secondi, mentre utilizzando la GPU il tempo di elaborazione scende a 0.05 s.

Risultati

Dalle precendenti analisi abbiamo visto come l'impiego del calcolo parallelo incrementi di molto le prestazioni precedenti per la definizione e la creazione del Mandelbrot set. Ogni volta che viene eseguito il programma, otteniamo un file dove è presente la rappresentazione grafica:



Sommario