# ELABORATO III

Prodotto matrice-matrice su architettura MIMD-DM

UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"
ROBERTO BASILE GIANNINI
N97000340

# Descrizione del problema

Si vuole realizzare un algoritmo per il calcolo del prodotto matrice-matrice in ambiente di calcolo parallelo su architettura MIMD a memoria distribuita, sfruttando gli strumenti offerti dalla libreria MPI. Le matrici coinvolte sono due e sono quadrate: per ragioni visive, si è scelto di utilizzare come elementi dei numeri interi. Gli elementi delle matrici *A* e *B* vengono generati random in un intervallo che va da 1 a 9: anche in questo caso, la scelta è stata presa esclusivamente per garantire un risultato a video più chiaro per l'individuazione delle colonne. Essendo ambo le matrici quadrate, l'unico parametro esplicitato (che rappresenterà dunque numero di righe e di colonne delle due matrici) viene indicato con *m* e l'utente (tramite file *.pbs*) ha la facoltà di scegliere il suo valore.

I processori sono organizzati su una griglia bidimensionale quadrata, dunque di dimensione *pxp*: anche in questo caso l'utente ha la facoltà di stabilire la dimensione della griglia. Questa condizione impone che il numero di processori coinvolti sia quadrato. Un'altra condizione che deve essere garantita è che *m* sia multiplo di *p*.

La strategia implementata è quella *Broadcast Multiply Rolling*.

# Input e Output

Il software richiede in ingresso i seguenti parametri:

- *int p*, dimensione della griglia quadrata;
- *int m_input*, dimensione delle matrici A e B quadrate.

Gli input vengono specificati tramite file *.pbs*, indicandoli con lo stesso ordine col quale sono stati presentati:

*p m_input*

Chiaramente, su questi parametri verranno effettuati opportuni controlli per garantire il rispetto di tutte le condizioni del caso, come sarà più chiaro nella sezione relativa agli *Indicatori di errore*. Come anticipato, gli elementi delle matrici non vengono specificati dall'utente, ma generati random.

Gli output a video sono i seguenti:

- stampa delle matrici A e B generate;
- eventuali messaggi di errore ed opportuna gestione;
- stampa dei *primi* blocchi $A_{i,j}$ e $B_{i,j}$ assegnati ad ogni processore;
- stampa del risultato e del tempo di esecuzione.

# Descrizione dell'algoritmo

L'idea di base è quella di sfruttare gli strumenti messi a disposizione dalla libreria MPI al fine di realizzare un algoritmo parallelo tale da garantire alte prestazioni al crescere della dimensione del problema. Essendo un'architettura a memoria distribuita, è necessario gestire la comunicazione tra processori, fondamentale al fine di finalizzare correttamente il calcolo del prodotto matrice-matrice. Una rappresentazione fortemente astratta dell'algoritmo è la seguente:

```
1.  INIZIO
2.
3.      //0.
4.          Inizializzazione ambiente
5.
6.      //1.
7.          Lettura input ed inizializzazione matrici
8.
9.      //2.
10.         Creazione griglia e sottogriglie
11.
12.     //3.
13.         Distribuzione matrici
14.
15.     //4.
16.         BMR
17.
18.     //5.
19.         Stampa risultato e tempi
20.
21.     //6.
22.         Finalizzazione
23.
24. FINE
```

Nel *passo 0*, oltre alla dichiarazione di tutte le variabili che verranno utilizzate, avviene l'inizializzazione dell'ambiente MPI per mezzo delle rispettive funzioni di ambiente: a tutti i processori del communicator verrà assegnato un identificativo di cui verranno a conoscenza, insieme al numero di processori stessi. Questi parametri risulteranno indispensabili per la corretta individuazione dei processori demandati a specifici compiti (per esempio, il processore P0 si occuperà della lettura, generazione e distribuzione delle matrici), per la comunicazioni, per la corretta allocazione delle strutture dati e così via.

Nel *passo 1* il processore P0 provvede alla lettura della dimensione *p* della griglia (viene richiesto di inserire solo il numero di righe della griglia dal momento che essa dovrà essere quadrata) e della dimensione *m_input* delle matrici quadrate (medesima osservazione). Viene effettuato un primo controllo su *p*, per assicurarsi che il parametro inserito sia valido: qualora non dovesse essere valido, P0 stamperà l'errore *E0* e porrà $p = 2$, parametro previsto come di default. Successivamente, *p* viene spedito in broadcast sul *COMM_WORLD* e tutti i processori provvederanno a calcolare il numero di colonne della griglia $q = p/nproc$, aspettandosi che sia uguale a *p* stesso. Infatti, tutti i processori verificheranno che $q = p$, ovvero che la griglia sia quadrata, e in caso negativo P0 stamperà a video l'errore *E1* e l'esecuzione verrà terminata. In caso affermativo, invece, P0 stamperà le dimensioni della griglia quadrata. Anche su *m_input* sono previsti dei controlli: il processore P0 si assicurerà che sia valido (altrimenti errore *E2* e valore di default *16*) e che sia multiplo di *p* (altrimenti errore *E3* e valore di default *p*10*). Avendo tutte le informazioni del caso, P0 provvederà ad allocare e generare le matrici A e B, e tutti i processori provvederanno a ricavare le dimensioni dei blocchi e ad allocarli (inizializzando a 0 il blocco C). Il blocco A_bcast verrà opportunamente utilizzato nell'operazione di *Broadcast* nel passo 4.

Nel *passo 2* avviene la creazione della griglia e delle sotto-griglie: *griglia_r* relativamente alla sotto-griglia di riga, e *griglia_c* relativamente a quella di colonna. Prevedendo la necessità di far comunicare processori (di medesima riga o di medesima colonna) non direttamente connessi tra loro, è stata prevista periodicità sulle righe e sulle colonne. Successivamente la creazione delle griglie, al vettore delle coordinate dei singoli processori vengono assegnate le coordinate su griglia dello specifico processore.

Nel *passo 3* avviene la distribuzione delle matrici A e B. A tal proposito, è stato utilizzato lo strumento *MPI_Type_vector* al fine di creare un tipo di dato che avesse *blocco_rig* righe di *blocco_col* elementi, con passo *mat_col* al fine di costruire correttamente un blocco a partire da una matrice. Dunque, sia per la matrice A che per la matrice B, sfruttando il tipo appena definito, è stata effettuata un'operazione di scattering tra i processori con sorgente P0.

Nel *passo 4* viene implementata la strategia *Broadcast Multiply Rolling*. Ogni processore, per un numero di volte pari alla quantità di diagonali presenti sulla griglia, parteciperà alle operazioni *BMR*. In particolare, l'operazione di *Broadcast* viene riportata come segue:

```
...
//Broadcast
Se sei un processore sulla k-esima diagonale della griglia
    blocco A_bcast = blocco A

Bcast di blocco A_bcast su griglia_r con sorgente P_diagonale_k
...
```

Ogni processore, verificando e confrontando opportunamente le proprie coordinate sulle griglia, è in grado di stabilire se si tratta di un processore della *k-esima* diagonale della griglia (ovvero la diagonale che si ottiene al passo *k*) e in caso affermativo essere sorgente della successiva operazione di broadcast *sulla propria riga*, non prima di aver predisposto il blocco *A_bcast* da spedire. Infatti, il processore della diagonale *k-esima* dovrà spedire il proprio blocco A a tutti i processori presenti sulla propria riga, ed è per questo che l'operazione di broadcast viene effettuata sulla *griglia_r*. Questo blocco *A_bcast*, una volta ricevuto dai processori in questione, non andrà a sovrascrivere il proprio blocco A degli stessi.

L'operazione di *Multiply* viene riportata come segue:

```
...
//Multiply
blocco C = blocco C + blocco A_bcast * blocco B
...
```

Ogni processore incrementerà i singoli elementi del proprio blocco C (per questo motivo inizializzati a 0) con il valore degli elementi ottenuti dal prodotto matrice-matrice tra *A_bcast* e il blocco B: si osserva che è ancora presente un prodotto matrice-matrice, ma di una dimensione più piccola rispetto al problema di partenza, coerentemente con l'obiettivo di scomporre il problema in sotto-problemi più piccoli al fine di costruire un algoritmo parallelo.

L’operazione di *Rolling* viene riportata come segue:

```
...
//Rolling
Spedisci blocco B su griglia_c al processore della riga precedente
Ricevi blocco B su griglia_c dal processore della riga successiva
...
```

A questo punto, ogni processore dovrà spedire il proprio blocco B al processore presente *nella stessa colonna* (da qui l’utilizzo della *griglia_c*), ma appartenente alla riga immediatamente precedente (il processore esattamente “sopra” a quello che sta spedendo). Successivamente, lo stesso processore sovrascriverà il blocco B appena spedito con il blocco B ricevuto dal processore appartenente ancora alla sua stessa colonna, ma alla riga immediatamente successiva (il processore esattamente “sotto” a quello che sta ricevendo).

Ripetendo questi passaggi *p* volte, ogni processore calcolerà un blocco C della matrice risultato.

Nel *passo 5* avviene la stampa del risultato e dei tempi misurati, mentre nel *passo 6* avviene la terminazione dell’algoritmo.

Di seguito viene descritta in maggior dettaglio (mantenendo comunque una forte astrazione) la soluzione implementata:

```
//I codici di errore sono trattati nella sezione relativa agli Indicatori di errore
//la stringa di ingresso si presenta come: p (dim. griglia quadrata), m_input (dim. matrici quadrate)

INIZIO

    //0.
        Inizializzazione ambiente

    //1.

        //lettura e controllo input
        Se sei P0
            Leggi p

            Se p < 1
                Stampa errore E0
                p = 2

        Bcast di p su COMM_WORLD
        q = nproc/p

        Se p e q sono diversi
```

```
            Se sei P0
                Stampa errore E1

            Terminazione
            FINE
        Altrimenti
            Se sei P0
                Stampa dimensioni della griglia

        Se sei P0
            Leggi m_input

                Se m_input <1
                    Stampa errore E2
                    m_input = 16

                Stampa m_input

                Se m_input non è multiplo di p
                    Stampa errore E3
                    m_input = p*10

            //costruzione matrici
            Inizializza dimensioni matrici A e B
            Alloca e inizializza matrici A e B
            Stampa A
            Stampa B

            //dimensioni blocchi
            Inizializza dimensioni blocchi A e B

        Bcast delle dimensioni dei blocchi A e B
        Inizializza dimensioni blocco C

        //costruzione blocchi
        Alloca blocco A
        Alloca blocco A_bcast
        Alloca blocco B
        Alloca blocco C
        Inizializza a 0 gli elementi del blocco C

    //2.
        //creazione griglia e sottogriglie
        Creazione griglia
        Creazione griglia_c
        Creazione griglia_r
        Allocazione ed inizializzazione delle coordinate

        Sincronizza processori
    //3.
        //distribuzione matrici
        Creazione tipo Blocco con MPI_Type_vector

        Distribuzione A in blocchi su griglia
        Distribuzione B in blocchi su griglia

```

```
        Stampa proprio blocco A
        Stampa proprio blocco B

        Sincronizza processori
    //4.
        //BMR
        Inizia misura tempo
        Per ogni k<p

            //Broadcast
            Se sei un processore sulla k-esima diagonale della griglia
                blocco A_bcast = blocco A

            Bcast di blocco A_bcast su griglia_r con sorgente P_diagonale_k

            //Multiply
            blocco C = blocco C + blocco A_bcast * blocco B

            //Rolling
            Spedisci blocco B su griglia_c al processore della riga precedente
            Ricevi blocco B su griglia_c dal processore della riga successiva

            Fine misura tempo

         //5.
             Stampa risultato e tempi

         //6.
             Finalizzazione

    FINE
```

# Indicatori di errore

Gli errori previsti riguardano dati di input non validi e condizioni su di essi non rispettate. In quasi tutti i casi si è deciso di gestire l'errore imponendo dei valori di default al fine di continuare l'esecuzione del programma, mentre in un solo caso si è deciso di terminare l'esecuzione.

Di seguito vengono riportate le diverse circostanze di errore ed indicata la rispettiva gestione:

- *Dimensione della griglia quadrata < 1*

  In questo caso si è deciso di continuare l'esecuzione del programma imponendo come valore di default *2*, così da avere una griglia *2 x 2*.

```
1.  //controllo validità di p
2.  if (p < 1) {
3.      printf ("[E0] Dimensione griglia %d non valida. Valore default: 2 \n", p);
4.      p = 2;
5.  }
```

- *Numero di righe della griglia e numero di colonne della griglia non sono uguali e dunque la griglia non è quadrata*

  In questo caso si è deciso di terminare l'esecuzione.

```
1.  //controllo griglia quadrata
2.  if (p != q) {
3.      if (menum == 0)
4.          printf("[E1] p e q sono diversi: griglia non quadrata. Terminazione... \n");
5.
6.      MPI_Finalize();
7.      return -1;
8.  }
```

- *Dimensione delle matrici quadrate non valida*

  In questo caso si è deciso di continuare l'esecuzione, ponendo come valore di default *16.*

```
1.  //controllo validità di m_input
2.  if (m_input < 1) {
3.      printf ("[E2] Dimensione input %d non valido. Valore default: 16. \n", m_input);
4.      m_input = 16;
5.  }
```

- *Dimensione delle matrici quadrate non è multiplo della dimensione della griglia quadrata*

  In questo caso si è deciso di continuare l'esecuzione, ponendo come valore di default *dimensione della griglia * 10.*

```
1.  //controllo m_input multiplo di p
2.  if ( (m_input%p) != 0) {
3.      printf ("[E3] Dimensione input %d non e' multiplo di p = %d. Valore di default:
    p*10. \n", m_input, p);
4.      m_input = p*10;
5.  }
```

Sono presenti dei file *.pbs* simulanti gli errori *E0, E1, E2, E3*, e i rispettivi file *.out* con i risultati ottenuti.

# Subroutine

Sono state utilizzate diverse funzioni non appartenenti alla libreria standard del C, alcune delle quali appartenenti alla libreria MPI, altre scritte personalmente. Di seguito verranno trattate individualmente, specificando testata, valori di input e output, e funzionamento.

**MPI**

Tra le funzioni di MPI coinvolte, cominciamo con la descrizioni delle funzioni di ambiente:

- *int MPI_Init(int *argc, char **argv[])*, si occupa di inizializzare l'ambiente a partire dagli argomenti di input del main: *int argc*, un intero che indica il numero di parole che indichiamo sulla riga di comando e *char *argv[],* un doppio puntatore a carattere, ovvero un vettore di stringhe che sono esattamente le parole scritte sulla riga di comando. *argv[0]* è il nome del programma, mentre i restanti saranno gli argomenti con i quali (nel nostro esempio) facciamo la somma. Con essi, la routine crea i vari processi che abbiamo lanciato e crea il common world. Si tratta sostanzialmente della prima necessaria operazione che deve essere eseguita, al fine di inizializzare l'ambiente MPI;

- *int MPI_Finalize(),* si occupa di chiudere i processi, l'ambiente, cestina i communicator liberando dunque gli identificativi e quindi si può terminare il programma. È l'operazione che deve sempre giungere alla fine del codice;

- *int MPI_Comm_rank(MPI_COMM_WOLRD, &menum)*, permette al processo chiamante di conoscere (specificando in ingresso il communicator di appartenenza) il proprio identificativo e lo salva nella variabile menum (il cui riferimento viene posto in input). Ovviamente, ogni processo avrà un suo identificativo univoco;

- *int MPI_Comm_size(MPI_COMM_WOLRD, &nproc),* restituisce il numero di processi totale (relativo al communicator specificato in ingresso) ed immagazzina questa informazione in *nproc*.

Procediamo adesso con la descrizione delle funzioni di comunicazione:

- *int MPI_Isend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request * request)*, si tratta dell'equivalente *non bloccante* di *MPI_Send*. Viene utilizzata dal processore mittente per inviare un dato contenuto in **buf*, avente *count* elementi, di tipo *datatype*, al processo destinatario identificato con *dest*, associando al messaggio l'identificativo univoco *tag* e tramite il communicator *comm* (che dovrà corrispondere a quello del ricevitore). In questo caso è presente un argomento *request* che mantiene informazioni su tutta la trasmissione, relativamente anche alla spedizione, oltre che alla ricezione. Esistono delle funzioni specifiche, utili a conoscere lo stato della spedizione ed eventualmente a comportarsi di conseguenza, come per esempio *MPI_Test* o *MPI_Wait*, che hanno in ingresso l'elemento *MPI_Request* relativo al messaggio inviato (nel primo caso ci si limita a fare un controllo, nel secondo in caso di non ancora avvenuta ricezione ci si mette in attesa);

- *int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status),* utilizzata dal processo destinatario che attende un messaggio da immagazzinare in **buf*, avente al massimo *count* elementi, di tipo *datatype*, da parte del processo con identificativo *source*, messaggio identificato univocamente con *tag*, tramite il communicator *comm* e avente lo stato **status*. Lo status contiene per l'appunto lo stato del messaggio, incluse le dimensioni e le informazioni relative alla ricezione. Il destinatario, dunque, può aggiornare lo stato di ricezione del messaggio operando sulla variabile *status*. Si tratta di un'operazione *bloccante*;

- *int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm ),* a differenza delle prime due, questa operazione di comunicazione è di natura collettiva. L'istruzione deve essere eseguita da tutti i processi coinvolti, senza una diretta discriminazione tra chi invia e chi riceve (cosa che avviene nel caso della comunicazione uno a uno). *int root* indica l'identificativo del processore che spedirà il messaggio (di tipo *datatype* e nel communicator *comm*). Per il *root* il *buffer* indicherà la struttura (di *count* elementi) dalla quale prelevare i dati da inviare, mentre per i processori riceventi indicherà la struttura (di *count* elementi) nella quale immagazzinare il dato spedito dal *root*. Anche in questo caso abbiamo un comportamento bloccante.

- *int MPI_Scatterv(const void *sendbuf, const int* sendcounts, const int* displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)*, dove in questo caso il *root* sarà relativo ad un processore che spedisce dei dati che giungeranno ai destinatari in porzioni. La differenza col broadcast sta nel fatto che con quest'ultimo il dato inviato dal *root* e ricevuto dai destinatari è sempre lo stesso, mentre nel caso dello scattering i diversi destinatari riceveranno diverse porzioni di dati. Il *sendbuf* indica la struttura soggetta alla spedizione (nel nostro caso il vettore che rappresenta la matrice), e dunque significativa solo per il *root*. *sendcounts* è il vettore contenente il numero di elementi da spedire: l'i-esimo elemento di questo vettore, indica la quantità di elementi da spedire all'i-esimo processore, di tipo *sendtype*. Analogamente, il vettore *displs* contiene, all'i-esima posizione, il displacement (relativo al *sendbuf* ) della i-esima spedizione. *recvbuf* indica la struttura nella quale memorizzare i *recvcount* elementi ricevuti dal *root*, di tipo *recvtype*. Tutto ciò nel communicator *comm*.

Per quanto riguarda le operazioni collettive:

- *int MPI_Barrier( MPI_Comm comm )* che permette di sincronizzare tutti i processi appartenenti ad un communicator *comm*, nel senso che quando i processi trovano questa primitiva si fermano e riprendono l'esecuzione tutti insieme solo quando è arrivato anche l'ultimo processo: viene annullata qualsiasi differenza di tempo tra i processi;

- *int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)*, abbiamo un processo designato con *root* che dovrà effettuare una operazione *op* su una serie di dati. Questi dati vengono inviati dai processi partecipanti alla riduzione, dunque spediranno un sendbuf di count elementi e di tipo *datatype*, dati che giungeranno quindi al processo *root*, appartenente al medesimo communicator *comm*. Il *recvbuf* riguarda esclusivamente il *root* e conterrà il risultato dell'operazione *op* sugli elementi inviati a quest'ultimo.

- *int MPI_Cart_create(MPI_Comm comm_old, int dim, int* ndim, int* period, int reorder, MPI_Comm * comm_cart)*, a partire dal communicator *comm_old*, viene restituito un nuovo communicator *comm_cart* dove i processi sono organizzati in una griglia di dimensioni *dim*. L'i-esima dimensione della griglia è riportata in *ndim[i]*, mentre il vettore *period* dà informazioni sulla periodicità delle dimensioni. Se per esempio *period[i] = 1*, allora la dimensione *i* sarà periodica (nel caso della griglia, se *i* indica la dimensione delle righe, ciò vorrà dire che l'ultimo processore di riga potrà comunicare con il primo della sua stessa riga), non lo sarà se *period[i] = 0*. Se *reorder* è pari a 1, gli identificatori vengono riordinati in base al nuovo communicator, se è 0 ciò non avviene;

- *int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int *coords)*, dato il communicator *comm*, ogni processore *rank* calcola le proprie *maxdims* coordinate *coords*;

- *int MPI_Cart_sub(MPI_Comm comm, const int *remain_dims, MPI_Comm *newcomm)*, il communicator *comm* viene partizionato in sotto-communicator *newcomm*, specificando con il vettore *remain_dims* quali dimensioni vengono mantenute (*remain_dims[i] = 1* ) e quali no (*remain_dims[i] = 0* ). È tramite questo strumento che costruiamo le sotto-griglie di riga e di colonna.

Per la misura dei tempi è stata utilizzata la funzione *double MPI_Wtime()* che restituisce una misura del tempo in secondi. Ulteriori routine sono stata utilizzate per la costruzione del tipo di dato relativo ai blocchi di matrici:

- *int MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype * newtype)*, si tratta di una funzione che crea un *newtype* replicando un esistente *MPI_Datatype oldtype* un certo numero di volte in un blocco. Il numero di blocchi è dato da *count*, mentre la quantità di elementi contenuti è data da *blocklength*. L'input *stride* indica la distanza tra ogni blocco, espressa in numero di elementi;

- *int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint extent, MPI_Datatype * newtype)*, utilizzata al fine di resettare il lower bound e l'upper bound dell'*oldtype* creato. Il lower bound è posto pari a *lb*, mentre per settare l'upper bound, bisogna settare l'*extent* (dal momento che *extent = upper bound – lower bound*). Il *newtype* sarà dello stesso tipo di *oldtype* ma con lower bound ed upper bound opportunamente settati;

- Il tipo appena creato va *committato* tramite la funzione *int MPI_Type_commit (MPI_Datatype *newtype).*

## Funzioni di supporto

Sono state personalmente scritte due funzioni di supporto, riguardanti la distribuzione delle matrici e la creazione della griglia. Di seguito:

- *void crea_griglia(MPI_Comm *griglia, MPI_Comm *grigliar, MPI_Comm *grigliac, int menum, int nproc, int grid_rig, int grid_col, int *coordinate)*, utilizzata per la creazione della griglia bidimensionale *griglia* e delle sotto-griglie *grigliar* (communicator di riga) e *grigliac* (communicator di colonna). La griglia ha numero di righe *grid_rig* e numero di colonne *grid_col*. L'identificatore *menum* del processore viene specificato al fine di garantire l'opportuna assegnazione delle *coordinate* ad ogni processore della griglia. La griglia è periodica, nel senso che presenta periodicità sia sulle righe che sulle colonne. Per l'implementazione di questa funzione sono state utilizzate le funzioni *MPI* per la costruzione di topologie, sotto-topologie e per l'assegnazione delle coordinate. Tali funzioni sono state trattate nel paragrafo precedente;

- *void distribuzione_mat( int menum, int nproc, int* mat, int* blocco, int mat_rig, int mat_col, int blocco_col, int blocco_rig, int p, int q, MPI_Comm comm)*, questa funzione si occupa della distribuzione della matrice *mat* (di dimensioni *mat_rig x mat_col* ), distribuzione alla quale partecipano *nproc* processori identificati con *menum*. La matrice viene decomposta in blocchi (ognuno memorizzato nella struttura *blocco* ) di dimensioni *blocco_rig x blocco_col*. I parametri *p* e *q* rappresentano il numero di righe e di colonne della griglia, utilizzati per la costruzione del vettore dei displacement al fine di garantire la corretta distribuzione dei blocchi tra i processori della griglia. Per l'implementazione di questa funzione sono state utilizzate le funzioni *MPI* relative alla creazione dei tipi, il resize del lower bound ed upper bound, e il loro commit. In più, è stata utilizzata la funzione *MPI_Scatterv*. Questi strumenti sono stati trattati nel paragrafo precedente.

# Analisi dei tempi

La misura dei tempi è stata effettuata per mezzo della funzione *double MPI_Wtime()*, assicurandoci che tutti i processori fossero sincronizzati al momento della misura. Gli istanti di tempo misurati sono stati due, rispettivamente memorizzati nelle variabili *t0* e *t1*:

- *t0* è stato memorizzato esattamente prima della prima istruzione della strategia BMR;

- *t1* è stato memorizzato esattamente dopo l'ultima istruzione della strategia BMR.

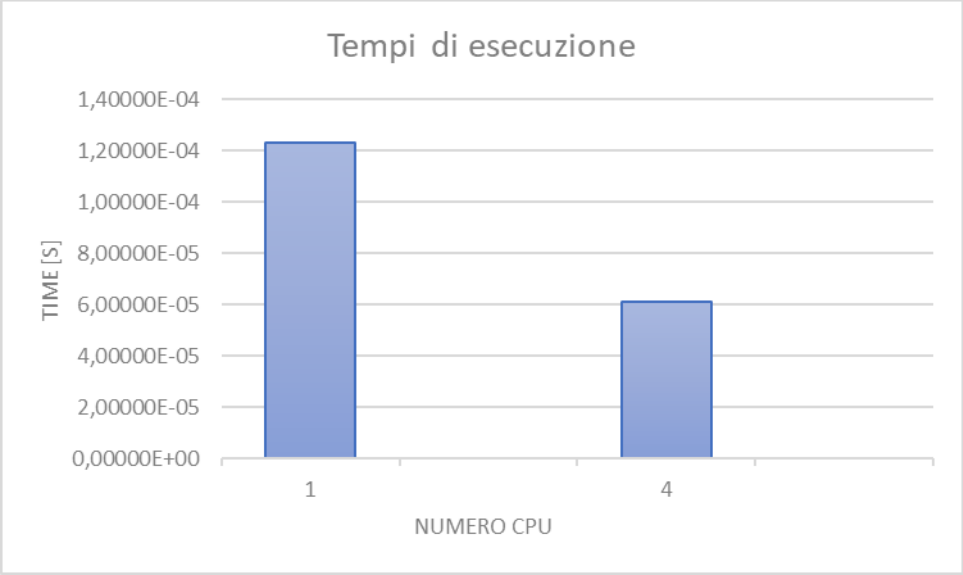
Ogni processore misura il proprio *tdiff = t1 – t0*, per poi spedire quest'informazione al processore P0 mediante la procedure *MPI_Reduce*, tramite la quale il processore *root* (ovvero P0) provvederà ad estrapolare, tra i *tdiff* ricevuti, il massimo (memorizzato in *timemax*). Nella misura non ricade né il prelievo e controllo dei dati (con relativa distribuzione), né la stampa dei risultati. Ogni caso di misura è stato strutturato nel seguente modo:

*nproc processori, dimensione n del problema, timemax misurato*

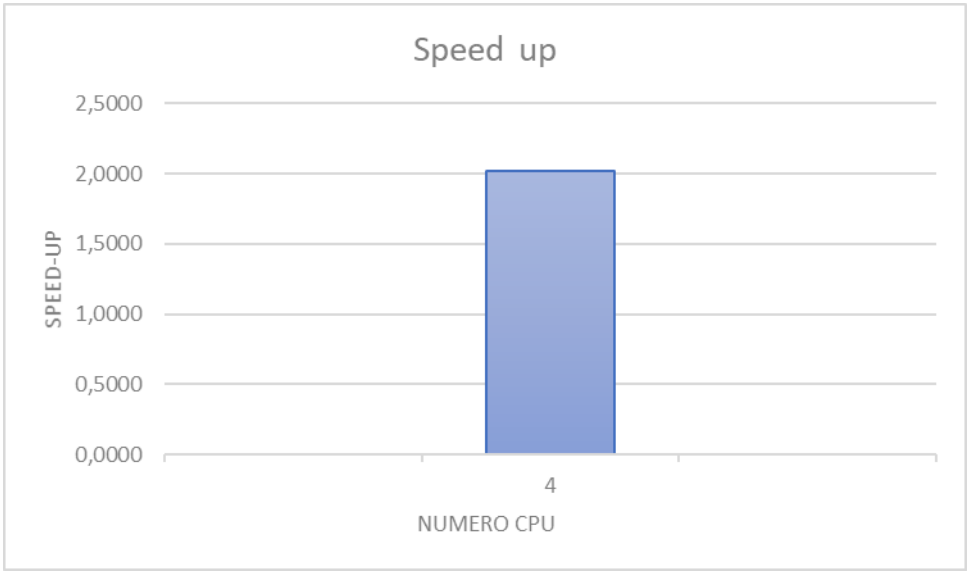
Ogni caso è stato misurato tre volte e la misura riportata è frutto di una media aritmetica delle tre misure. Il numero di processori coinvolti è stato: 1 e 4. Non sono state considerate ulteriori soluzioni multiprocessore coerentemente con la necessità di avere una griglia quadrata e con la disponibilità di massimo 8 processori (nel caso ne avessimo avuti 9, avremmo potuto ancora costruire una griglia quadrata). Dunque, l'unica soluzione multiprocessore possibile nel nostro caso è quella con 4 processori. Nel caso di un solo processore, è stato eseguito lo stesso algoritmo, non prevedendo un algoritmo puramente sequenziale. Il numero di casi totali previsti è stato di cinque: *8x8, 16x16, 32x32, 64x64, 256x256*.

**Dimensione matrice 8x8**

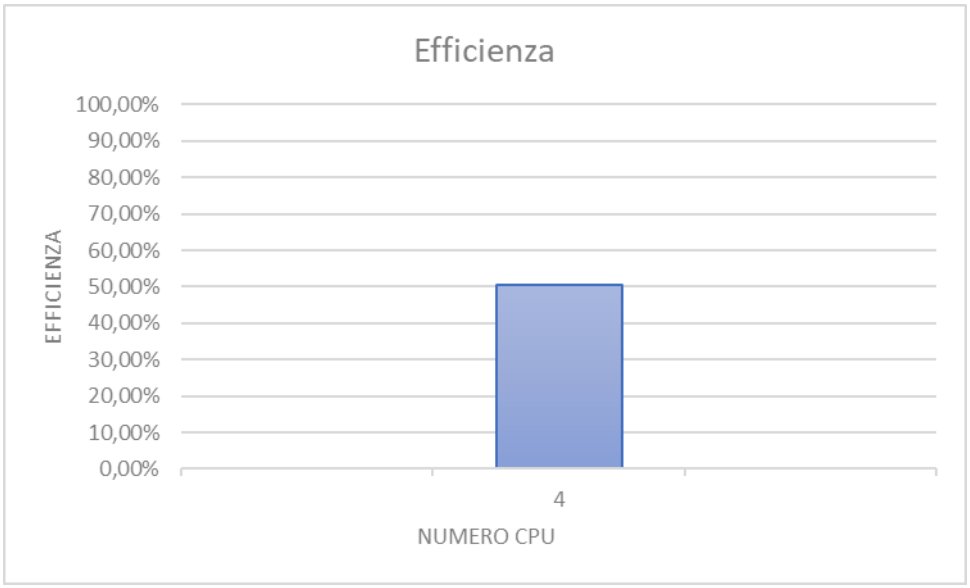
| #proc | Timemax(s) |
|---|---|
| 1 | 1,23024E-04 |
| 4 | 6,10351E-05 |



| #proc | Speed up (effettivo) | Speed up (ideale) |
|---|---|---|
| 4 | 2,0156 | 4 |

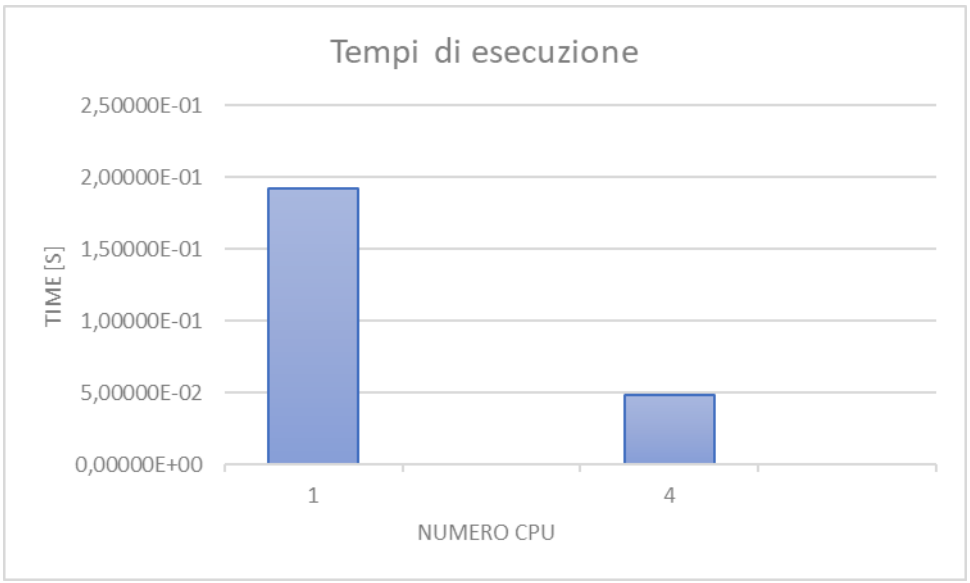


| #proc | Efficienza |
|---|---|
| 4 | 50,39% |

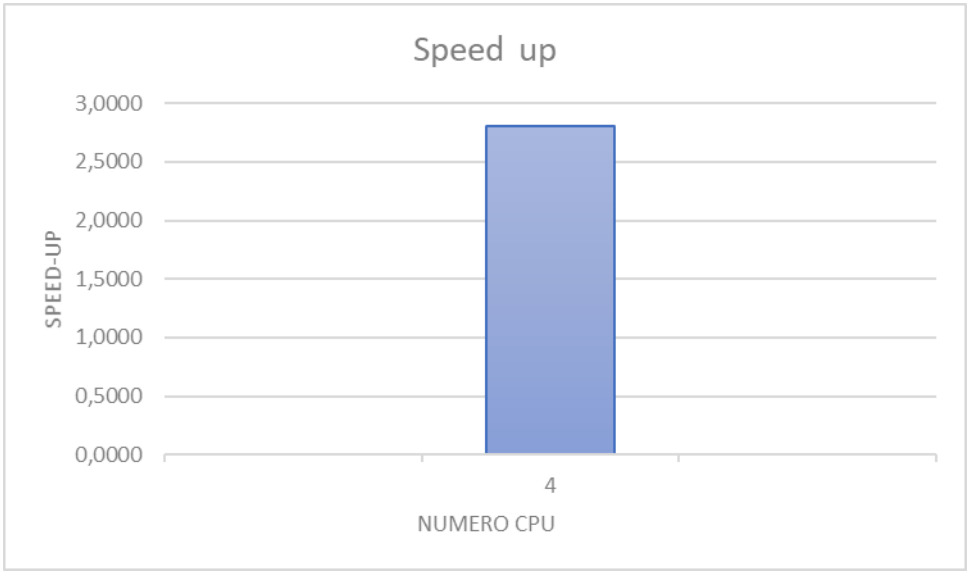


**Dimensione matrice 16x16**

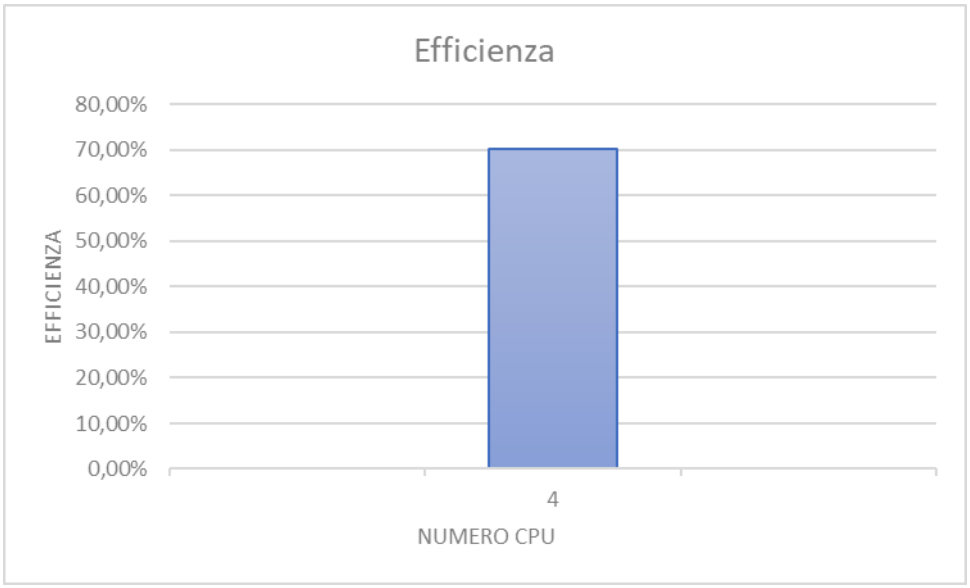
| #proc | Timemax(s) |
|---|---|
| 1 | 1,83468E-04 |
| 4 | 6,53883E-05 |



| #proc | Speed up (effettivo) | Speed up (ideale) |
|---|---|---|
| 4 | 2.8058 | 4 |

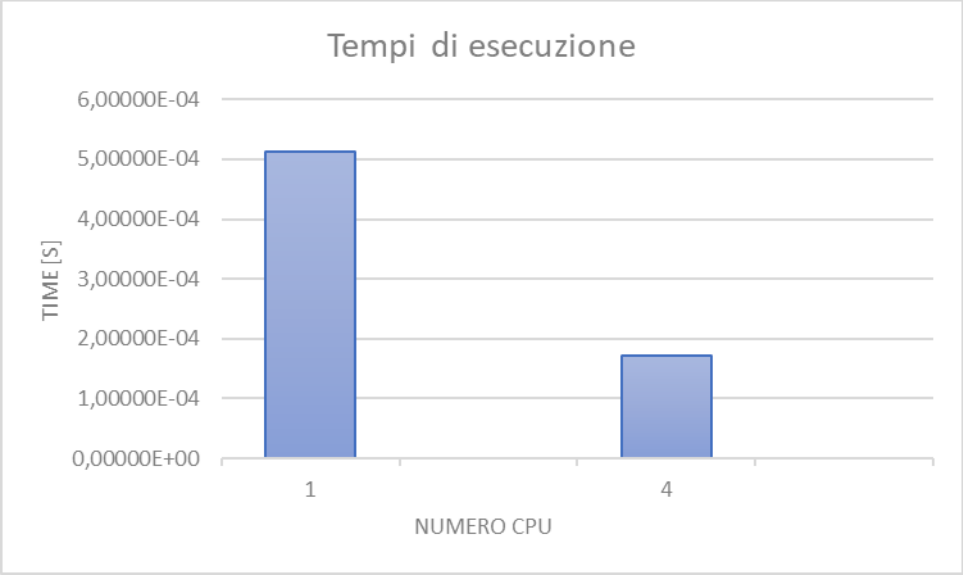


| #proc | Efficienza |
|---|---|
| 4 | 70,15% |

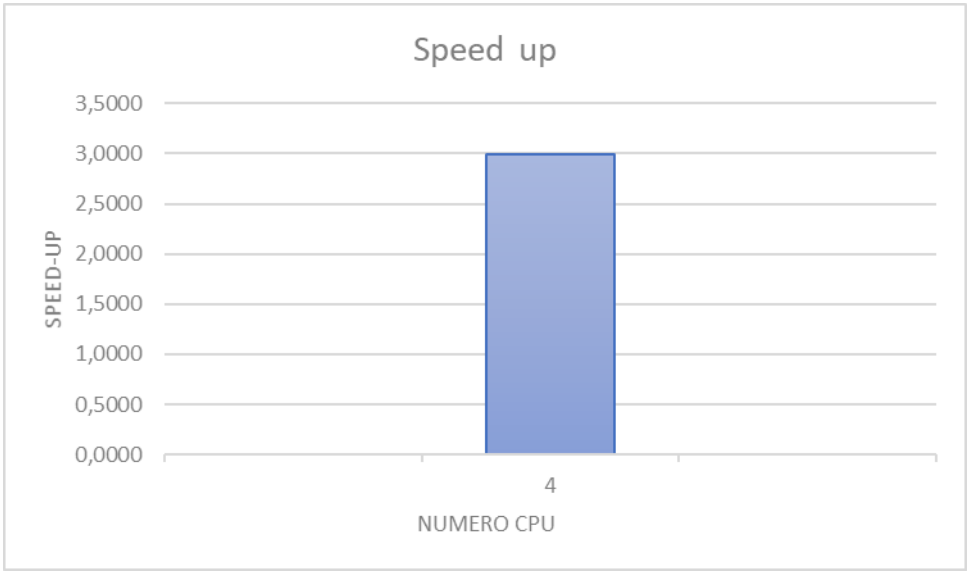


**Dimensione matrice 32x32**

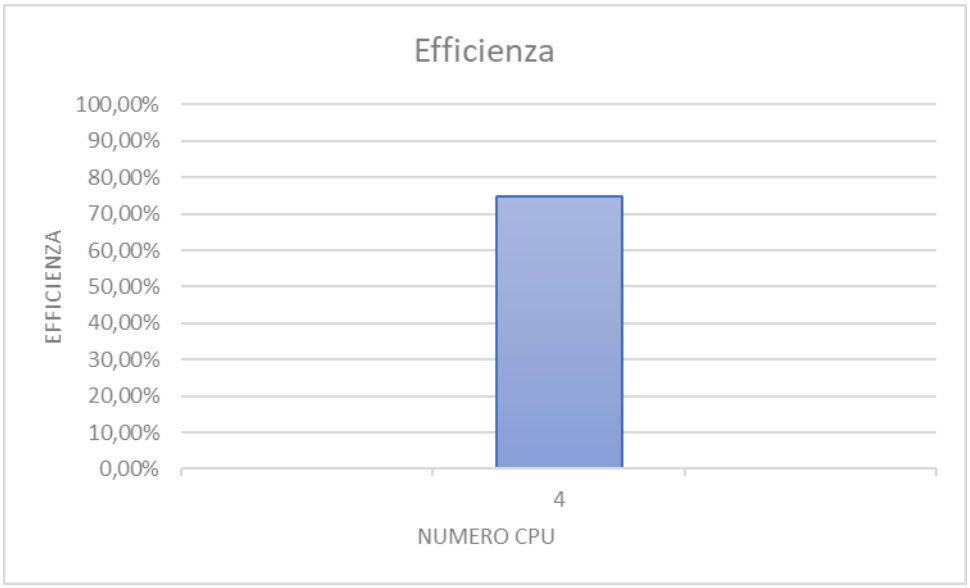
| #proc | Timemax(s) |
|---|---|
| 1 | 5,13079E-04 |
| 4 | 1,71222E-04 |



| #proc | Speed up (effettivo) | Speed up (ideale) |
|---|---|---|
| 4 | 2,9966 | 4 |

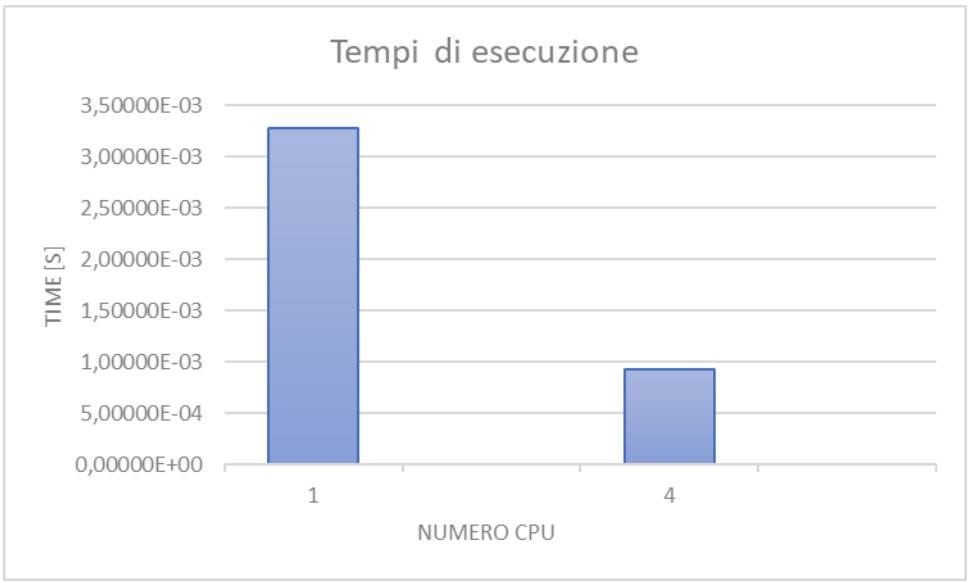


| #proc | Efficienza |
|---|---|
| 4 | 74,91% |

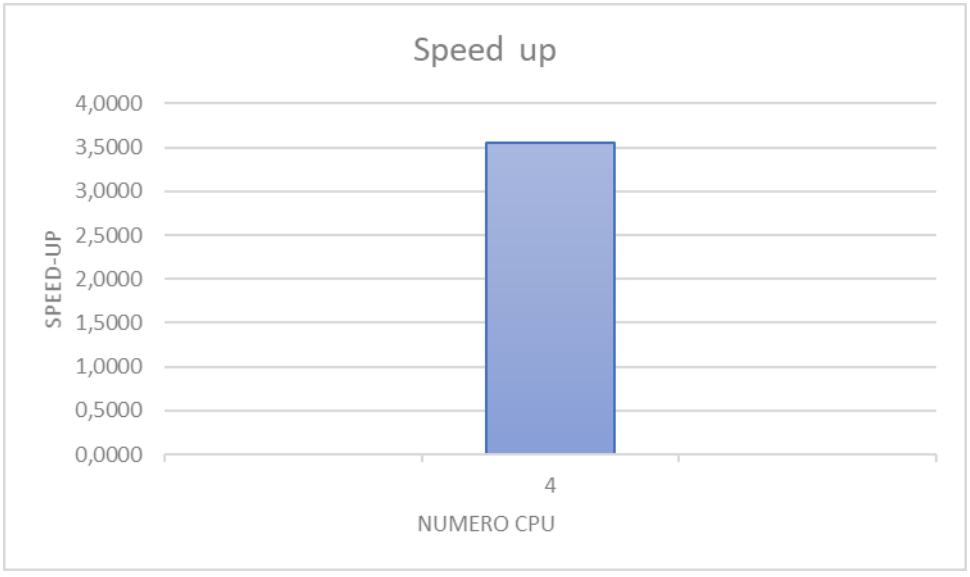


**Dimensione matrice 64x64**

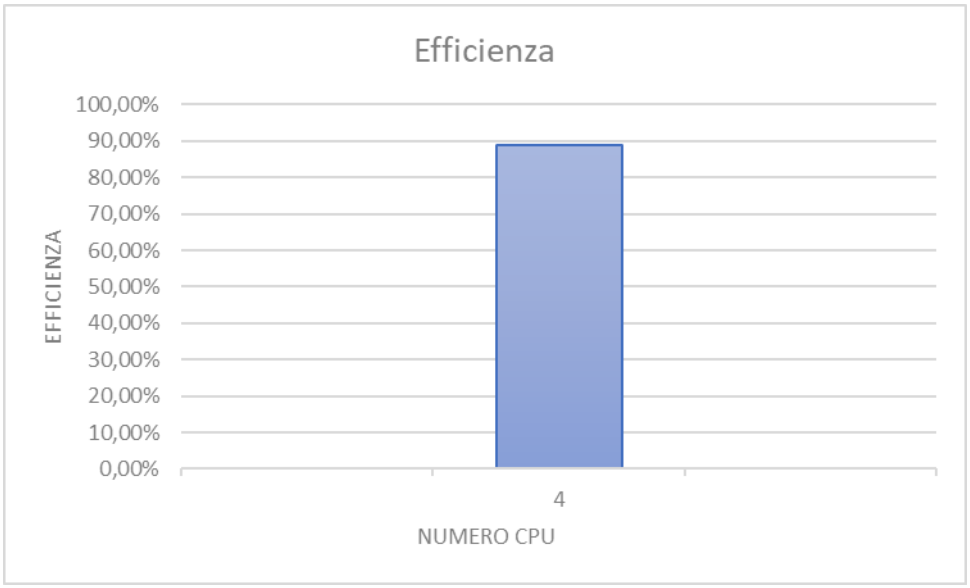
| #proc | Timemax(s) |
|---|---|
| 1 | 3,27288E-03 |
| 4 | 9,21002E-04 |



| #proc | Speed up (effettivo) | Speed up (ideale) |
|---|---|---|
| 4 | 3,5536 | 4 |

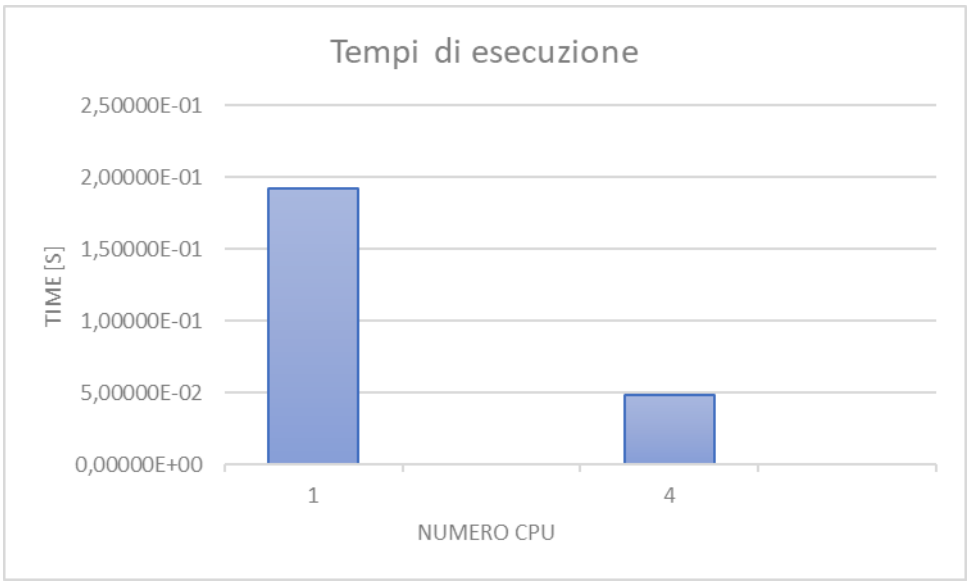


| #proc | Efficienza |
|---|---|
| 4 | 88,84% |

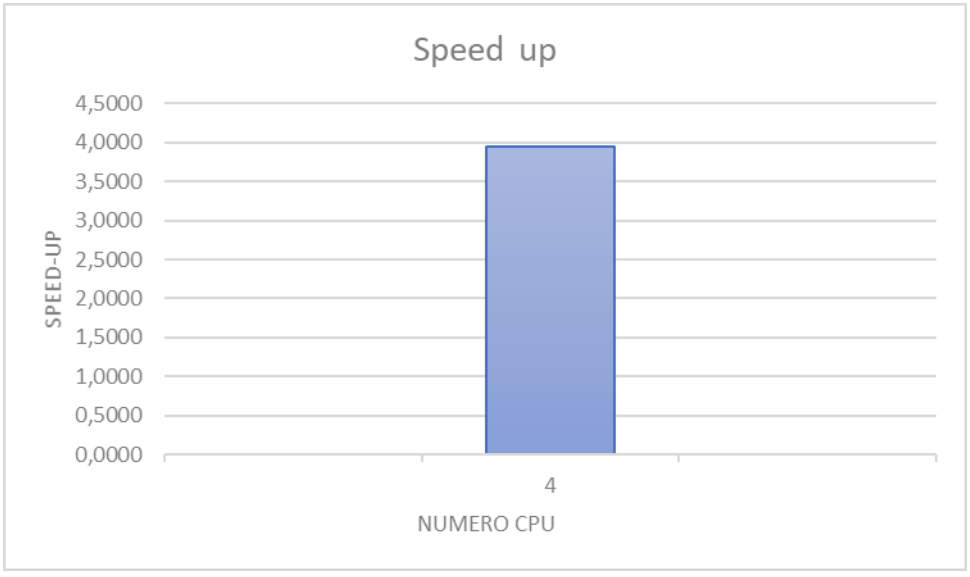


**Dimensione matrice 256x256**

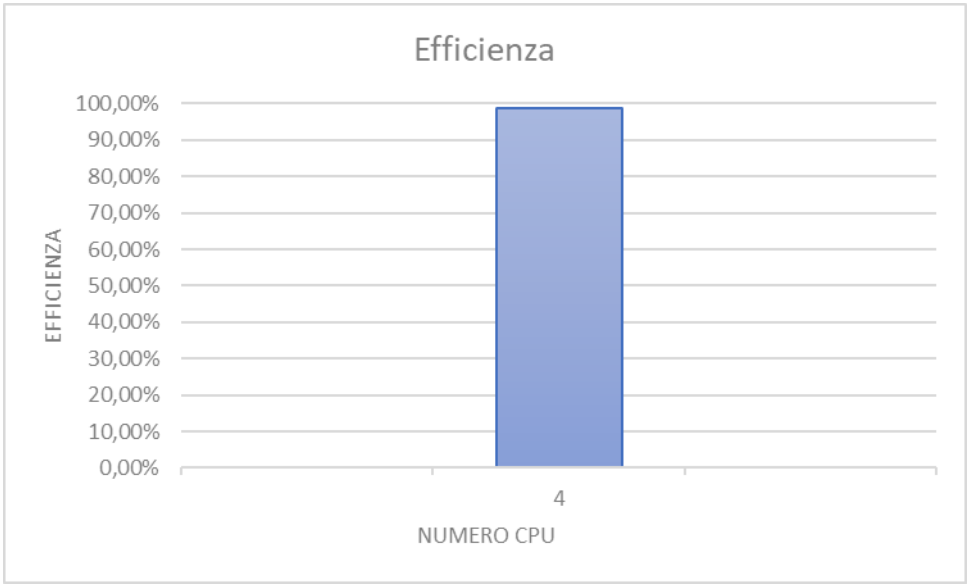
| #proc | Timemax(s) |
|---|---|
| 1 | 1,92163E-01 |
| 4 | 4,86556E-02 |



| #proc | Speed up (effettivo) | Speed up (ideale) |
|---|---|---|
| 4 | 3,9495 | 4 |

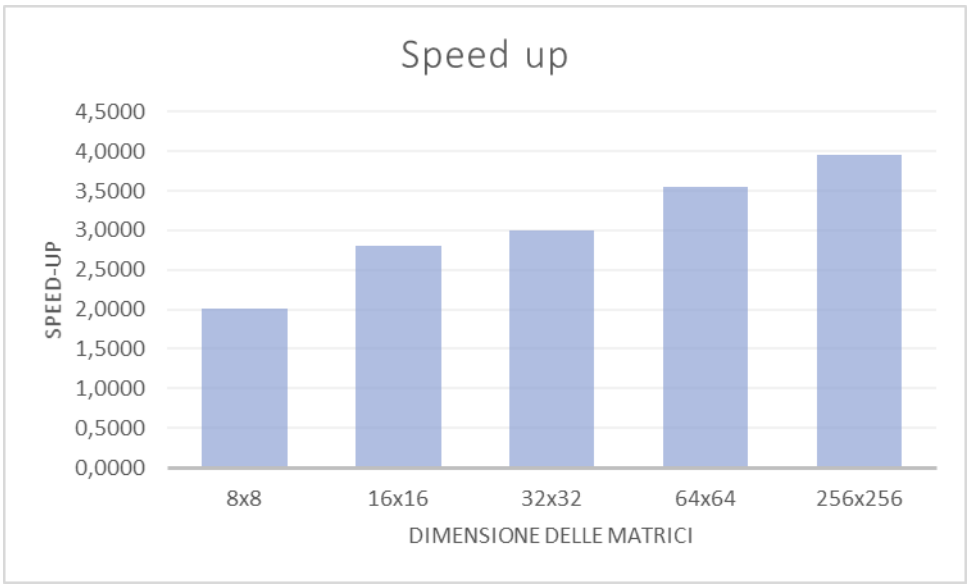


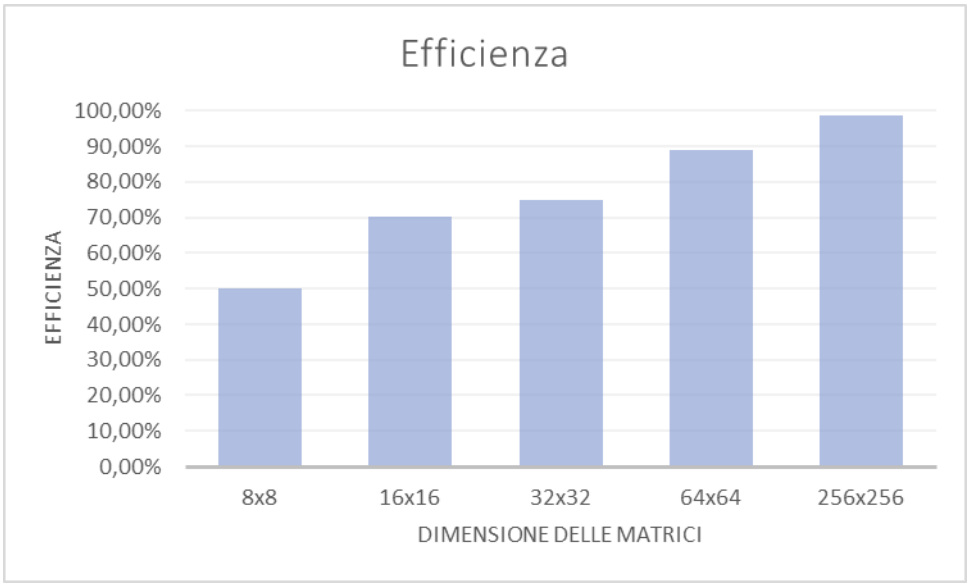
| #proc | Efficienza |
|---|---|
| 4 | 98,74% |



**Conclusioni**

Di seguito viene riportato un rapido confronto dei parametri al variare della dimensione del problema, considerando il caso di 4 CPU:



Già a partire da matrici di dimensioni *8x8* lo speed up è superiore a 2 (su uno speed up ideale di 4) e l'efficienza è dunque pari al 50%. Aumentando la dimensione del problema, ovvero con matrici *16x16*, guadagniamo circa 20 punti percentuali sull'efficienza, fino a raggiungere quasi il 90% con matrici *64x64* e superare il 98% con matrici *256x256*. Ciò non ci sorprende, coerentemente con quanto visto con la legge di Ware-Amdahl: aumentando la dimensione del problema, la frazione sequenziale $\alpha$ dell'algoritmo parallelo si riduce, a favore della frazione parallela (1-$\alpha$, considerando la legge non generalizzata). Ciò comporta un incremento dello speed up, essendo questi asintoticamente pari allo speed up ideale quando $\alpha$ tende a 0. Tuttavia, è bene ricordare che la legge di Ware-Amdahl non tiene conto dell'overhead di comunicazione, fattore che è certamente presente nelle misure.

# Esempi d’uso

Sono stati riportati cinque esempi d’uso: due casi ordinari e i tre casi di errore. I rispettivi file *.pbs* e *.out* sono riportati nelle cartelle *Esempi d’suo* e *Errori*. Di seguito vengono riportati sia i dati di input che i risultati ottenuti da terminale.

1. Input: *dimensione griglia p = 1, dimensione matrici m_input = 16. 1 CPU*

   Output:

```
…/mpiexec -machinefile  E -np 1 /homes/DMA/PDC/2020/BSLRRT94K/es3/matmat
[P0] dimensioni griglia: 1x1.
[P0] Dimensione in input: 16.
[P0] Matrice A (16x16):
4 8 5 3 3 5 5 2 5 2 8 6 7 5 3 4
8 8 3 4 9 8 9 9 6 2 7 8 2 3 8 5
8 3 8 2 5 1 1 7 9 6 1 7 1 1 1 7
8 1 8 5 6 8 5 1 7 2 6 9 3 4 2 1
5 9 9 7 8 7 5 7 3 5 4 2 6 2 6 2
2 4 7 6 9 9 6 7 1 2 4 3 6 5 2 1
3 1 7 1 8 9 5 8 3 7 9 6 8 3 7 8
7 2 4 6 1 7 1 9 8 4 2 2 9 3 9 2
4 5 9 9 4 4 7 6 1 5 2 7 7 7 3 4
8 6 8 9 1 8 8 6 3 7 8 9 8 5 1 9
9 9 8 2 1 3 7 9 7 7 6 3 4 8 6 9
2 2 6 2 1 2 5 1 9 3 9 5 6 9 4 5
6 9 6 4 2 2 3 7 8 8 9 9 4 3 6 5
5 3 4 5 4 9 3 1 9 2 5 5 8 6 8 2
5 4 5 5 3 8 2 8 4 1 7 8 3 2 1 7
4 3 9 5 2 2 6 8 1 8 2 6 5 9 7 7
[P0] Matrice B (16x16):
1 3 2 4 8 1 2 2 1 7 7 2 8 8 6 9
1 6 4 9 5 7 5 4 4 6 9 6 5 7 3 4
7 3 7 3 3 6 4 2 3 2 3 8 7 8 7 7
2 9 4 5 4 8 8 7 5 7 1 7 2 4 1 6
```

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

[P0] Il mio primo blocco A:
4 8 5 3 3 5 5 2 5 2 8 6 7 5 3 4
8 8 3 4 9 8 9 9 6 2 7 8 2 3 8 5
8 3 8 2 5 1 1 7 9 6 1 7 1 1 1 7
8 1 8 5 6 8 5 1 7 2 6 9 3 4 2 1
5 9 9 7 8 7 5 7 3 5 4 2 6 2 6 2
2 4 7 6 9 9 6 7 1 2 4 3 6 5 2 1
3 1 7 1 8 9 5 8 3 7 9 6 8 3 7 8
7 2 4 6 1 7 1 9 8 4 2 2 9 3 9 2
4 5 9 9 4 4 7 6 1 5 2 7 7 7 3 4
8 6 8 9 1 8 8 6 3 7 8 9 8 5 1 9
9 9 8 2 1 3 7 9 7 7 6 3 4 8 6 9
2 2 6 2 1 2 5 1 9 3 9 5 6 9 4 5
6 9 6 4 2 2 3 7 8 8 9 9 4 3 6 5
5 3 4 5 4 9 3 1 9 2 5 5 8 6 8 2
5 4 5 5 3 8 2 8 4 1 7 8 3 2 1 7
4 3 9 5 2 2 6 8 1 8 2 6 5 9 7 7
[P0] Il mio primo blocco B:
1 3 2 4 8 1 2 2 1 7 7 2 8 8 6 9
1 6 4 9 5 7 5 4 4 6 9 6 5 7 3 4
7 3 7 3 3 6 4 2 3 2 3 8 7 8 7 7
2 9 4 5 4 8 8 7 5 7 1 7 2 4 1 6
4 5 8 6 2 1 5 2 2 7 1 8 6 5 3 5
2 4 9 6 2 7 1 4 3 2 2 2 3 9 6 6
5 4 9 6 4 5 7 3 9 5 8 3 8 1 8 9
4 7 3 4 3 4 7 3 5 6 2 5 4 7 1 8
2 7 2 3 2 6 6 2 2 4 4 9 3 9 6 4
7 9 7 7 3 3 7 5 6 8 7 9 6 7 7 5
2 6 5 4 3 1 3 2 3 4 8 3 4 4 6 8
1 2 3 1 2 9 3 7 7 9 7 1 6 2 3 8
6 8 9 8 6 9 9 8 3 7 8 4 8 5 2 9
4 4 7 5 1 1 2 6 9 6 4 4 6 7 9 2
5 8 7 8 5 6 7 7 1 5 9 6 7 1 3 1
3 1 3 1 1 3 6 9 8 1 3 2 5 2 1 7
```

```
[P0] Il mio blocco risultato C:
243 388 420 380 259 381 365 339 331 395 430 343 416 403 349 473
305 508 532 493 341 465 484 413 419 533 521 455 540 510 429 616
231 330 301 271 220 311 341 279 286 364 315 365 373 409 295 439
240 366 421 337 253 375 329 303 309 407 372 355 417 435 387 495
321 496 513 481 318 441 461 366 340 465 435 471 478 496 373 532
275 399 472 393 240 367 370 311 323 393 326 369 400 419 332 471
360 487 561 454 292 431 474 426 400 470 472 437 521 481 408 597
275 465 412 401 291 408 420 357 273 409 387 392 408 449 312 457
318 446 489 419 287 442 446 407 415 471 416 413 476 441 373 537
352 527 565 486 358 529 522 497 506 551 544 456 569 553 467 708
350 506 506 479 342 432 497 431 455 499 542 479 556 551 468 599
256 369 395 325 215 331 350 327 337 356 398 344 394 376 372 424
300 502 446 438 315 448 467 404 391 505 525 460 492 494 401 560
264 440 461 410 273 418 388 368 303 414 418 386 427 443 372 456
213 352 362 310 229 366 339 328 324 368 339 309 367 404 294 493
344 442 479 408 275 400 449 421 429 456 435 416 491 429 386 508
[TIMEMAX][P0] Tempo massimo impiegato: 2.601147e-04 s
```

2. Input: *dimensione griglia p = 2, dimensione matrici m_input = 16. 4 CPU*

   Output:

```
.../mpiexec -machinefile  E -np 4
/homes/DMA/PDC/2020/BSLRRT94K/es3/matmat
[P0] dimensioni griglia: 2x2.
[P0] Dimensione in input: 16.
[P0] Matrice A (16x16):
2 7 9 2 1 5 5 6 5 2 7 8 4 4 7 9
3 3 9 6 8 3 4 5 6 7 1 2 7 5 4 6
9 3 5 7 5 1 3 1 2 9 6 3 1 1 2 3
3 1 8 1 1 1 3 7 7 4 6 2 8 8 5 5
1 1 2 4 1 4 2 9 1 7 2 1 6 4 1 8
2 7 7 3 7 9 7 4 1 1 5 6 8 9 1 7
7 2 1 7 5 2 6 3 6 6 3 2 7 2 1 8
8 5 1 5 4 5 6 3 6 8 8 4 5 9 1 3
1 8 7 4 9 2 6 4 7 7 5 4 8 3 2 4
7 1 6 9 3 2 2 8 9 9 1 2 6 8 4 5
7 2 8 4 3 2 7 7 6 1 1 4 3 9 5 1
7 1 9 1 9 1 6 6 7 6 8 4 3 2 8 7
1 4 1 1 5 6 7 2 6 5 3 8 3 6 6 9
6 5 7 4 3 2 9 8 7 7 9 7 9 5 4 7
8 3 6 1 6 3 9 2 6 1 7 8 6 4 5 9
6 1 3 7 2 3 3 6 9 2 4 6 6 5 2 2
```

```
[P0] Matrice B (16x16):
7 7 9 1 7 9 2 1 7 7 6 3 8 2 9 2
9 1 8 1 3 1 6 9 9 7 4 3 3 5 4 7
9 4 8 6 1 7 7 7 2 3 7 9 2 6 9 8
6 7 8 6 6 3 6 3 7 7 6 7 2 7 3 8
1 8 2 1 5 8 7 5 9 4 2 8 8 1 7 4
6 3 8 9 3 4 2 1 1 5 7 9 3 9 5 3
7 6 2 1 3 6 5 9 8 6 7 6 5 2 7 1
3 5 9 5 6 8 5 4 4 1 3 6 9 5 6 5
2 7 5 4 4 7 3 2 3 7 5 5 7 2 5 9
7 2 2 3 1 7 7 4 7 7 7 4 3 3 8 2
1 3 5 2 9 5 1 1 2 5 3 6 4 8 3 8
7 5 9 7 9 4 8 4 1 3 7 1 6 4 2 4
4 4 3 2 8 3 9 8 5 2 4 8 7 7 7 5
9 6 9 8 9 6 2 7 8 6 6 2 7 5 5 2
9 5 3 5 7 2 1 9 3 5 8 1 9 3 3 6
6 2 4 3 7 3 7 4 6 3 5 4 6 8 3 3

[P0] Il mio primo blocco A:
2 7 9 2 1 5 5 6
3 3 9 6 8 3 4 5
9 3 5 7 5 1 3 1
3 1 8 1 1 1 3 7
1 1 2 4 1 4 2 9
2 7 7 3 7 9 7 4
7 2 1 7 5 2 6 3
8 5 1 5 4 5 6 3
[P0] Il mio primo blocco B:
7 7 9 1 7 9 2 1
9 1 8 1 3 1 6 9
9 4 8 6 1 7 7 7
6 7 8 6 6 3 6 3
1 8 2 1 5 8 7 5
6 3 8 9 3 4 2 1
7 6 2 1 3 6 5 9
3 5 9 5 6 8 5 4

[P1] Il mio primo blocco A:
5 2 7 8 4 4 7 9
6 7 1 2 7 5 4 6
2 9 6 3 1 1 2 3
7 4 6 2 8 8 5 5
1 7 2 1 6 4 1 8
1 1 5 6 8 9 1 7
6 6 3 2 7 2 1 8
6 8 8 4 5 9 1 3
```

```
[P1] Il mio primo blocco B:
7 7 6 3 8 2 9 2
9 7 4 3 3 5 4 7
2 3 7 9 2 6 9 8
7 7 6 7 2 7 3 8
9 4 2 8 8 1 7 4
1 5 7 9 3 9 5 3
8 6 7 6 5 2 7 1
4 1 3 6 9 5 6 5

[P2] Il mio primo blocco A:
1 8 7 4 9 2 6 4
7 1 6 9 3 2 2 8
7 2 8 4 3 2 7 7
7 1 9 1 9 1 6 6
1 4 1 1 5 6 7 2
6 5 7 4 3 2 9 8
8 3 6 1 6 3 9 2
6 1 3 7 2 3 3 6
[P2] Il mio primo blocco B:
2 7 5 4 4 7 3 2
7 2 2 3 1 7 7 4
1 3 5 2 9 5 1 1
7 5 9 7 9 4 8 4
4 4 3 2 8 3 9 8
9 6 9 8 9 6 2 7
9 5 3 5 7 2 1 9
6 2 4 3 7 3 7 4

[P3] Il mio primo blocco A:
7 7 5 4 8 3 2 4
9 9 1 2 6 8 4 5
6 1 1 4 3 9 5 1
7 6 8 4 3 2 8 7
6 5 3 8 3 6 6 9
7 7 9 7 9 5 4 7
6 1 7 8 6 4 5 9
9 2 4 6 6 5 2 2
[P3] Il mio primo blocco B:
3 7 5 5 7 2 5 9
7 7 7 4 3 3 8 2
2 5 3 6 4 8 3 8
1 3 7 1 6 4 2 4
5 2 4 8 7 7 7 5
8 6 6 2 7 5 5 2
3 5 8 1 9 3 3 6
6 3 5 4 6 8 3 3
```

```
[P0] Il mio blocco risultato C:
510 347 509 354 464 393 407 429
455 383 433 313 395 432 433 410
355 292 349 203 312 351 303 252
398 325 408 292 407 385 332 361
301 222 311 231 298 282 297 252
500 374 512 357 472 413 438 445
360 335 357 222 377 367 362 304
457 385 480 307 462 445 365 358
[P1] Il mio blocco risultato C:
365 371 462 407 451 438 410 429
429 371 425 446 436 368 464 388
344 335 337 312 305 270 359 290
327 310 375 362 418 350 396 358
283 227 282 297 303 305 296 231
439 377 443 470 447 453 450 375
395 344 357 364 385 317 383 306
451 437 434 399 442 381 447 359
[P2] Il mio blocco risultato C:
438 371 426 277 404 423 455 433
477 415 497 351 442 469 407 375
440 378 456 307 388 409 316 370
456 417 445 295 458 495 405 406
434 329 391 305 413 353 359 372
556 452 563 362 557 527 504 492
477 407 468 298 497 445 411 414
356 354 430 289 403 370 319 284
[P3] Il mio blocco risultato C:
446 389 410 453 431 371 458 417
440 411 454 420 471 378 474 401
363 340 399 348 422 298 410 330
417 395 452 440 510 363 488 419
369 354 413 333 423 342 354 311
496 463 529 506 556 477 544 479
417 393 457 416 497 387 452 384
318 327 360 345 393 318 356 348
[TIMEMAX][P0] Tempo massimo impiegato: 7.104874e-05 s
```

3. Input: *dimensione griglia p = 0, dimensione matrici m_input = 0. 4 CPU*

   Output:

```
.../mpiexec -machinefile  E -np 4
/homes/DMA/PDC/2020/BSLRRT94K/es3/matmat
[E0] Dimensione griglia 0 non valida. Valore default: 2
[P0] dimensioni griglia: 2x2.
[E2] Dimensione input 0 non valido. Valore default: 16.
[P0] Dimensione in input: 16.
[P0] Matrice A (16x16):
...
```

   *(Non riporto il resto dell'output dal momento che si tratta di un calcolo già visto. Il continuo è presente nel file matmat_E0_E2.out).*

4. Input: *dimensione griglia p = 2, dimensione matrici m_input = 16. 1 CPU*

   Output:

```
.../mpiexec -machinefile  E -np 1
/homes/DMA/PDC/2020/BSLRRT94K/es3/matmat

[E1] p e q sono diversi: griglia non quadrata. Terminazione...
```

5. Input: *dimensione griglia p = 2, dimensione matrici m_input = 17. 4 CPU*

   Output:

```
.../mpiexec -machinefile  E -np 4
/homes/DMA/PDC/2020/BSLRRT94K/es3/matmat
[P0] dimensioni griglia: 2x2.
[P0] Dimensione in input: 17.
[E3] Dimensione input 17 non e' multiplo di p = 2. Valore di default:
p*10.
[P0] Matrice A (20x20):
3 5 5 7 8 3 9 8 6 2 7 1 5 8 2 7 5 3 8 3
9 7 6 3 1 1 4 6 9 8 8 2 3 4 6 8 4 5 4 7
4 9 7 6 5 6 4 1 6 9 3 5 6 6 5 4 6 6 1 5
4 6 5 4 7 1 3 8 4 4 5 7 1 9 4 6 4 5 6 9
4 6 3 8 2 7 2 8 2 9 1 5 4 5 7 1 4 7 9 7
1 4 4 2 2 5 5 5 9 8 2 2 4 4 9 3 9 8 8 1
6 9 3 9 2 9 9 3 6 6 9 5 8 2 4 9 6 8 2 4
6 3 5 7 5 2 9 4 7 6 2 3 3 4 9 4 2 9 7 5
3 4 9 1 5 3 7 9 1 6 3 4 9 5 1 4 4 8 5 1
```

```
4 4 2 6 7 1 7 6 7 2 2 1 6 1 8 1 2 4 7 9
9 7 4 6 2 2 9 3 9 2 4 3 5 5 6 1 3 4 4 1
5 5 8 1 4 6 9 5 7 4 4 7 2 5 3 1 7 1 4 6
9 5 7 5 7 1 3 9 4 6 9 7 9 7 5 3 2 2 5 8
6 6 3 5 2 4 5 6 4 6 9 1 1 6 3 7 5 5 5 6
2 4 3 8 9 6 1 8 7 3 4 1 8 7 5 7 8 1 3 9
4 3 9 5 6 3 9 1 7 4 7 6 8 9 4 5 3 2 1 1
4 4 1 3 8 4 9 6 2 3 5 5 5 5 9 1 5 9 9 2
3 6 6 8 3 9 1 6 9 8 6 3 3 4 3 8 7 3 3 6
3 7 2 5 9 8 3 4 7 2 4 8 7 9 4 8 6 3 4 5
1 7 6 1 1 8 9 6 8 9 2 1 6 1 3 4 9 6 5 4
[P0] Matrice B (20x20):
7 6 9 3 3 2 1 7 4 2 9 2 8 5 3 6 4 9 9 2
8 9 1 2 9 3 5 8 6 8 1 4 2 9 4 5 1 2 2 2
1 8 4 8 4 6 4 5 5 2 4 1 1 4 2 9 5 5 6 1
1 6 2 2 3 3 6 2 2 5 3 2 4 6 9 5 9 2 9 2
3 1 2 3 5 4 9 7 6 5 7 6 8 7 7 9 9 2 1 2
4 3 3 7 7 1 2 6 9 8 8 9 9 7 2 2 8 8 8 4
4 3 7 9 7 3 8 7 2 6 6 5 7 6 1 4 4 9 7 3
8 3 2 5 1 1 6 6 9 2 8 1 2 5 7 9 5 4 4 6
9 9 2 6 3 2 9 7 1 5 9 6 7 9 1 7 9 4 4 6
3 9 4 5 4 2 2 9 3 3 5 2 9 4 6 2 5 5 8 4
7 6 9 5 3 8 9 9 2 1 6 5 9 7 7 4 8 6 1 1
8 3 1 7 7 6 6 9 8 5 3 6 8 1 1 1 8 9 9 9
1 3 2 7 1 6 1 6 9 8 5 7 2 5 2 6 1 6 5 6
8 6 2 6 6 9 4 4 7 3 1 5 4 1 2 4 6 1 9 6
8 4 1 7 6 1 3 4 4 6 1 2 2 9 7 5 9 8 6 4
1 6 8 2 4 7 5 1 5 5 4 4 6 2 8 3 9 9 6 3
5 4 4 4 4 8 8 1 6 4 4 5 7 9 6 2 4 9 9 9
2 1 1 7 9 8 7 7 5 2 9 9 5 2 1 8 7 6 8 2
9 2 4 5 8 7 6 3 6 3 2 7 3 2 3 9 7 9 6 3
1 6 9 4 5 1 2 2 6 1 3 4 9 4 8 8 2 2 8 7

[P0] Il mio primo blocco A:
3 5 5 7 8 3 9 8 6 2
9 7 6 3 1 1 4 6 9 8
4 9 7 6 5 6 4 1 6 9
4 6 5 4 7 1 3 8 4 4
4 6 3 8 2 7 2 8 2 9
1 4 4 2 2 5 5 5 9 8
6 9 3 9 2 9 9 3 6 6
6 3 5 7 5 2 9 4 7 6
3 4 9 1 5 3 7 9 1 6
4 4 2 6 7 1 7 6 7 2
```

```
[P0] Il mio primo blocco B:
7 6 9 3 3 2 1 7 4 2
8 9 1 2 9 3 5 8 6 8
1 8 4 8 4 6 4 5 5 2
1 6 2 2 3 3 6 2 2 5
3 1 2 3 5 4 9 7 6 5
4 3 3 7 7 1 2 6 9 8
4 3 7 9 7 3 8 7 2 6
8 3 2 5 1 1 6 6 9 2
9 9 2 6 3 2 9 7 1 5
3 9 4 5 4 2 2 9 3 3

[P1] Il mio primo blocco A:
7 1 5 8 2 7 5 3 8 3
8 2 3 4 6 8 4 5 4 7
3 5 6 6 5 4 6 6 1 5
5 7 1 9 4 6 4 5 6 9
1 5 4 5 7 1 4 7 9 7
2 2 4 4 9 3 9 8 8 1
9 5 8 2 4 9 6 8 2 4
2 3 3 4 9 4 2 9 7 5
3 4 9 5 1 4 4 8 5 1
2 1 6 1 8 1 2 4 7 9
[P1] Il mio primo blocco B:
9 2 8 5 3 6 4 9 9 2
1 4 2 9 4 5 1 2 2 2
4 1 1 4 2 9 5 5 6 1
3 2 4 6 9 5 9 2 9 2
7 6 8 7 7 9 9 2 1 2
8 9 9 7 2 2 8 8 8 4
6 5 7 6 1 4 4 9 7 3
8 1 2 5 7 9 5 4 4 6
9 6 7 9 1 7 9 4 4 6
5 2 9 4 6 2 5 5 8 4

[P2] Il mio primo blocco A:
9 7 4 6 2 2 9 3 9 2
5 5 8 1 4 6 9 5 7 4
9 5 7 5 7 1 3 9 4 6
6 6 3 5 2 4 5 6 4 6
2 4 3 8 9 6 1 8 7 3
4 3 9 5 6 3 9 1 7 4
4 4 1 3 8 4 9 6 2 3
3 6 6 8 3 9 1 6 9 8
3 7 2 5 9 8 3 4 7 2
1 7 6 1 1 8 9 6 8 9
```

```
[P2] Il mio primo blocco B:
7 6 9 5 3 8 9 9 2 1
8 3 1 7 7 6 6 9 8 5
1 3 2 7 1 6 1 6 9 8
8 6 2 6 6 9 4 4 7 3
8 4 1 7 6 1 3 4 4 6
1 6 8 2 4 7 5 1 5 5
5 4 4 4 4 8 8 1 6 4
2 1 1 7 9 8 7 7 5 2
9 2 4 5 8 7 6 3 6 3
1 6 9 4 5 1 2 2 6 1

[P3] Il mio primo blocco A:
4 3 5 5 6 1 3 4 4 1
4 7 2 5 3 1 7 1 4 6
9 7 9 7 5 3 2 2 5 8
9 1 1 6 3 7 5 5 5 6
4 1 8 7 5 7 8 1 3 9
7 6 8 9 4 5 3 2 1 1
5 5 5 5 9 1 5 9 9 2
6 3 3 4 3 8 7 3 3 6
4 8 7 9 4 8 6 3 4 5
2 1 6 1 3 4 9 6 5 4
[P3] Il mio primo blocco B:
6 5 9 7 7 4 8 6 1 1
3 6 8 1 1 1 8 9 9 9
5 7 2 5 2 6 1 6 5 6
1 5 4 1 2 4 6 1 9 6
1 2 2 9 7 5 9 8 6 4
4 4 6 2 8 3 9 9 6 3
4 5 7 9 6 2 4 9 9 9
9 9 5 2 1 8 7 6 8 2
2 7 3 2 3 9 7 9 6 3
3 4 9 4 8 8 2 2 8 7

[P0] Il mio blocco risultato C:
524 500 420 531 500 496 608 550 535 437
540 600 471 521 489 440 528 588 490 394
472 563 363 539 532 449 508 592 535 468
529 504 398 503 518 468 546 544 553 377
508 482 334 521 529 398 464 541 564 417
511 460 301 533 496 422 514 513 480 413
536 603 487 609 593 520 617 675 590 548
495 487 382 553 531 418 536 557 476 416
413 412 326 523 446 450 470 539 532 377
429 399 330 451 422 312 459 458 432 369
```

```
[P1] Il mio blocco risultato C:
515 481 557 549 474 607 642 572 605 401
530 427 594 560 481 580 610 601 636 407
490 476 585 564 436 527 597 558 665 425
462 445 566 482 477 585 614 530 628 437
462 454 533 501 457 557 590 560 683 430
470 460 499 530 379 510 588 586 605 414
613 575 698 655 515 576 705 718 744 457
511 457 550 524 430 594 639 594 659 383
480 428 455 427 347 535 500 537 561 365
429 390 459 495 407 555 506 467 510 353
[P2] Il mio blocco risultato C:
496 466 335 479 442 366 473 522 409 397
501 480 380 528 476 391 502 541 493 391
572 571 463 576 498 489 549 656 610 431
479 502 431 461 466 427 501 512 464 357
473 512 389 493 450 436 535 497 578 452
458 503 375 547 454 471 520 565 476 429
534 387 339 550 542 452 551 567 530 425
492 583 410 511 486 434 538 552 536 444
544 514 377 539 543 497 570 579 614 505
458 497 361 543 492 398 504 543 506 440
[P3] Il mio blocco risultato C:
453 395 468 510 329 491 517 518 552 339
468 425 554 512 356 497 533 556 596 420
546 468 617 564 517 653 639 603 670 464
469 415 556 493 455 504 565 536 586 361
498 469 575 583 537 585 626 521 622 468
469 442 536 501 370 499 593 546 591 383
489 498 533 518 405 560 610 606 603 395
527 466 614 580 502 537 654 573 656 438
509 544 620 564 466 552 672 593 658 484
510 468 540 553 377 508 526 591 607 418
[TIMEMAX][P0] Tempo massimo impiegato: 9.202957e-05 s
```