Università degli Studi di Milano Bicocca

Dipartimento di Informatica

Corso di Laurea Magistrale

Secondo progetto di Metodi per il Calcolo Scientifico

ANNO ACCADEMICO 2024/2025

Farioli Alessio - 879217

Isceri Alessandro - 879309

Indice

1. Introduzione

1.1 Obiettivi del Progetto

Questo progetto è diviso in due parti: nella prima parte l'obiettivo è quello di implementare la **DCT2** e confrontare i tempi di esecuzione della versione vista in aula con la versione implementata dalla libreria open source **Scipy**, la quale utilizza **FFT** (Fast Fourier Transform).

Nella seconda parte l'obiettivo è la creazione di un software che, ricevute in input un'immagine in scala di grigi, la soglia di taglio delle frequenze e la dimensione dei blocchi in cui dividere l'immagine, ne effettui la compressione e la decompressione e riporti a schermo i risultati.

1.2 Strumenti Utilizzati

È stata utilizzata la libreria open source **Scipy** per effettuare la **DCT** e la libreria open source **PIL** per la gestione e la lettura delle immagini. Il linguaggio scelto per l'implementazione è stato **Python**.

Il sistema operativo sul quale è stato eseguito il codice è **Windows 11 Home**. Il computer utilizzato per l'esecuzione è un Modello **Acer Aspire A715-75G**, con processore **Intel Core i7-10750H** CPU @ 2.60GHz, 2592 Mhz.

2. Prima Parte

2.1 Descrizione DCT implementata

Per eseguire la procedura **DCT** (1-Dimensionale o 2-Dimensionale) viene inizialmente calcolata la matrice **D** tale per cui:

D(k, j) = α_k^N * cos(k*π*($\frac{2*j+1}{2*N}$)), dove:

$$\alpha_k^N = \begin{cases} \frac{1}{\sqrt{N}} \text{ se k = 1} \\ \frac{\sqrt{2}}{\sqrt{N}} \text{ altrimenti} \end{cases}$$

Una volta calcolata la matrice D, eseguire la DCT1D è triviale: basta effettuare un dot product tra D e il vettore x su cui vogliamo effettuare la DCT; mentre per eseguire la IDCT1D bisogna effettuare il prodotto tra la matrice D trasposta ed il vettore c che contiene le frequenze. Per effettuare una DCT2D invece, è sufficiente effettuare una DCT1D per righe e poi per colonne (o viceversa).

Infine, per calcolare la IDCT2D basta sostituire nell'algoritmo DCT2D la chiamata a DCT1D con una chiamata a IDCT1D.

Algorithm 1 DCT2D

```
function DCT2D(F, D)
    n ← len(f)
    C ← F
    for j = 0 to n do
        C[:, j] ← DCT1D(D, C[:, j])
    end for
    for i = 0 to n do
        C[i, :] ← DCT1D(D, C[i, :]^t)^t
    end for
    return C
end function
```

Di seguito viene riportato l'algoritmo per calcolare la matrice D basato sulla Trasformata Discreta di Fourier.

Algorithm 2 compute_D

```
function COMPUTE_D(n)
    D ← zeros[n, n]
    α ← 1/√n
    for k = 0 to n do
        for j = 0 to n do
            D[k, j] ← α * cos(k * π * (2*j+1)/(2*n))
        end for
        α ← √(2/n)
    end for
    return D
end function
```

2.2 Descrizione DCT FFT

La **Fast Fourier Transform** (FFT) è un algoritmo che permette di calcolare in minor tempo la trasformata discreta di Fourier, fondamentale nel calcolo della **DCT**. Infatti, come si può notare dall'algoritmo 2, il tempo di esecuzione è O(n^2), dato che è presente un doppio ciclo for innestato di lunghezza n, mentre l'algoritmo per la FFT impiega tempo O(n*log(n)), dove n è il numero di campionamenti della funzione.

Esistono diversi algoritmi per effettuare la FFT, ma tutti condividono la stessa complessità computazionale.

Gli algoritmi più famosi si basano sulla fattorizzazione di n. Scipy utilizza l'algoritmo di **Bluestein** nel caso in cui n non sia fattorizzabile efficacemente (ad esempio se n è un numero primo), altrimenti viene utilizzato l'algoritmo standard (**Cooley-Tookey**) per FFT, che lavora con massima efficienza quando n è una potenza perfetta di 2. Questo succede perchè l'algoritmo di Cooley-Tookey è basato sul paradigma **Divide-et-Impera** e lavora fattorizzando il numero n in questo modo: $n = n_1 * n_2$ ed effettua due chiamate ricorsive della Discrete Fourier Transform: una su n_1

ed una su n_2, per poi riassemblare i risultati ottenuti, inoltre effettua O(n) moltiplicazioni per l'unità immaginaria.

2.3 Risultati e loro analisi

Di seguito vengono confrontate le performance delle due implementazioni dell'algoritmo per la DCT bidimensionale, ovvero quella realizzata ex novo e quella implementata dalla libreria di Scipy che utilizza la FFT.

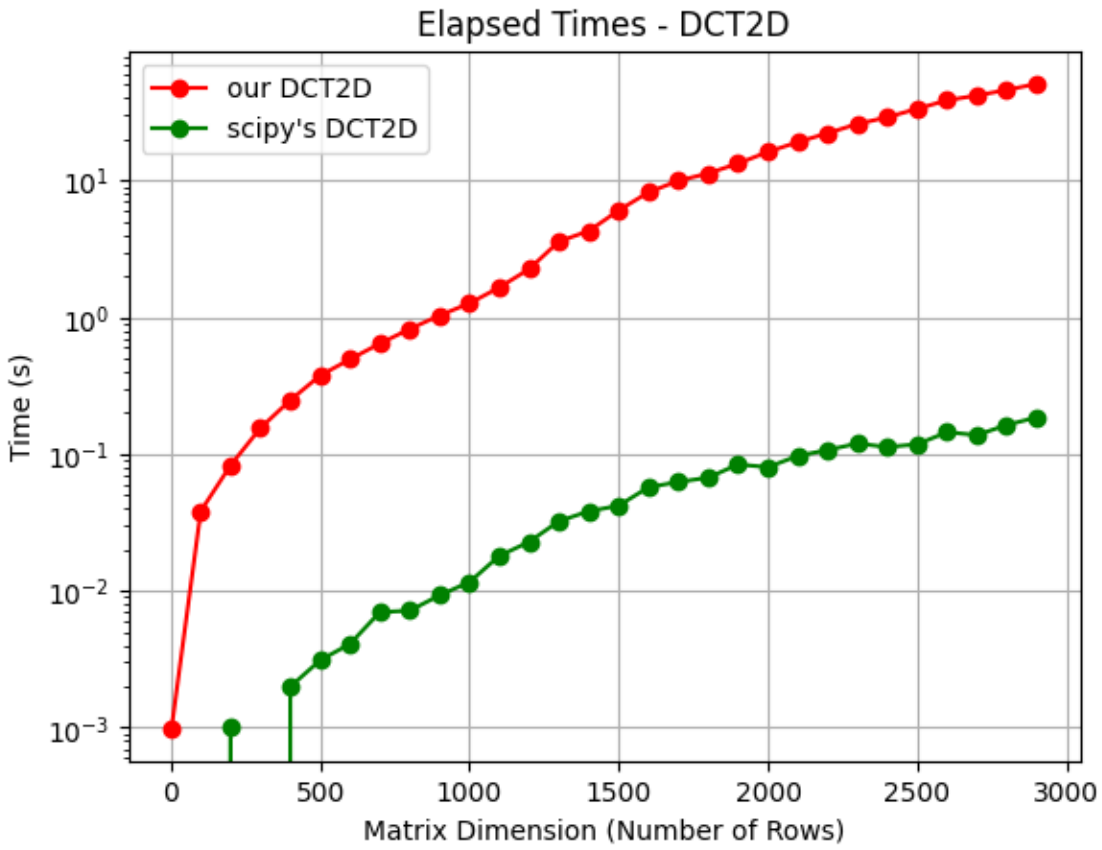

Figura 2.1: Confronto di tempi di esecuzione delle due implementazioni DCT.

Come si può vedere dall'immagine, e come prevedibile, l'implementazione della DCT di Scipy, descritta in 2.2, è decisamente più efficiente in termini computazionali della DCT implementata per il progetto, descritta in 2.1.

Questo succede perchè, come visto prima 2, l'algoritmo implementato impiega un tempo quadratico per effettuare il calcolo della matrice D, dovuto ad un doppio ciclo for, mentre l'algoritmo di

Cooley-Tookey riesce ad effettuare la DCT2 in un tempo minore, con complessità computazionale pari ad O(n*log(n)), aumentando di molto l'efficienza grazie al paradigma Divide-et-Impera.

3. Seconda Parte

3.1 Architettura

Per la creazione dell'applicativo è stato usato il pattern **MVC** (Model-View-Controller), il quale permette di separare l'interfaccia grafica dalla logica applicativa e prevede che l'utente usi l'interfaccia (**View**) per interagire con il programma. Il **Controller** si occupa di ricevere gli input dall'interfaccia e di delegare i compiti attraverso le funzionalità esposte dal **Model**.
Nell'interfaccia utente è possibile specificare tre parametri:

- **F**: la dimensione dei blocchi in cui verrà suddivisa l'immagine (numero intero).
- **d**: la soglia di taglio delle frequenze (numero intero $\in [0, 2 * F - 2]$).
- **Path**: il percorso dell'immagine in scala di grigi da comprimere (selezionabile attraverso il file manager del proprio computer).

Il codice è strutturato in cinque cartelle:

1. **View**: contiene il file *interface.py*, dove viene definita la User Interface.
2. **Controller**: contiene il file *controller.py*, che ha il compito di gestire le chiamate prodotte dall'utente quando viene utilizzata l'interfaccia e chiamare le funzioni del Model per eseguire ciò che viene richiesto dall'utente (in questo caso la compressione di un'immagine).
3. **Model**: contiene diversi files, tra cui *DCT1D.py*, *DCT2D.py* che contengono le funzioni per eseguire la DCT (in una o due dimensioni) descritte in 2.1, il file *utils.py* che contiene le funzioni per leggere un'immagine in scala di grigi, per mostrare le immagini affiancate (originale e post elaborazione), per effettuare il taglio delle frequenze e infine la funzione per arrotondare i valori e riportarli nel range $[0, 255]$ qualora fosse necessario. Infine *compression_scipyDCT.py* permette di eseguire la compressione dell'immagine con la DCT "esterna", ovvero quella presente nella libreria Scipy, descritta in 2.2.

4. **Extra_Compressors**: contiene i file *compression_ourDCT.py*, *compression_ourDCT_multiprocess.py* che permettono di eseguire la compressione dell'immagine in due modi differenti: il primo effettua la compressione utilizzando la DCT implementata secondo l'algoritmo descritto in 2.1, ed il secondo effettua la compressione utilizzando la stessa metodologia, ma con un approccio multiprocesso per provare a risparmiare del tempo di elaborazione.

5. **Test**: Un insieme di files che permette di ottenere una visualizzazione grafica dei risultati ottenuti in termini di efficienza computazionale confrontando l'approccio multiprocesso con quello a processo singolo che usa la DCT vista in aula o quella di Scipy a seconda del test.

3.2 Immagini e Risultati

3.2.1 Bridge

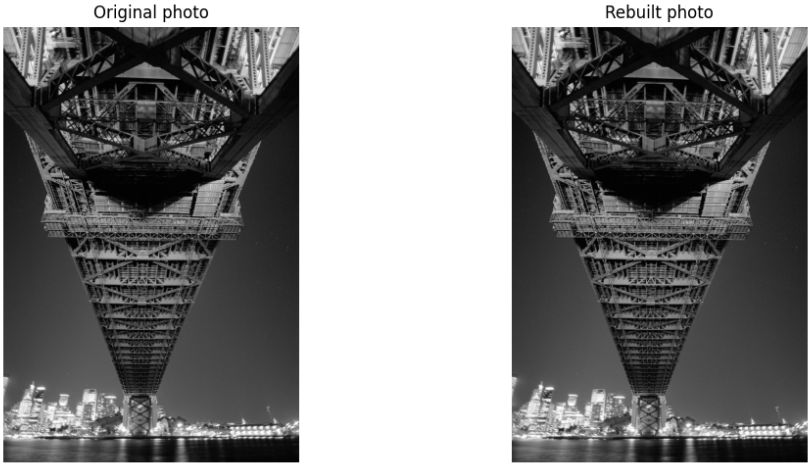

Figura 3.1: Compressione di un'immagine di un ponte con parametri F=8 e d=10.

Come si può notare dall'immagine sopra riportata, con dei parametri di compressione sensati per un'immagine di grandi dimensioni l'immagine compressa e poi decompressa è una riproduzione abbastanza fedele dell'immagine originale.

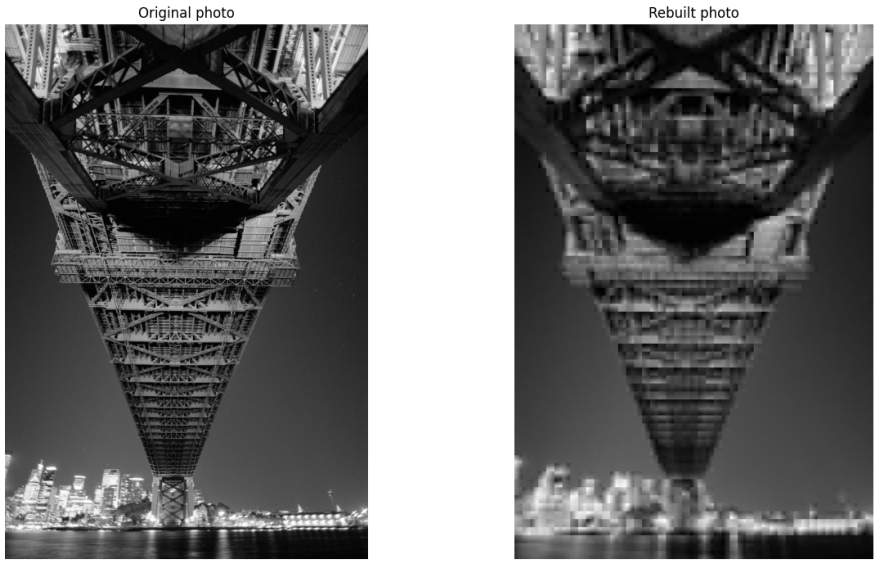

Figura 3.2: Compressione di un'immagine di un ponte con parametri F=60 e d=2.

In questo caso, a causa dell'elevato taglio delle frequenze (d = 2) si può notare che nelle zone di maggior contrasto l'immagine tende ad essere molto differente dall'originale. Ad esempio, osservando il ponte si nota una riduzione della qualità dell'immagine, mentre se si osserva il cielo, che ha un contrasto ridotto, l'immagine compressa e decompressa non è così sgranata.

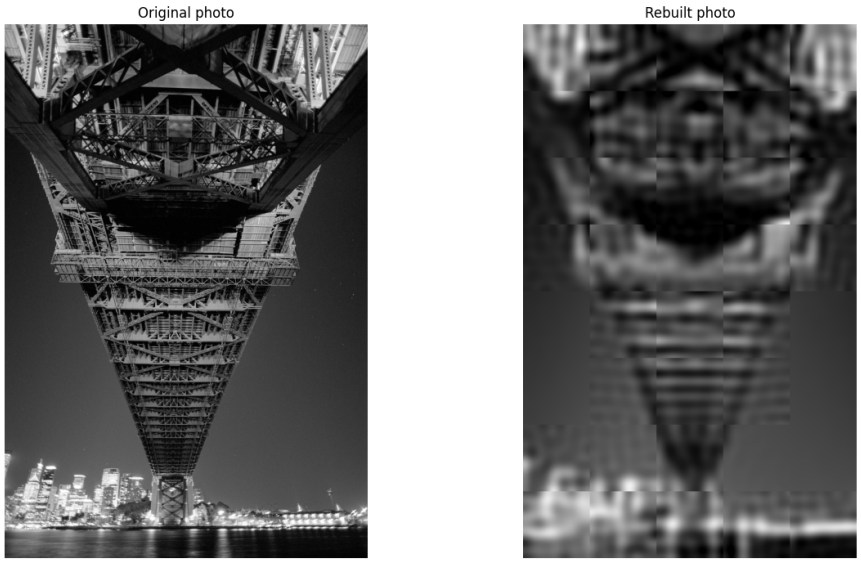

Figura 3.3: Compressione di un'immagine di un ponte con parametri F=500 e d=10.

Nell'immagine sopra si può vedere come l'utilizzo di blocchi troppo grandi porta ad una distorsione notevole dell'immagine. Infatti è possibile vedere chiaramente i blocchi in cui l'immagine è stata suddivisa.

L'immagine inoltre risulta di bassa qualità a causa dell'elevato taglio delle frequenze (d = 10) per dei blocchi molto grandi (F = 500).

3.2.2 C

Figura 3.4: Compressione di un'immagine di una lettera c con parametri F=8 e d=10.

Come si può notare dall'immagine sopra riportata, utilizzando dei parametri standard (come quelli usati in prima battuta per l'immagine del ponte) anche una foto di piccole dimensioni può essere compressa abbastanza bene.

Si nota in particolare che alcuni pixel bianchi all'interno della lettera C vengono "sporcati". Questo accade probabilmente perchè l'immagine è di piccole dimensioni e quindi questi dettagli si notano a occhio nudo. Per un'immagine abbastanza grande come quella del ponte, questi dettagli sono sicuramente trascurabili nella visione d'insieme.

Inoltre ciò è anche dovuto al fatto che ci sono salti drastici nei colori, motivo per cui i bordi della lettera vengono particolarmente distorti nonostante venga salvato un buon numero di frequenze.

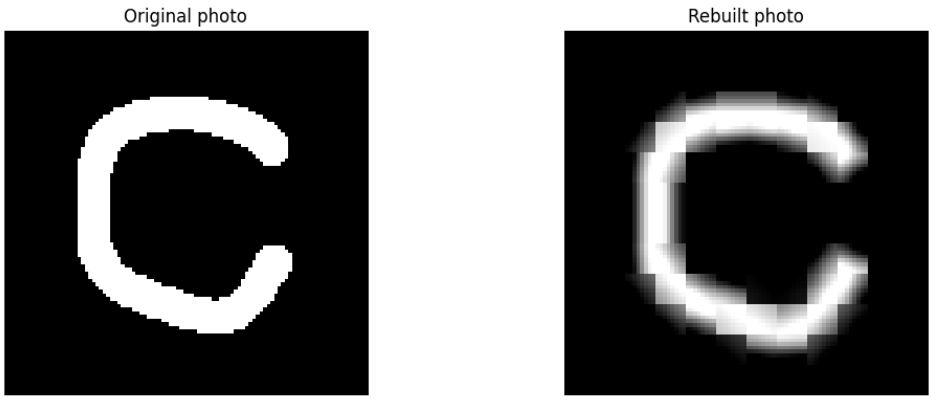

Figura 3.5: Compressione di un'immagine di una lettera c con parametri F=8 e d=2.

Nel caso di un aumento del taglio delle frequenze si nota che la sgranatura peggiora ancora, sempre a causa della difficoltà di comprimere i salti drastici di colore.

3.2.3 Gradiente

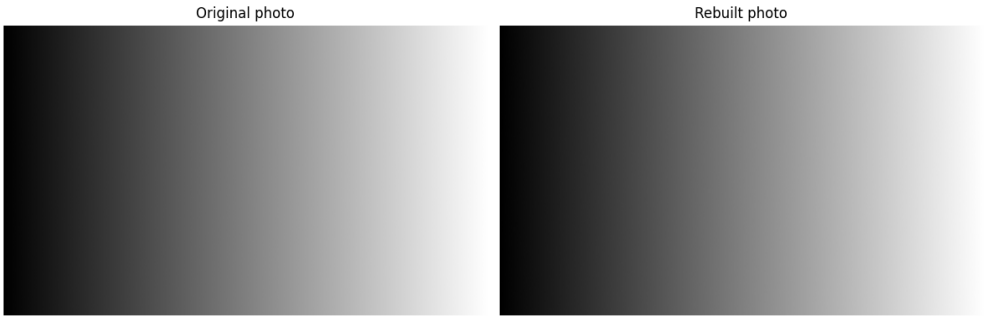

Figura 3.6: Compressione di un'immagine di una sfumatura con parametri F=8 e d=10.

Nel caso del gradiente, utilizzando i parametri "standard", la compressione dell'immagine è molto buona. Questo è anche dovuto al fatto che i salti di colore in questa foto sono molto graduali, e quindi facili da gestire.

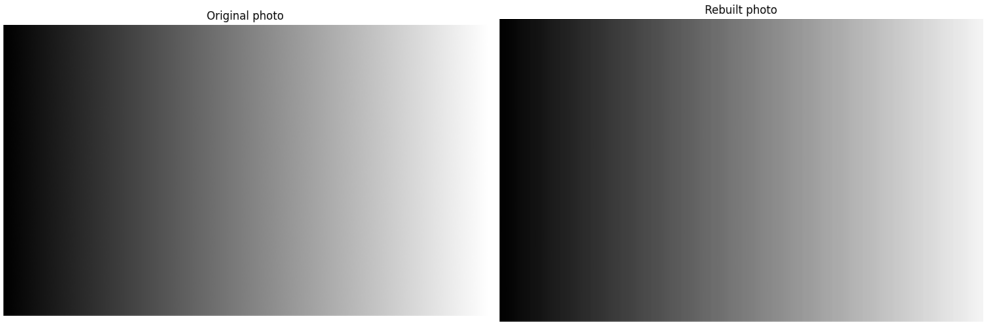

Figura 3.7: Compressione di un'immagine di una sfumatura con parametri F=60 e d=2.

Anche in questo caso, grazie al fatto che, come precedentemente detto, i cambiamenti di colore sono graduali e non improvvisi, anche il taglio di un elevato numero di frequenze e l'aumento di dimensione dei blocchi non crea notevoli distorsioni.

3.3 Extra - Multiprocessing

Per vedere se era possibile risparmiare tempo durante la procedura di compressione e decompressione di un'immagine, è stato implementato un algoritmo multiprocesso.

In questa procedura viene presa in input l'immagine (contenuta in un'area di memoria condivisa), viene partizionata in p (numero di processi) sotto-immagini, con p scelto dall'utente, e successivamente viene creato un processo per ogni sotto-immagine. Ogni processo lavora solamente slla propria sotto-immagine, la quale viene divisa in blocchi di lato costante pari ad F. Su ogni blocco si effettua uno shift dei valori pari a -128, si applica la DCT2D come descritta in 2.1, si arrotondano i valori per ottenere solo numeri interi, si tagliano le frequenze ed infine il processo posiziona il blocco compresso in un'area di memoria comune tra i processi che conterrà la immagine compressa.

Nella decompressione invece, si parte dall'area di memoria che contiene l'immagine compressa, si creano p processi, ogni processo lavora su una parte dell'immagine e ha il compito di dividerla in blocchi di dimensione F, applicare ad ogni blocco la IDCT2. Successivamente verrà applicato

uno shift di +128 a tutta la matrice ed inoltre verranno aggiustati i valori tramite arrotondamento e verranno rimessi in scala i valori superiori a 255 o inferiori a 0. Infine verrà posizionato il blocco decompresso nell'area di memoria condivisa che conterrà la foto decompressa.

Nel caso dei grafici sotto presenti, è stato deciso il valore 8 per F, mentre il valore di p varia tra 2, 4 e 6. La soglia di taglio scelta è 5. Le immagini compresse generate sono identiche tra DCT2D con e senza multiprocessing.

La procedura per comprimere e decomprimere implementata nel Model (ovvero quella richiesta per il secondo obiettivo) è equivalente a quella descritta sopra ma con p = 1, intoltre utilizza la DCT e IDCT di Scipy, descritta in 2.2.

Infine, è stata implementata una terza versione del metodo di compressione e decompressione equivalente a quella riportata sopra ma che utilizza la DCT e IDCT descritte in 2.1.

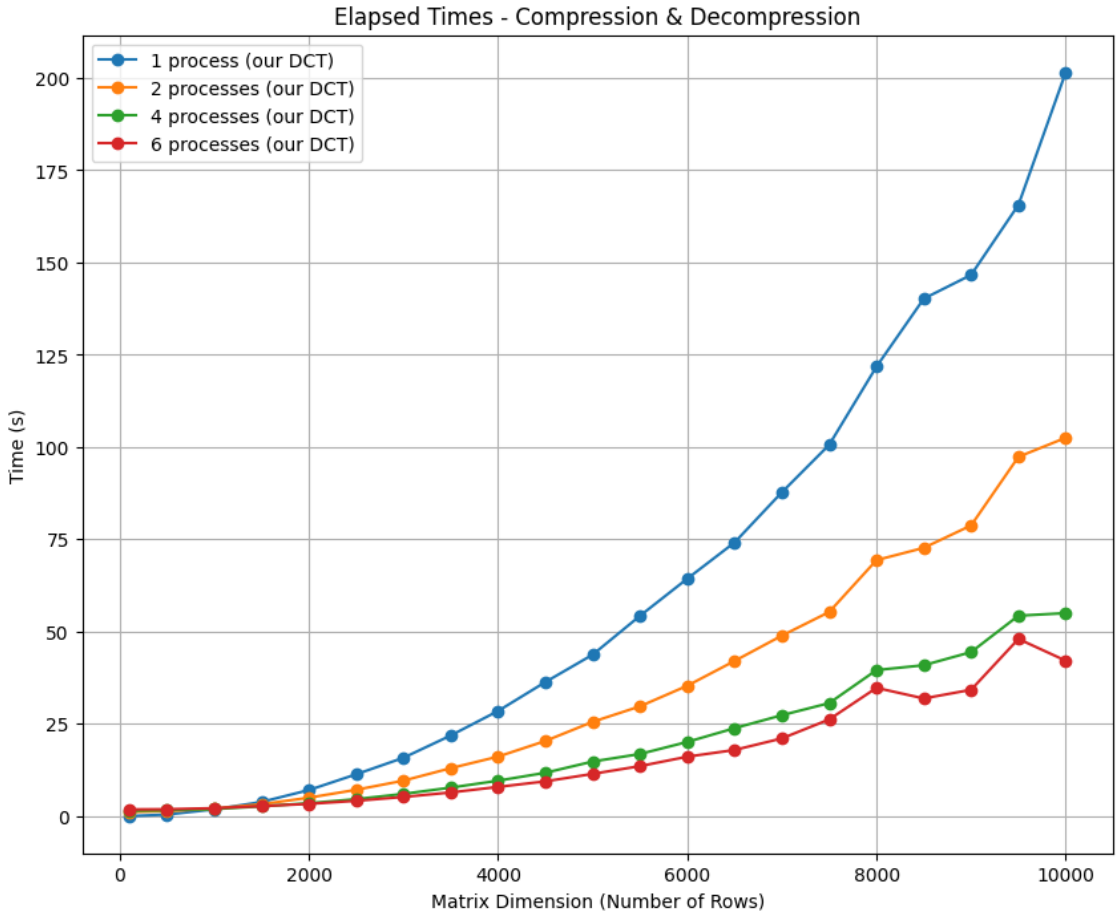

Figura 3.8: Confronto dei tempi della compressione e decompressione single process che usa la DCT vista in aula con quella multiprocesso.

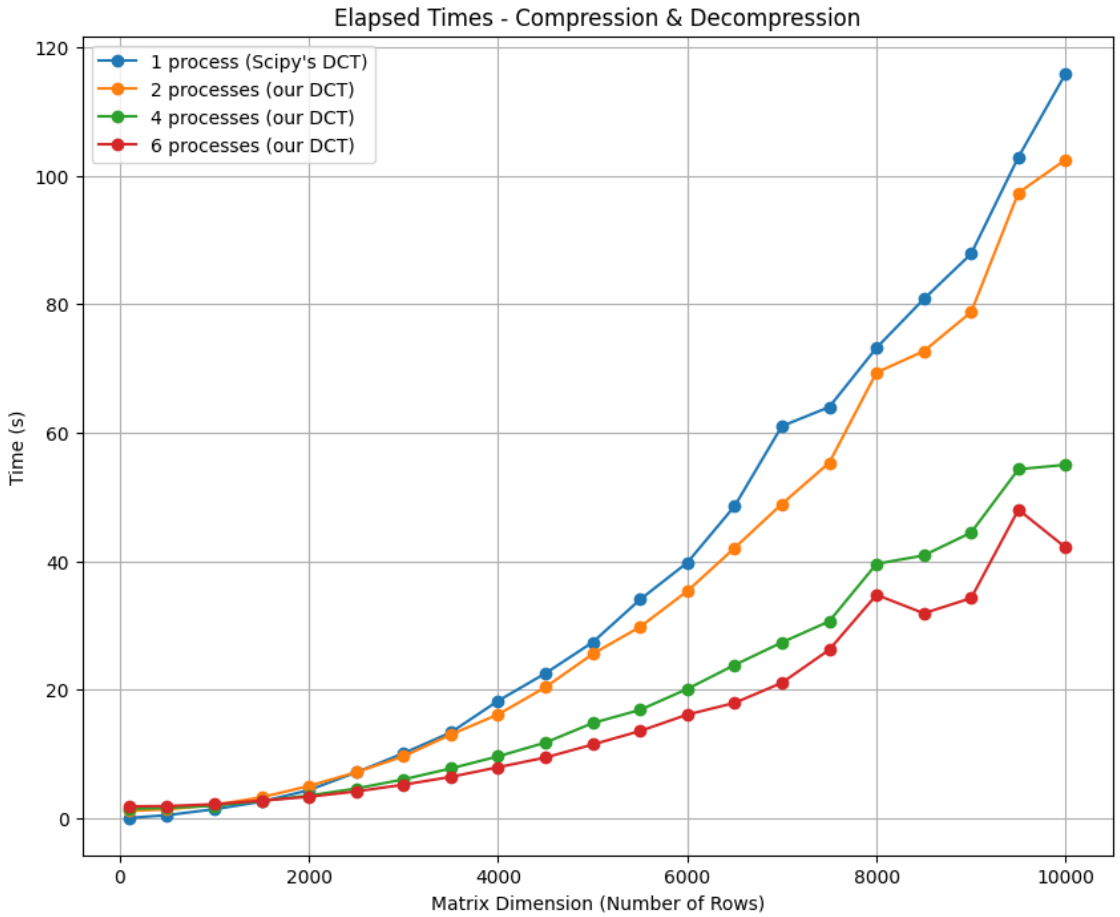

Figura 3.9: Confronto dei tempi della compressione e decompressione single process che usa la DCT di Scipy con quella multiprocesso.

Dai risultati ottenuti si può vedere che l'approccio multiprocesso riduce notevolmente i tempi di calcolo, soprattutto nel caso in cui si confrontino con la procedura single process che utilizza la DCT vista in aula (immagine 3.8). Infatti con soli due processi, per n $\geq$ 4000, si risparmia circa la metà del tempo e se si considera l'utilizzo di 4 o 6 processi il miglioramento è ancora maggiore. Anche confrontando i tempi di esecuzione con la procedura single process che utilizza la Fast Fourier Transform (immagine 3.9) si nota un buon miglioramento, soprattutto se vengono utilizzati 4 o 6 processi.

4. Conclusioni

Per quanto riguarda la prima parte del progetto, come previsto l'approccio che utilizza la Fast Fourier Transform è notevolmente più rapido dell'implementazione discussa in aula.
Per quanto riguarda la seconda parte del progetto, sono stati effettuati numerosi test per studiare il comportamento del software creato, così da verificare gli effetti di selezione di blocchi troppo grandi o il taglio di un numero eccessivo di frequenze. Sono anche state testate immagini con contrasto più o meno intenso per vedere come l'approssimazione migliorava o peggiorava, e, come previsto, la compressione riduce la qualità dell'immagine nelle zone di alto contrasto, come passaggi improvvisi tra pixel bianchi e neri. Infine, per provare a migliorare i tempi di esecuzione, è stato implementato un algoritmo di compressione e decompressione con approccio multiprocesso, e, dopo averne analizzato i risultati prodotti, è stato constatato che tale approccio porta ad un considerevole risparmio in termini di tempo, e come possibile sviluppo futuro si potrebbe estendere ulteriormente questo algoritmo facendo in modo che utilizzi la DCT di Scipy, e analizzando quanto altro tempo può essere risparmiato con la combinazione degli approcci. Si noti inoltre, che il software sviluppato lavora solamente con immagini in bianco e nero, e quindi deve applicare poche trasformazioni ai singoli blocchi; infatti, nel caso di immagini colorate, il numero di trasformazioni applicata a ciascun blocco sarebbe stata maggiore, e quindi l'approccio multiprocesso avrebbe permesso di risparmiare ulteriore tempo rispetto all'approccio standard.