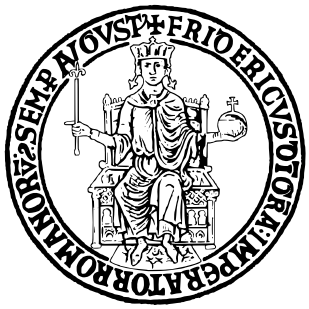
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Corso di Parallel and Distributed Computing

Calcolo della somma di N numeri

Relatore
Prof. Giuliano Laccetti
Prof.ssa Valeria Mele

Candidato
Fabrizio Vitale N97/0449
Giovanni Falcone N97/0451
Luigi Mangiacapra N97/0454

Anno Accademico 2023-2024

Indice

Capitolo 1

Descrizione del problema

Il goal del problema è calcolare la somma di N numeri in ambiente *parallelo* su architettura **MIMD** (**Multiple Instruction Multiple Data**) a memoria distribuita, utilizzando la libreria `MPI` in linguaggio C. In particolare, verranno adoperate 3 strategie differenti per effettuare tale somma e tramite dei grafici verranno mostrate le differenze tra le 3 strategie in termini di **tempo di esecuzione**, **speed up** ed **efficienza**.

L'algoritmo prende in input (da terminale) gli N numeri da sommare:

- se $N \leq 20$, allora verranno presi in input gli N valori forniti al momento del lancio del programma.
- se $N > 20$, gli elementi da sommare verranno generati randomicamente. Ovviamente se $N > 20$, i valori inseriti da terminali non verranno presi in considerazione.

Capitolo 2

Descrizione algoritmo

In questo capitolo descriveremo la struttura del programma, ovvero come abbiamo organizzato i vari file *.c*, *.h* e *.pbs*, le strategie che questo applicherà e le relative problematiche.

2.1 Struttura del programma

Innanzitutto descriviamo come abbiamo suddiviso i vari file e cosa ciascuno di essi contiene. La struttura del programma è così suddivisa:

```
/
├── Main.c
├── Strategy.c
├── Strategy.h
├── Utils.h
├── Utils.c
├── job-script.pbs
└── sum.pbs
```

dove

- `Main.c` è file principale che contiene il main del programma: richiama le funzioni della libreria `MPI` per le inizializzazioni e le opportune funzioni dai file header per applicare le diverse strategie.
- `Strategy.c` è il file che contiene le funzioni relative alle strategie da usare.
- `Strategy.h` è il file che contiene i diversi prototipi.
- `Utils.c` è il file che contiene le funzioni necessarie al controllo dei parametri e altre funzioni di utilità come il riempimento dell'array, della distribuzioni degli operandi, ecc
- `Utils.h` è il file che contiene i prototipi.
- `sum.pbs`: il pbs utilizzato per testare le funzionalità del programma (non "blocca" il cluster)

- `job-script.pbs`: il pbs utilizzato per raccogliere i tempi da analizzare (usato per "bloccare" il cluster)

2.2 Strategie

Definiamo, dunque, le strategie che vogliamo applicare.

2.2.1 Strategia I

Tramite questa strategia ciascun processore calcola la propria somma parziale e invia tale somma ad un processore prestabilito (nel nostro caso il processore P_0), il quale alla fine conterrà la somma totale.

Al passo 0 ciascun processore P_i effettua la propria somma locale S_i. Al passo 1 il processore P_1 invia la propria somma parziale al processore P_0 che provvede a fare la somma tra la propria somma parziale e quella inviata da P_1. E così via.

In generale all'$i-$esimo passo, il processore P_i invia la propria somma parziale al processore P_0 che provvede a fare la somma:

$$S_{0,i-1} = S_{0,i-1} + S_i$$

Uno schema di questa strategia viene mostrato in Figura 2.1 con $i = 4$.

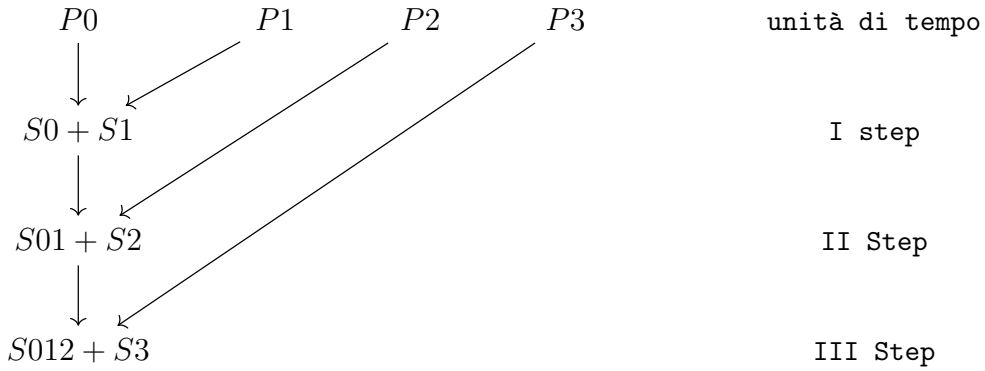


Figura 2.1: Schema funzionamento I strategia

L'algoritmo viene mostrato nel Listing 2.1. Il suo prototipo è descritto nella Sezione 3.2.1.

```
1 int first_strategy(int menum, int nproc, int sum){
2     int sum_parz = 0;
3     int tag;
4     MPI_Status status;
5
6     if(menum == 0){
7         for(int i = 1; i < nproc; i++){
8             tag = 80 + i;
9             MPI_Recv(&sum_parz, 1, MPI_INT, i, tag, MPI_COMM_WORLD,
```

```
10              &status);
11              sum += sum_parz;
12          }
13      }else{
14          tag = menum + 80;
15          MPI_Send(&sum, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
16      }
17
18      return sum;
19  }
```

Listing 2.1: Algoritmo strategia I

2.2.2 Strategia II

Con questa strategia ciascuna coppia di processori comunica tra loro la propria somma parziale. Anche in questo caso, la somma totale si trova in unico processore prestabilito (che nel nostro caso è sempre P_0). Inoltre, poichè le comunicazioni avvengono a coppie è necessario che il numero di processori sia una potenza di 2.

Uno schema di questa strategia viene mostrato in Figura 2.2 con $i = 4$.

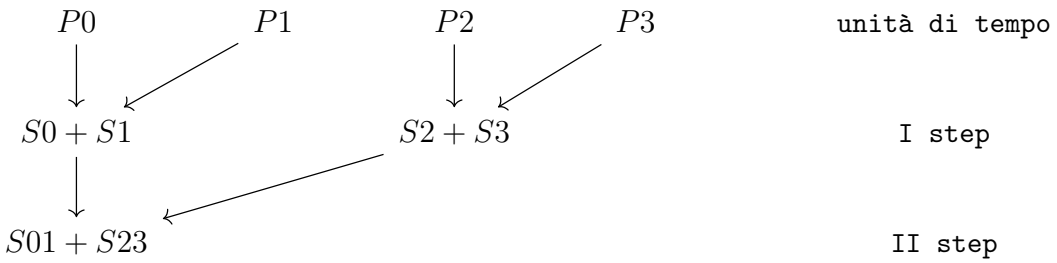


Figura 2.2: Schema funzionamento II strategia

L'algoritmo viene mostrato nel Listing 2.2. Il suo prototipo è descritto nella Sezione 3.2.2.

```
1  int second_strategy(int menum, int logNproc, int *array, int sum){
2      int sum_parz = 0;
3      int tag;
4      int partner;
5
6      int power_for_partecipation;
7      int does_processor_partecipate;
8
9      int power_for_communication;
10     int does_processor_receive;
11
12     MPI_Status status;
13
14     for(int i = 0; i < logNproc; i++){
15         power_for_partecipation =  array[i];
```

```
16          does_processor_partecipate = (menum % power_for_partecipation) ↩
        == 0;
17
18          if(does_processor_partecipate){
19              power_for_communication = array[i + 1];
20              does_processor_receive = (menum % power_for_communication) ↩
        == 0;
21
22              if (does_processor_receive){
23                  partner = menum + power_for_partecipation;
24                  tag = 60 + i;
25                  MPI_Recv(&sum_parz, 1, MPI_INT, partner, tag,
26                  MPI_COMM_WORLD, &status);
27                  sum += sum_parz;
28              } else{
29                  partner = menum - power_for_partecipation;
30                  tag = 60 + i;
31                  MPI_Send(&sum, 1, MPI_INT, partner, tag,
32                  MPI_COMM_WORLD);
33              }
34          }
35      }
36
37      return sum;
38      }
39  }
```

Listing 2.2: Algoritmo strategia II

2.2.3 Strategia III

La strategia III è identica alla II eccetto che tutte le coppie inviano e ricevano la propria somma parziale in modo che alla fine tutti i processori abbiano la somma totale.

Uno schema di questa strategia viene mostrato in Figura 2.3 con $i = 4$.

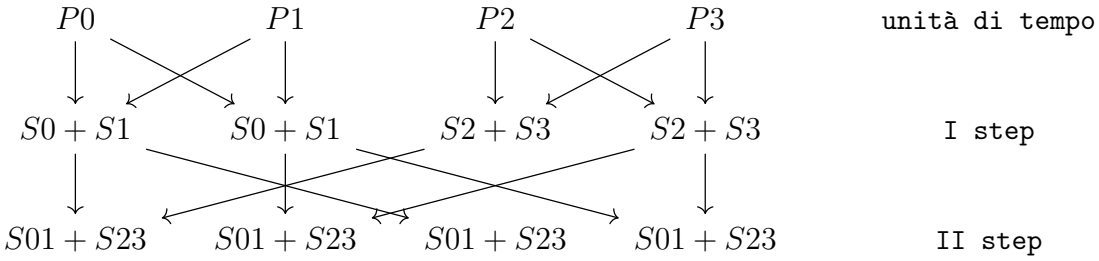


Figura 2.3: Schema funzionamento III strategia

L'algoritmo viene mostrato nel Listing 2.3. Il suo prototipo è descritto nella Sezione 3.2.3.

```
1  int third_strategy(int menum, int logNproc, int *array, int sum){
2      int partner;
```

```
3     int send_tag;
4     int recv_tag;
5     int sum_parz;
6     MPI_Status status;
7
8     sum_parz = 0;
9     for(int i = 0; i < logNproc; i++){
10        if ((menum % array[i + 1]) < array[i]) {
11            partner = menum + array[i];
12            send_tag = 40 + i;
13            recv_tag = 40 + i;
14
15            // Invia la somma locale al processo partner
16            MPI_Send(&sum, 1, MPI_INT, partner, send_tag,
17            MPI_COMM_WORLD);
18
19            // Ricevi la somma del processo partner
20            MPI_Recv(&sum_parz, 1, MPI_INT, partner, recv_tag,
21            MPI_COMM_WORLD, &status);
22
23            // Aggiorna la variabile 'sum' con la somma ricevuta
24            sum += sum_parz;
25        } else {
26            partner = menum - array[i];
27            send_tag = 40 + i;
28            recv_tag = 40 + i;
29
30            // Ricevi la somma dal processo partner
31            MPI_Recv(&sum_parz, 1, MPI_INT, partner, recv_tag,
32            MPI_COMM_WORLD, &status);
33
34            // Invia la somma locale al processo partner
35            MPI_Send(&sum, 1, MPI_INT, partner, send_tag,
36            MPI_COMM_WORLD);
37            sum += sum_parz;
38        }
39    }
40
41    return sum;
42 }
```

Listing 2.3: Algoritmo strategia III

2.3 Problematiche affrontate

La problematica principale consiste sul come costruire l'albero delle comunicazioni per la seconda e terza strategie. Al fine di avere un algoritmo ottimizzato, quindi, tutte le inizializzazioni e operazioni costose come logaritmi e potenze, come si evince dal dal Listing 6.1, sono state effettuate prima di utilizzare la funzione `MPI_Wtime`. In particolare, per le potenze, è stato utilizzato un vettore in modo da poter accedere in modo costante durante i diversi cicli for.

Capitolo 3

Descrizione routine

In questo capitolo vengono trattate le funzioni utilizzate per lo scopo del problema: nella sezione 3.1 discuteremo delle funzioni della libreria `MPI` utilizzate, mentre nella sezione 3.2 discuteremo delle funzioni di supporto utilizzate per risolvere il nostro problema.

3.1 Funzioni *MPI* utilizzate

3.1.1 *MPI_Init*

```
int MPI_Init(int *argc, char ***argv)
```

Descrizione: Inizializza l'ambiente di esecuzione MPI.

Parametri di input:

- `argc`: puntatore al numero di parametri
- `argv`: vettore di argomenti

Errors: Restituisce il codice `MPI_SUCCESS` in caso di successo, `MPI_ERR_OTHER` altrimenti.

3.1.2 *MPI_Comm_rank*

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Descrizione: Determina l'identificativo del processo chiamante nel comunicatore.

Parametri di input:

- `comm`: comunicatore

Parametri di output:

- `rank`: id del processo chiamante nel gruppo `comm`

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo, `MPI_ERR_COMM` altrimenti (comunicatiore invalido, e.g comunicatore NULL).

3.1.3 *MPI_Comm_size*

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

Descrizione: Restituisce la dimensione del gruppo associato al comunicatore.

Parametri di input:

- `comm`: comunicatore

Parametri di output:

- `size`: numero di processori nel gruppo `comm`.

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo, `MPI_ERR_COMM` in caso di comunicatore non valido o `MPI_ERR_ARG` se un argomento non è valido e non è identificato da una classe di errore specificata.

3.1.4 *MPI_Bcast*

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
    int root, MPI_Comm comm)
```

Descrizione: Invia un messaggio in broadcast dal processo con id `root` a tutti gli altri processi del gruppo.

Parametri di input:

- `buffer`: puntatore al buffer
- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `root`: id del processo da cui ricevere
- `comm`: communicator

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`, `MPI_ERR_ROOT`).

3.1.5 *MPI_send*

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int ↩
    dest, int tag, MPI_Comm comm)
```

Descrizione: Invia un messaggio in modalità standard e bloccante.

Parametri di input:

- `buf`: puntatore al buffer
- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `dest`: identificativo del processo destinatario
- `tag`: identificativo del messaggio
- `comm`: comunicatore

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`, `MPI_ERR_RANK`).

3.1.6 *MPI_Recv*

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm, MPI_Status *status)
```

Descrizione: Funzione di ricezione, bloccante: ritorna solo dopo che il buffer di ricezione contiene il nuovo messaggio ricevuto.

Parametri di input:

- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `source`: identificativo del processo mittente
- `tag`: identificativo del messaggio
- `comm`: comunicatore

Parametri di output:

- `buf`: puntatore al buffer di ricezione
- `status`: racchiude informazioni sulla ricezione del messaggio

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`, `MPI_ERR_RANK`).

3.1.7 *MPI_Wtime*

```
double MPI_Wtime()
```

Descrizione: Restituisce un tempo in secondi.

Output: tempo in secondi (double).

3.1.8 *MPI_Reduce*

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm)
```

Descrizione: esegue un'operazione di riduzione globale (come somma, massimo, AND logico, ecc.) su tutti i membri di un gruppo.

Parametri di input:

- `sendbuf`: puntatore al buffer
- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `op`: operazione di riduzione (`MPI_MAX`, `MPI_MIN`, `MPI_SUM`, ...)
- `root`: identificativo del processo che visualizzerà il risultato
- `comm`: comunicatore

Parametri di output:

- `recvbuff`: puntatore al buffer di ricezione

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`).

3.1.9 *MPI_Barrier*

```
int MPI_Barrier(MPI_Comm comm)
```

Descrizione: Fornisce un meccanismo sincronizzare per tutti i processi del gruppo: ogni processo si blocca finchè tutti gli altri processi del gruppo non hanno eseguito anch'essi tale routine.

Parametri di input:

- `comm`: communicator

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo, `MPI_ERR_COMM` altrimenti.

3.1.10 *MPI_Abort*

```
int MPI_Abort(MPI_Comm comm, int errorcode)
```

Descrizione: Interrompe tutti i processi appartenenti al gruppo `comm`.

Parametri di input:

- `comm`: communicator
- `errorcode`: codice di errore da restituire all'ambiente di invocazione

Errors: Restituisce solo `MPI_SUCCESS`

3.1.11 *MPI_Finalize*

```
int MPI_Finalize()
```

Descrizione: Termina l'ambiente di esecuzione MPI. Tutti i processi devono chiamare questa routine prima di uscire. Il numero di processi in esecuzione dopo la chiamata di questa routine non è definito.

Errors: Restituisce solo `MPI_SUCCESS`

3.2 Funzioni *ausiliare* utilizzate

3.2.1 La funzione *first_strategy*

```
int first_strategy(int menum, int nproc, int sum)
```

Descrizione: Esegue la somma applicando la prima strategia.

Parametri di input:

- `menum`: l'identificativo del processo
- `nproc`: il numero di processori da utilizzare
- `sum`: la somma parziale fatta precedentemente da ciascun processore

Output: La somma totale.

3.2.2 La funzione *second_strategy*

```
int second_strategy(int menum, int nproc, int *array, int sum)
```

Descrizione: Esegue la somma applicando la seconda strategia.

Parametri di input:

- `menum`: l'identificativo del processo
- `nproc`: il numero di processori da utilizzare
- `array`: il vettore contenente le potenze di due per verificare chi deve partecipare alla comunicazione, e chi deve inviare/ricevere.
- `sum`: la somma parziale fatta precedentemente da ciascun processore

Output: La somma totale.

3.2.3 La funzione *third_strategy*

```
int second_strategy(int menum, int nproc, int *array, int sum)
```

Descrizione: Esegue la somma applicando la terza strategia.

Parametri di input:

- `menum`: l'identificativo del processo
- `nproc`: il numero di processori da utilizzare
- `array`: il vettore contenente le potenze di due per verificare chi deve partecipare alla comunicazione, e chi deve inviare/ricevere.
- `sum`: la somma parziale fatta precedentemente da ciascun processore

Output: La somma totale.

3.2.4 La funzione *check_if_inputs_are_valid*

```
int check_if_inputs_are_valid(int argc, int N, int strategy)
```

Descrizione: Verifica se i parametri passati in ingresso al programma sono quelli corretti. Più precisamente è richiesto che N, ossia il numero di valori nel caso in cui $N \leq 20$, sia uguale a $argc - 3$ (cioè solo i valori da sommare, in quanto vanno esclusi il nome del programma, N stesso e la strategia), che la strategia sia un numero compreso fra 1 e 3 e, infine, che N non sia minore o uguale a 0.

Parametri di input:

- `argc`: il numero di parametri passati in ingresso
- `N`: il numero di valori da sommare
- `strategy`: la strategia da applicare

Output: Restituisce 0 se i parametri sono corretti, `EXIT_FAILURE` altrimenti.

3.2.5 La funzione *fill_array*

```
void fill_array(int *elements, int N, char *argv[])
```

Descrizione: Riempe l'array in modo randomico nel caso in cui $N > 20$, altrimenti viene riempito utilizzando i valori di $argv$ (quelli dal terzo in poi).

Parametri di input:

- `N`: il numero di valori che si vogliono sommare
- `argv`: il vettore di argomenti

Parametri di Output:

- `elements` il vettore di interi contenente i valori da sommare

3.2.6 La funzione *strategy_2_OR_3_are_applicable*

```
int strategy_2_OR_3_are_applicable(int strategy, int nproc)
```

Descrizione: Verifica se le strategie 2 o 3 sono applicabili, ovvero se numero di processori è una potenza di 2.

Parametri di input:

- `strategy`: la strategia da applicare
- `nproc`: il numero di processori che si vuole utilizzare

Output: Restituisce 0 se il numero dei processori è potenza di 2, 1 altrimenti.

3.2.7 La funzione *sequential_sum*

```
int sequential_sum(int *array, int n)
```

Descrizione: Esegue la somma degli elementi del vettore. Usata da ciascun processore per eseguire la propria somma locale (parziale) prima di applicare la strategia desiderata.

Parametri di input:

- `array`: l'array di interi
- `nproc`: la dimensione dell'array

Output: La somma degli elementi dell'array.

3.2.8 La funzione *operand_distribution*

```
void operand_distribution(int menum, int *elements, int *elements_loc, ↩
    int nloc, int nproc, int rest)
```

Descrizione: Il processo con identificativo 0 distribuisce i diversi operandi da sommare a ciascun processo.

Parametri di input:

- `element`: il vettore di interi da distribuire
- `menum`: l'identificativo del processo
- `nloc`: il numero di elementi che ciascun processore dovrebbe fornire "di partenza"
- `nproc`: il numero di processi
- `rest`: il resto della divisione tra N e $nloc$. In base a questo intero capiamo se altri processi devono sommare elementi in più (evitando che quelli "extra" vengano sommati solo dal processo con ID 0).

Parametri di output:

- `elements_loc`: l'array di interi che ciascun processo dovrà sommare inizialmente

3.2.9 La funzione *compute_power_of_two*

```
void compute_power_of_two(int logNproc, int *array)
```

Descrizione: Calcola le potenze di due in base al numero di step da fare per le strategie 2 e 3.

Parametri di input:

- `logNproc`: il numero di step

Parametri di output:

- `array`: l'array di potenze di due

3.2.10 La funzione *print_result*

```
void print_result(int menum, int strategy, int sum, double timetot)
```

Descrizione: Stampa l'ouput: la somma parziale di ciascun processore per le strategie 2 e 3 e i rispettivi tempi, la somma totale e il tempo impiegato altrimenti (strategia 1).

Parametri di input:

- `menum`: l'identificativo del processo
- `strategy`: la strategia applicata
- `sum`: la somma totale o parziale a seconda della strategia
- `timetot`: il tempo impiegato

Capitolo 4

Testing

4.1 Esempio *pbs* utilizzato

A seconda del numero dei valori da testare e del tempo di attesa per il cluster si è deciso di utilizzare due *pbs* diversi:

- uno usato per effettuare più test per più valori in contemporanea, in particolare per N piccoli, in modo da non "monopolizzare" il cluster
- un altro usato per valori più grandi (dieci milioni)

In ogni caso, il numero dei processori e strategia vanno modificati manualmente, in quanto se fossero stati automatizzati anch'essi, i test avrebbero richiesto più tempo occupando, quindi, il cluster.

4.1.1 Primo *pbs*

```
#!/bin/bash

#PBS -q studenti
#PBS -l nodes=8:ppn=8
#PBS -N Main
#PBS -o Main.out
#PBS -e Main.err

cat $PBS_NODEFILE
echo ----------------------------
sort -u $PBS_NODEFILE > hostlist

NCPU=$(wc -l < hostlist)
echo ----------------------------
echo 'This job is allocated on '${NCPU}' cpu(s)'' on host:'
cat hostlist
echo ----------------------------

PBS_O_WORKDIR=$PBS_O_HOME/Progetto_Sum

```

```
for i in {1..10}
do
    echo "**********************************************************************"
    echo "*              ITERATION " $i"                    *"
    echo "**********************************************************************"

    echo -------------------------
    echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main4 $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils.c"
    /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main4 $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils.c -lm -std=c99

    echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np $NCPU $PBS_O_WORKDIR/Main4"
    /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU $PBS_O_WORKDIR/Main4 10000000 3
done
```

Il quale fornisce in output:

```
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
```

```
wn277.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
--------------------------
--------------------------
This job is allocated on 8 cpu(s) on host:
wn273.scope.unina.it
wn274.scope.unina.it
wn275.scope.unina.it
wn276.scope.unina.it
wn277.scope.unina.it
wn278.scope.unina.it
wn279.scope.unina.it
wn280.scope.unina.it
--------------------------
--------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    MNGLGU98B/Progetto_Sum/Main /homes/DMA/PDC/2024/MNGLGU98B/↩
    Progetto_Sum/Main.c /homes/DMA/PDC/2024/MNGLGU98B/Progetto_Sum/↩
    Strategy.c /homes/DMA/PDC/2024/MNGLGU98B/Progetto_Sum/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
    homes/DMA/PDC/2024/MNGLGU98B/Progetto_Sum/Main
Generazione numeri randomici...
Il tempo impiegato da 7 é di 5.016088e-03 s
```

```
82
83 Il tempo impiegato da 1 é di 5.007029e-03 s
84 Il tempo impiegato da 0 é di 5.025864e-03 s
85 Il tempo impiegato da 3 é di 5.012989e-03 s
86 Il tempo impiegato da 6 é di 5.033970e-03 s
87 Il tempo impiegato da 5 é di 5.011082e-03 s
88 Il tempo impiegato da 2 é di 5.018950e-03 s
89 Il tempo impiegato da 4 é di 5.030870e-03 s
90
91
92  Sono il processo 0 e la somma totale e' 10000000
93 Tempo totale impiegato per l'algoritmo: 5.033970e-03
94
95
96  Sono il processo 6 e la somma totale e' 10000000
97
98
99  Sono il processo 2 e la somma totale e' 10000000
100
101  Sono il processo 1 e la somma totale e' 10000000
102  Sono il processo 3 e la somma totale e' 10000000
103  Sono il processo 5 e la somma totale e' 10000000
104  Sono il processo 4 e la somma totale e' 10000000
105  Sono il processo 7 e la somma totale e' 10000000
```

4.1.2 Secondo *pbs*

```
1  #!/bin/bash
2
3  #PBS -q studenti
4  #PBS -l nodes=8:ppn=8
5  #PBS -N Main
6  #PBS -o Main.out
7  #PBS -e Main.err
8
9
10 cat $PBS_NODEFILE
11 echo ----------------------------
12 sort -u $PBS_NODEFILE > hostlist
13
14 NCPU=$(wc -l < hostlist)
15 echo ----------------------------
16 echo 'This job is allocated on '${NCPU}' cpu(s)'' on host:'
17 cat hostlist
18 echo ----------------------------
19
20 PBS_O_WORKDIR=$PBS_O_HOME/ProgettoSomma
21
22 for n in 10000 100000 1000000
23 do
24     for i in {1..10}
25     do
```

```
        echo "↩
    **********************************************************************↩
    "
        echo "* ITERATION " $i" --- N = " $n "                          *"
        echo "↩
    **********************************************************************↩
    "

        echo -------------------------
        echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o ↩
    $PBS_O_WORKDIR/Main $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c ↩
    $PBS_O_WORKDIR/Utils.c"
        /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main ↩
    $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils↩
    .c -lm -std=c99

        echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile ↩
    hotlist -np $NCPU $PBS_O_WORKDIR/Main"
        /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -↩
    np $NCPU $PBS_O_WORKDIR/Main $n 1
    done
done
```

Il quale fornisce il seguente output:

```
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
```

```
wn277.scope.unina.it
wn277.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn276.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn275.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn274.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
wn273.scope.unina.it
--------------------------
--------------------------
This job is allocated on 8 cpu(s) on host:
wn273.scope.unina.it
wn274.scope.unina.it
wn275.scope.unina.it
wn276.scope.unina.it
wn277.scope.unina.it
wn278.scope.unina.it
wn279.scope.unina.it
wn280.scope.unina.it
--------------------------
**************************************************************
* ITERATION  1  --- N =  100000                              *
**************************************************************
--------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
    Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
```

```
82 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
       homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
83 Il tempo impiegato da 7 é di 6.103516e-05 s
84 Il tempo impiegato da 1 é di 5.793571e-05 s
85 Il tempo impiegato da 2 é di 5.793571e-05 s
86 Il tempo impiegato da 6 é di 6.198883e-05 s
87 Il tempo impiegato da 5 é di 6.198883e-05 s
88 Il tempo impiegato da 4 é di 6.103516e-05 s
89 Il tempo impiegato da 3 é di 5.888939e-05 s
90 Il tempo impiegato da 0 é di 1.049042e-04 s
91 La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato ↩
       1.049042e-04.
92 ********************************************************************
93 * ITERATION  2  --- N =  100000                                    *
94 ********************************************************************
95 -------------------------
96 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
       FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
       ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
       Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
97 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
       homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
98 Il tempo impiegato da 7 é di 6.508827e-05 s
99 Il tempo impiegato da 3 é di 5.793571e-05 s
100 Il tempo impiegato da 1 é di 5.793571e-05 s
101 Il tempo impiegato da 4 é di 6.079674e-05 s
102 Il tempo impiegato da 2 é di 6.294250e-05 s
103 Il tempo impiegato da 5 é di 6.103516e-05 s
104 Il tempo impiegato da 6 é di 6.079674e-05 s
105 Il tempo impiegato da 0 é di 1.111031e-04 s
106 La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato ↩
       1.111031e-04.
107 ********************************************************************
108 * ITERATION  3  --- N =  100000                                    *
109 ********************************************************************
110 -------------------------
111 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
       FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
       ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
       Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
112 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
       homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
113 Il tempo impiegato da 7 é di 6.103516e-05 s
114 Il tempo impiegato da 1 é di 5.793571e-05 s
115 Il tempo impiegato da 3 é di 5.912781e-05 s
116 Il tempo impiegato da 2 é di 5.888939e-05 s
117 Il tempo impiegato da 4 é di 6.198883e-05 s
118 Il tempo impiegato da 5 é di 6.198883e-05 s
119 Il tempo impiegato da 6 é di 6.103516e-05 s
120 Il tempo impiegato da 0 é di 1.008511e-04 s
121 La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato ↩
       1.008511e-04.
122 ********************************************************************
123 * ITERATION  4  --- N =  100000                                    *
```

```
124 ************************************************************************
125 -------------------------
126 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
        Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
127 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
128 Il tempo impiegato da 7 é di 6.103516e-05 s
129 Il tempo impiegato da 2 é di 5.793571e-05 s
130 Il tempo impiegato da 3 é di 5.793571e-05 s
131 Il tempo impiegato da 5 é di 6.103516e-05 s
132 Il tempo impiegato da 4 é di 6.103516e-05 s
133 Il tempo impiegato da 6 é di 6.198883e-05 s
134 Il tempo impiegato da 0 é di 1.189709e-04 s
135 Il tempo impiegato da 1 é di 5.698204e-05 s
136 La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato ↩
        1.189709e-04.
137 ************************************************************************
138 * ITERATION  5  --- N =  100000                                *
139 ************************************************************************
140 -------------------------
141 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
        Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
142 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
143 Il tempo impiegato da 7 é di 6.103516e-05 s
144 Il tempo impiegato da 1 é di 5.793571e-05 s
145 Il tempo impiegato da 6 é di 6.198883e-05 s
146 Il tempo impiegato da 5 é di 6.222725e-05 s
147 Il tempo impiegato da 4 é di 6.198883e-05 s
148 Il tempo impiegato da 3 é di 6.484985e-05 s
149 Il tempo impiegato da 0 é di 1.111031e-04 s
150 Il tempo impiegato da 2 é di 5.888939e-05 s
151 La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato ↩
        1.111031e-04.
152 ************************************************************************
153 * ITERATION  6  --- N =  100000                                *
154 ************************************************************************
155 -------------------------
156 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
        Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
157 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
158 Il tempo impiegato da 7 é di 6.008148e-05 s
159 Il tempo impiegato da 1 é di 5.888939e-05 s
160 Il tempo impiegato da 5 é di 6.103516e-05 s
161 Il tempo impiegato da 3 é di 5.793571e-05 s
162 Il tempo impiegato da 2 é di 5.912781e-05 s
163 Il tempo impiegato da 4 é di 6.198883e-05 s
```

```
Il tempo impiegato da 6 é di 6.198883e-05 s
Il tempo impiegato da 0 é di 1.118183e-04 s
La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato 1.118183e-04.
*****************************************************************
* ITERATION  7  --- N =  100000                                 *
*****************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 6.079674e-05 s
Il tempo impiegato da 5 é di 6.079674e-05 s
Il tempo impiegato da 3 é di 6.198883e-05 s
Il tempo impiegato da 6 é di 6.198883e-05 s
Il tempo impiegato da 1 é di 5.793571e-05 s
Il tempo impiegato da 4 é di 6.103516e-05 s
Il tempo impiegato da 2 é di 5.793571e-05 s
Il tempo impiegato da 0 é di 1.080036e-04 s
La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato 1.080036e-04.
*****************************************************************
* ITERATION  8  --- N =  100000                                 *
*****************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 6.079674e-05 s
Il tempo impiegato da 1 é di 6.198883e-05 s
Il tempo impiegato da 2 é di 5.817413e-05 s
Il tempo impiegato da 6 é di 6.198883e-05 s
Il tempo impiegato da 3 é di 5.888939e-05 s
Il tempo impiegato da 5 é di 6.198883e-05 s
Il tempo impiegato da 4 é di 6.103516e-05 s
Il tempo impiegato da 0 é di 1.099110e-04 s
La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato 1.099110e-04.
*****************************************************************
* ITERATION  9  --- N =  100000                                 *
*****************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
```

```
Il tempo impiegato da 7 é di 6.079674e-05 s
Il tempo impiegato da 3 é di 5.793571e-05 s
Il tempo impiegato da 2 é di 5.793571e-05 s
Il tempo impiegato da 5 é di 6.103516e-05 s
Il tempo impiegato da 4 é di 6.198883e-05 s
Il tempo impiegato da 6 é di 6.198883e-05 s
Il tempo impiegato da 1 é di 5.793571e-05 s
Il tempo impiegato da 0 é di 1.130104e-04 s
La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato 1.130104e-04.
**********************************************************************
* ITERATION  10 --- N =  100000                                      *
**********************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 6.103516e-05 s
Il tempo impiegato da 1 é di 5.793571e-05 s
Il tempo impiegato da 2 é di 5.912781e-05 s
Il tempo impiegato da 4 é di 6.198883e-05 s
Il tempo impiegato da 5 é di 6.103516e-05 s
Il tempo impiegato da 6 é di 6.103516e-05 s
Il tempo impiegato da 3 é di 5.793571e-05 s
Il tempo impiegato da 0 é di 1.070499e-04 s
La somma totale e' 100000 e l'algoritmo, per calcolarla, ha impiegato 1.070499e-04.
**********************************************************************
* ITERATION  1  --- N =  1000000                                     *
**********************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 5.881786e-04 s
Il tempo impiegato da 1 é di 5.109310e-04 s
Il tempo impiegato da 2 é di 5.068779e-04 s
Il tempo impiegato da 6 é di 5.099773e-04 s
Il tempo impiegato da 4 é di 5.102158e-04 s
Il tempo impiegato da 3 é di 5.080700e-04 s
Il tempo impiegato da 5 é di 5.059242e-04 s
Il tempo impiegato da 0 é di 6.291866e-04 s
La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato 6.291866e-04.
**********************************************************************
* ITERATION  2  --- N =  1000000                                     *
**********************************************************************
-------------------------
```

```
246 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
      Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
247 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
248 Il tempo impiegato da 7 é di 5.049706e-04 s
249 Il tempo impiegato da 3 é di 5.021095e-04 s
250 Il tempo impiegato da 5 é di 5.090237e-04 s
251 Il tempo impiegato da 0 é di 5.128384e-04 s
252 Il tempo impiegato da 1 é di 5.021095e-04 s
253 Il tempo impiegato da 2 é di 5.030632e-04 s
254 Il tempo impiegato da 4 é di 5.059242e-04 s
255 Il tempo impiegato da 6 é di 5.061626e-04 s
256 La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato ↩
      5.128384e-04.
257 ************************************************************************
258 * ITERATION  3  --- N =  1000000                                       *
259 ************************************************************************
260 -------------------------
261 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
      Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
262 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
263 Il tempo impiegato da 7 é di 5.059242e-04 s
264 Il tempo impiegato da 4 é di 5.109310e-04 s
265 Il tempo impiegato da 5 é di 5.109310e-04 s
266 Il tempo impiegato da 0 é di 5.538464e-04 s
267 Il tempo impiegato da 1 é di 5.061626e-04 s
268 Il tempo impiegato da 6 é di 5.099773e-04 s
269 Il tempo impiegato da 2 é di 5.030632e-04 s
270 Il tempo impiegato da 3 é di 5.059242e-04 s
271 La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato ↩
      5.538464e-04.
272 ************************************************************************
273 * ITERATION  4  --- N =  1000000                                       *
274 ************************************************************************
275 -------------------------
276 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
      Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
277 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
278 Il tempo impiegato da 7 é di 5.130768e-04 s
279 Il tempo impiegato da 1 é di 5.028248e-04 s
280 Il tempo impiegato da 2 é di 5.068779e-04 s
281 Il tempo impiegato da 6 é di 5.059242e-04 s
282 Il tempo impiegato da 5 é di 5.049706e-04 s
283 Il tempo impiegato da 4 é di 5.102158e-04 s
284 Il tempo impiegato da 3 é di 5.018711e-04 s
285 Il tempo impiegato da 0 é di 5.578995e-04 s
```

```
La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato 5.578995e-04.
**********************************************************************
* ITERATION  5  --- N =  1000000                                     *
**********************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 5.080700e-04 s
Il tempo impiegato da 1 é di 5.040169e-04 s
Il tempo impiegato da 5 é di 5.059242e-04 s
Il tempo impiegato da 6 é di 5.052090e-04 s
Il tempo impiegato da 2 é di 5.071163e-04 s
Il tempo impiegato da 4 é di 5.049706e-04 s
Il tempo impiegato da 3 é di 5.030632e-04 s
Il tempo impiegato da 0 é di 5.578995e-04 s
La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato 5.578995e-04.
**********************************************************************

* ITERATION  6  --- N =  1000000                                     *
**********************************************************************

-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 5.049706e-04 s
Il tempo impiegato da 1 é di 5.021095e-04 s
Il tempo impiegato da 3 é di 5.030632e-04 s
Il tempo impiegato da 6 é di 5.068779e-04 s
Il tempo impiegato da 4 é di 5.059242e-04 s
Il tempo impiegato da 2 é di 5.028248e-04 s
Il tempo impiegato da 5 é di 5.087852e-04 s
Il tempo impiegato da 0 é di 5.140305e-04 s
La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato 5.140305e-04.
**********************************************************************
* ITERATION  7  --- N =  1000000                                     *
**********************************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
```

```
323 Il tempo impiegato da 7 é di 5.071163e-04 s
324 Il tempo impiegato da 1 é di 5.068779e-04 s
325 Il tempo impiegato da 5 é di 5.059242e-04 s
326 Il tempo impiegato da 4 é di 5.099773e-04 s
327 Il tempo impiegato da 6 é di 5.099773e-04 s
328 Il tempo impiegato da 3 é di 5.059242e-04 s
329 Il tempo impiegato da 2 é di 5.059242e-04 s
330 Il tempo impiegato da 0 é di 5.648136e-04 s
331 La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato ↩
      5.648136e-04.
332 ********************************************************************
333 * ITERATION  8  --- N =  1000000                          *
334 ********************************************************************
335 -------------------------
336 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
      Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
337 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
338 Il tempo impiegato da 7 é di 5.068779e-04 s
339 Il tempo impiegato da 1 é di 5.068779e-04 s
340 Il tempo impiegato da 5 é di 5.099773e-04 s
341 Il tempo impiegato da 3 é di 5.059242e-04 s
342 Il tempo impiegato da 2 é di 5.059242e-04 s
343 Il tempo impiegato da 4 é di 5.090237e-04 s
344 Il tempo impiegato da 6 é di 5.090237e-04 s
345 Il tempo impiegato da 0 é di 5.629063e-04 s
346 La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato ↩
      5.629063e-04.
347 ********************************************************************
348 * ITERATION  9  --- N =  1000000                          *
349 ********************************************************************
350 -------------------------
351 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
      Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
352 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
353 Il tempo impiegato da 7 é di 5.068779e-04 s
354 Il tempo impiegato da 4 é di 5.090237e-04 s
355 Il tempo impiegato da 3 é di 5.061626e-04 s
356 Il tempo impiegato da 2 é di 5.071163e-04 s
357 Il tempo impiegato da 5 é di 5.068779e-04 s
358 Il tempo impiegato da 1 é di 5.090237e-04 s
359 Il tempo impiegato da 6 é di 5.059242e-04 s
360 Il tempo impiegato da 0 é di 5.578995e-04 s
361 La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato ↩
      5.578995e-04.
362 ********************************************************************
363 * ITERATION  10 --- N =  1000000                          *
364 ********************************************************************
365 -------------------------
```

```
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 8 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
Il tempo impiegato da 7 é di 5.040169e-04 s
Il tempo impiegato da 1 é di 5.018711e-04 s
Il tempo impiegato da 2 é di 5.030632e-04 s
Il tempo impiegato da 6 é di 5.061626e-04 s
Il tempo impiegato da 5 é di 5.049706e-04 s
Il tempo impiegato da 4 é di 5.059242e-04 s
Il tempo impiegato da 3 é di 5.018711e-04 s
Il tempo impiegato da 0 é di 5.559921e-04 s
La somma totale e' 1000000 e l'algoritmo, per calcolarla, ha impiegato 5.559921e-04.
```

4.2 Casi limite

4.2.1 Caso $N = 0$

File *pbs* per la generazione dell'errore

```
#!/bin/bash

#PBS -q studenti
#PBS -l nodes=1:ppn=1
#PBS -N Main
#PBS -o Main.out
#PBS -e Main.err


cat $PBS_NODEFILE
echo --------------------------
sort -u $PBS_NODEFILE > hostlist

NCPU=$(wc -l < hostlist)
echo --------------------------
echo 'This job is allocated on '${NCPU}' cpu(s)'' on host:'
cat hostlist
echo --------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoSomma

echo --------------------------
echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils.c"
/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils.c -lm -std=c99

```

```
26 echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np ↩
       $NCPU $PBS_O_WORKDIR/Main"
27 /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU ↩
       $PBS_O_WORKDIR/Main 0 1
```

File *.out*

```
1  wn280.scope.unina.it
2  --------------------------
3  --------------------------
4  This job is allocated on 1 cpu(s) on host:
5  wn280.scope.unina.it
6  --------------------------
7  --------------------------
8  Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
       FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
       ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
       Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
9  Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 1 /↩
       homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
10 Input non valido: chiusura del programma.
```

File *.err*

```
1  Inserire un numero maggiore di 0!
2  --------------------------------------------------------------------------
3  MPI_ABORT was invoked on rank 0 in communicator MPI_COMM_WORLD
4  with errorcode 1.
5
6  NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
7  You may or may not see output from other processes, depending on
8  exactly when Open MPI kills them.
9  --------------------------------------------------------------------------
10 --------------------------------------------------------------------------
11 mpiexec has exited due to process rank 0 with PID 17843 on
12 node wn280.scope.unina.it exiting without calling "finalize". This may
13 have caused other processes in the application to be
14 terminated by signals sent by mpiexec (as reported here).
15 --------------------------------------------------------------------------
```

4.2.2 Caso *strategia errata*

```
1  #!/bin/bash
2
3  #PBS -q studenti
4  #PBS -l nodes=1:ppn=1
5  #PBS -N Main
6  #PBS -o Main.out
7  #PBS -e Main.err
```

```
8
9
10 cat $PBS_NODEFILE
11 echo --------------------------
12 sort -u $PBS_NODEFILE > hostlist
13
14 NCPU=$(wc -l < hostlist)
15 echo --------------------------
16 echo 'This job is allocated on '${NCPU}' cpu(s)'' on host:'
17 cat hostlist
18 echo --------------------------
19
20 PBS_O_WORKDIR=$PBS_O_HOME/ProgettoSomma
21
22 echo --------------------------
23 echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/↩
       Main $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/↩
       Utils.c"
24 /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main ↩
       $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils↩
       .c -lm -std=c99
25
26 echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np ↩
       $NCPU $PBS_O_WORKDIR/Main"
27 /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU ↩
       $PBS_O_WORKDIR/Main 100 0
```

File *.out*

```
1 wn280.scope.unina.it
2 --------------------------
3 --------------------------
4 This job is allocated on 1 cpu(s) on host:
5 wn280.scope.unina.it
6 --------------------------
7 --------------------------
8 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
       FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
       ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
       Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
9 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 1 /↩
       homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
10 Input non valido: chiusura del programma.
```

File *.err*

```
1 Strategia non valida: inserire un numero compreso fra 1 e 3!
2 -------------------------------------------------------------------------
3 MPI_ABORT was invoked on rank 0 in communicator MPI_COMM_WORLD
4 with errorcode 1.
5
```

```
NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------
--------------------------------------------------------------------------
mpiexec has exited due to process rank 0 with PID 17960 on
node wn280.scope.unina.it exiting without calling "finalize". This may
have caused other processes in the application to be
terminated by signals sent by mpiexec (as reported here).
--------------------------------------------------------------------------
```

4.2.3 Caso $N! = argv$

```
#!/bin/bash

#PBS -q studenti
#PBS -l nodes=1:ppn=1
#PBS -N Main
#PBS -o Main.out
#PBS -e Main.err


cat $PBS_NODEFILE
echo ----------------------------
sort -u $PBS_NODEFILE > hostlist

NCPU=$(wc -l < hostlist)
echo ----------------------------
echo 'This job is allocated on '${NCPU}' cpu(s)'' on host:'
cat hostlist
echo ----------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoSomma

echo ----------------------------
echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/↩
    Main $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/↩
    Utils.c"
/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main ↩
    $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Strategy.c $PBS_O_WORKDIR/Utils↩
    .c -lm -std=c99

echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np ↩
    $NCPU $PBS_O_WORKDIR/Main"
/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU ↩
    $PBS_O_WORKDIR/Main 10 1 1 2 3
```

File *.out*

```
wn280.scope.unina.it
----------------------------
```

```
3 -------------------------
4 This job is allocated on 1 cpu(s) on host:
5 wn280.scope.unina.it
6 -------------------------
7 -------------------------
8 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoSomma/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoSomma/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/↩
      Strategy.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Utils.c
9 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 1 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoSomma/Main
10 Input non valido: chiusura del programma.
```

File *.err*

```
1 --------------------------------------------------------------------------
2 MPI_ABORT was invoked on rank 0 in communicator MPI_COMM_WORLD
3 with errorcode 1.
4
5 NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
6 You may or may not see output from other processes, depending on
7 exactly when Open MPI kills them.
8 --------------------------------------------------------------------------
9 Il numero di elementi inserito non corrisponde ad N!
10 --------------------------------------------------------------------------
11 mpiexec has exited due to process rank 0 with PID 18319 on
12 node wn280.scope.unina.it exiting without calling "finalize". This may
13 have caused other processes in the application to be
14 terminated by signals sent by mpiexec (as reported here).
15 --------------------------------------------------------------------------
```

Capitolo 5

Analisi performance

In questo capitolo analizzeremo il tempo medio impiegato, lo speed up e l'efficienza al variare del numero dei processori con N fissato. Calcoleremo, inoltre, lo speed up e l'efficienza scalata per ciascun processore al variare di N. Quindi:

- $N = 10.000.000$ nel primo caso, ossia, quello in cui N è fisso e varia il numero dei processori
- Numero dei processori P_i con $i \in 1, ..., 8$
- Per calcolare il tempo medio, per ciascun caso il programma è stato eseguito esattamente 10 volte per considerare, appunto, la media aritmetica
- Una volta ricavato il tempo medio è stato possibile calcolare lo speed e l'efficienza mediante le seguenti formule:
  - Speed up $S = \frac{T(1)}{T(P)}$
  - Efficienza $E = \frac{S(p)}{p}$
- Infine, abbiamo deciso poi di calcolare Speed up ed efficienza scalati. Per capire quale fosse la giusta quantità N da testare a seconda della quantità dei processori è stata utilizzata la seguente formula:

$$K = \frac{P_1 * log(P_1)}{P_0 * log(P_0)}$$
$$N_1 = N_0 * K = N_0 * I(N_0, P_0, P_1)$$

  Trovata l'isoefficienza abbiamo calcolato lo speed up e l'efficienza scalata:

  - Speed up Scalato: $SS = \frac{I(N_0,P_0,P_1)*T(P_0,N_0)}{T(P_1,N_1)}$
  - Efficienza Scalata: $ES = \frac{SS}{p}$

Tutti i test sono stati effettuati con un vettore (di dimensione N, naturalmente) contenenente solo il valore 1.

5.1 Analisi con *diecimila*

5.1.1 Analisi I strategia

N. processori	P1	P2	P3	P4	P5	P6	P7	P8
	4.696846e-05	3.695488e-05	1.239777e-04	3.099442e-05	1.239777e-04	1.540184e-04	1.435614e-02	6.198883e-05
	4.720688e-05	3.886223e-05	1.211166e-04	2.694130e-05	1.211166e-04	5.118847e-04	1.938200e-02	6.794930e-05
	4.696846e-05	3.814697e-05	1.170635e-04	4.601479e-05	1.170635e-04	5.300045e-04	2.180600e-02	6.699562e-05
	4.696846e-05	3.695488e-05	1.249313e-04	5.102158e-05	1.249313e-04	5.400181e-04	2.163315e-02	6.604195e-05
	4.816055e-05	3.695488e-05	1.201630e-04	4.696846e-05	1.201630e-04	9.298325e-05	2.139997e-02	6.604195e-05
	4.696846e-05	3.719330e-05	1.230240e-04	3.004074e-05	1.230240e-04	1.158714e-04	1.300788e-02	6.699562e-05
	4.720688e-05	3.790855e-05	1.330376e-04	4.506111e-05	1.330376e-04	5.421638e-04	1.939821e-02	6.699562e-05
	4.696846e-05	3.695488e-05	1.180172e-04	4.601479e-05	1.180172e-04	1.161098e-04	1.924014e-02	6.699562e-05
	4.696846e-05	4.196167e-05	1.180172e-04	2.813339e-05	1.180172e-04	5.159378e-04	1.431012e-02	6.604195e-05
	4.696846e-05	3.695488e-05	1.299381e-04	2.694130e-05	1.299381e-04	5.512238e-04	2.187395e-02	7.510185e-05
Tempo medio	4.713535e-5	3.788471e-5	1.2292862e-4	3.781318e-5	2.892017e-5	3.670215e-5	1.8640756e-2	6.711483e-5
Speed up	1	1.24417872012	0.38343674565	1.24653229377	1.62984346219	1.28426672552	0.00252861793	0.70230901277
Efficienza	1	0.62208936006	0.12781224855	0.31163307344	0.32596869243	0.21404445425	0.00036123113	0.08778862659

Tabella 5.1: Strategia I con $N = 10^4$

Figura 5.1: Plot *I strategia* del tempo al variare del numero dei processori per $N = 10^4$

Come si evince dal grafico sopra (Fig. 5.1), in generale, il tempo è più o meno costante in quanto si utilizzano $N = 10^4$ elementi, ed è probabile che a causa di questa bassa quantità non si riscontri un tempo sempre decrescente. Inoltre, si presenta un problema quando i processori utilizzati sono 7 e ciò potrebbe essere un problema dovuto al cluster.

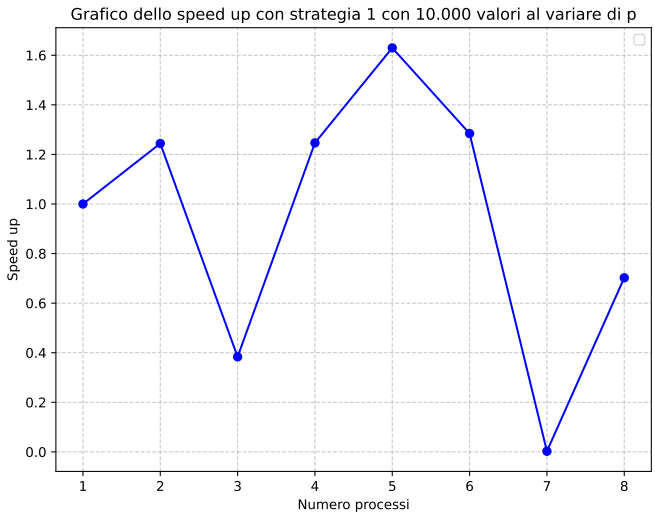


Figura 5.2: Plot *I strategia* dello speed up al variare del numero dei processori per $N = 10^4$

In quest'altro grafico (Fig. 5.2), la funzione non è lineare, ma è comunque, evidente che lo speed up non si avvicini a quello ideale implicando che aumentare il numero dei processori con un N così piccolo, in questo caso, è inutile.

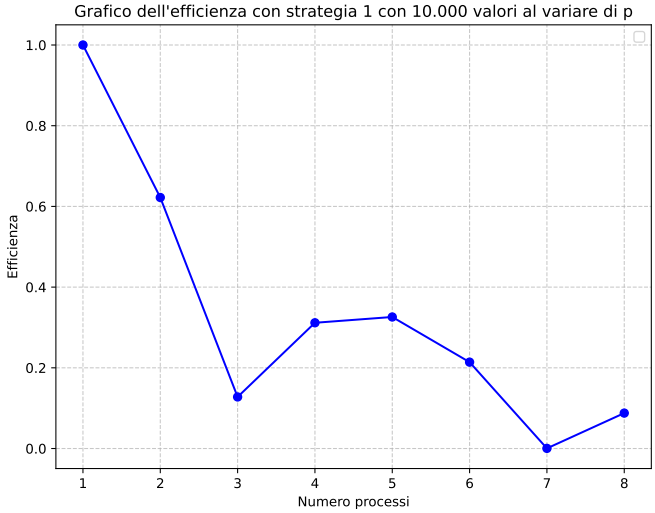


Figura 5.3: Plot *I strategia* dell'efficienza al variare del numero dei processori per $N = 10^4$

Nel grafico 5.3 si nota come l'efficienza degrada molto lentamente a causa del numero esiguo di elementi pari a $N = 10^4$. Con 3 e 7 processori, stranamen-

te, non degrada lentamente ma più velocemente, ciò nonostante la differenza è quasi illusoria.

5.1.2 Analisi II strategia

N. processori	P1	P2	P4	P8
	4.696846e-05	3.600121e-05	3.480911e-05	5.292892e-05
	4.720688e-05	3.719330e-05	4.696846e-05	3.719330e-05
	4.696846e-05	3.290176e-05	3.504753e-05	4.482269e-05
	4.696846e-05	3.600121e-05	3.910065e-05	4.386902e-05
	4.816055e-05	3.385544e-05	4.696846e-05	3.504753e-05
	4.696846e-05	3.218651e-05	5.888939e-05	4.410744e-05
	4.720688e-05	3.409386e-05	5.888939e-05	5.698204e-05
	4.696846e-05	3.194809e-05	3.910065e-05	4.315376e-05
	4.696846e-05	3.194809e-05	3.814697e-05	4.410744e-05
	4.696846e-05	3.314018e-05	5.793571e-05	4.386902e-05
Tempo medio	4.713535e-05	3.392696e-05	4.558563e-05	4.460811e-05
Speed up	1	1.38931840636	1.03399580087	1.05665427206
Efficienza	1	0.69465920318	0.25849895021	0.132081784

Tabella 5.2: Strategia II con $N = 10^4$

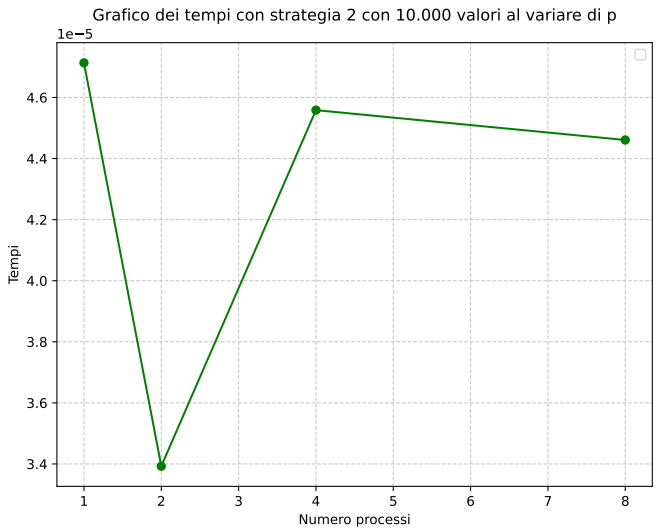


Figura 5.4: Plot *II strategia* del tempo al variare del numero dei processori per $N = 10^4$

Nel grafico 5.4, a differenza dei precedenti, abbiamo un miglioramento quasi impercettibile dei tempi a causa del numero esiguo di elementi pari a $N = 10^4$ sui tempi rispetto la strategia 1. Anche in questo caso la diminuzione progressiva dei tempi risulta quasi irrisoria.

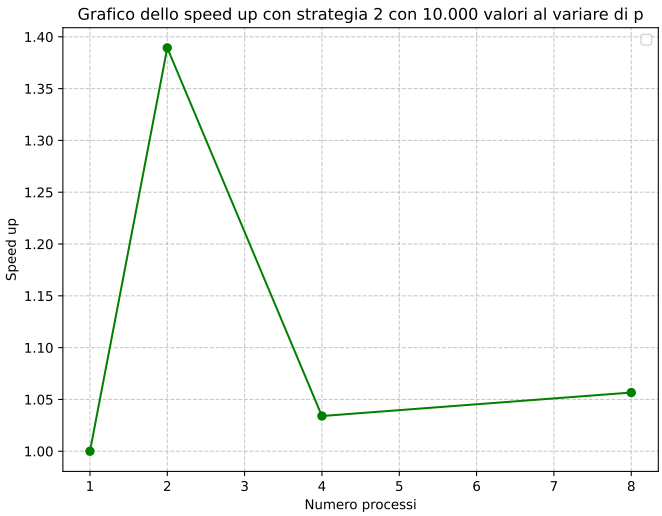


Figura 5.5: Plot *II strategia* dello speed up al variare del numero dei processori per $N = 10^4$

Per quanto riguarda lo speed up (Figura 5.5) col variare del tempo con l'ausilio della strategia 2 si nota quanto i valori siano relativamente lontani dai rispettivi speed up ideali, questo a causa del numero basso di valori utilizzato. Tuttavia, rispetto alla prima strategia, con 2 processori lo speed up è leggermente migliorato, ma risulta comunque difficile fare un confronto preciso a causa del basso numero di elementi da sommare.

Figura 5.6: Plot *II strategia* dell'efficienza al variare del numero dei processori per $N = 10^4$

Nel grafico 5.6 utilizzato per rappresentare la variazione dell'efficienza col

variare del tempo, si intuisce come l'efficienza degrada nel tempo a causa del numero di valori fisso utilizzato.

5.1.3 Analisi III strategia

N. processori	P1	P2	P4	P8
	4.696846e-05	6.198883e-05	4.792213e-05	5.602837e-05
	4.720688e-05	5.006790e-05	4.410744e-05	5.102158e-05
	4.696846e-05	5.507469e-05	4.220009e-05	5.412102e-05
	4.696846e-05	5.197525e-05	4.506111e-05	5.602837e-05
	4.816055e-05	4.100800e-05	4.220009e-05	5.412102e-05
	4.696846e-05	5.006790e-05	4.196167e-05	5.578995e-05
	4.720688e-05	4.506111e-05	4.410744e-05	5.388260e-05
	4.696846e-05	4.410744e-05	4.100800e-05	5.507469e-05
	4.696846e-05	3.790855e-05	4.410744e-05	4.792213e-05
	4.696846e-05	3.910065e-05	4.315376e-05	5.698204e-05
Tempo medio	4.713535e-05	4.7636032e-05	4.3582917e-05	5.4097177e-05
Speed up	1	0.98948942	1.0815097	0.87130886
Efficienza	1	0.49474471	0.270377425	0.10891360

Tabella 5.3: Strategia III con $N = 10^4$

Figura 5.7: Plot *III strategia* del tempo al variare del numero dei processori per $N = 10^4$

Nel grafico 5.7 si può notare come il tempo aumenti all'aumentare del numero dei processori e questo è sia a causa del numero esiguo da sommare e

del numero di comunicazioni da fare, implicando che la strategia è abbastanza sprecata nel caso in cui N non sia sufficientemente grande.

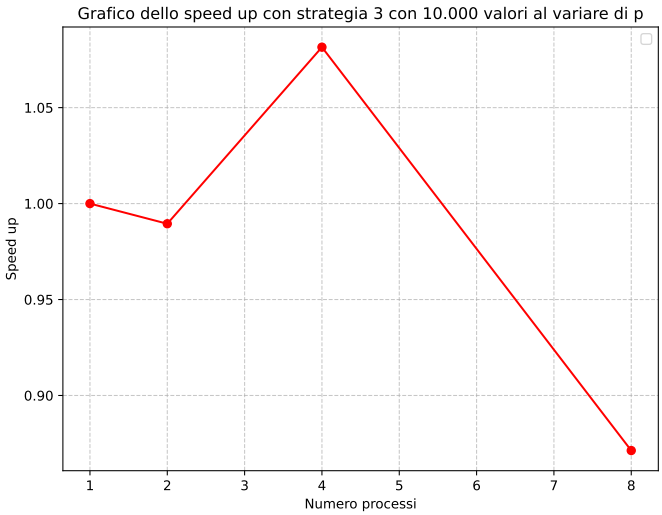


Figura 5.8: Plot *I strategia* dello speed up al variare del numero dei processori per $N = 10^4$

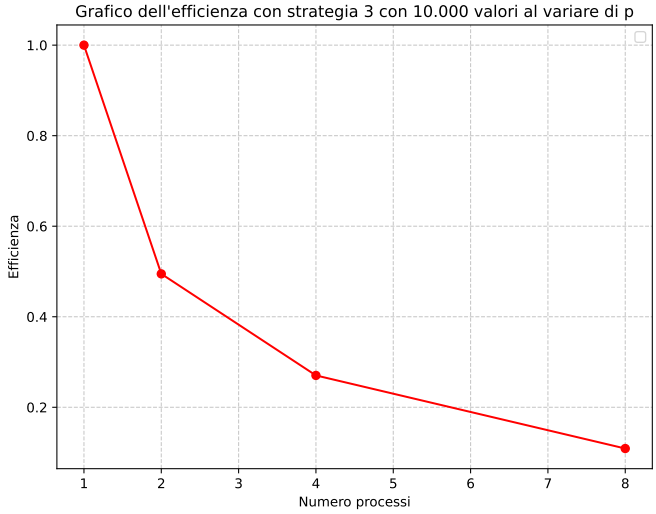


Figura 5.9: Plot *III strategia* dell'efficienza al variare del numero dei processori per $N = 10^4$

Speed up ed efficienza, invece, sono analoghi alla strategia 2: si allontanano sempre di più da quelli ideali.

5.2 Analisi con *centomila*

5.2.1 Analisi I strategia

N. processori	P1	P2	P3	P4	P5	P6	P7	P8
	4.529953e-04	2.329350e-04	1.739979e-03	1.151562e-04	1.008511e-04	4.920959e-04	9.685993e-03	1.049042e-04
	4.568100e-04	2.160072e-04	1.693964e-03	1.149178e-04	9.608269e-05	4.761219e-04	1.875591e-02	1.111031e-04
	4.518032e-04	2.429485e-04	7.350206e-03	1.139641e-04	1.020432e-04	4.808903e-04	1.921415e-02	1.008511e-04
	5.600452e-04	2.110004e-04	7.929802e-04	1.149178e-04	1.020432e-04	4.951954e-04	1.009703e-02	1.189709e-04
	3.919601e-04	2.110004e-04	1.021862e-03	1.161098e-04	9.489059e-05	4.727840e-04	1.882720e-02	1.111031e-04
	3.881454e-04	2.110004e-04	1.582146e-03	1.180172e-04	1.020432e-04	4.780293e-04	1.927710e-02	1.118183e-04
	4.680157e-04	2.441406e-04	2.400875e-04	3.938675e-04	9.608269e-05	4.999638e-04	1.920104e-02	1.080036e-04
	3.890991e-04	2.100468e-04	2.360344e-04	1.158714e-04	1.010895e-04	5.049706e-04	1.893497e-02	1.099110e-04
	3.869534e-04	2.138615e-04	2.331734e-04	1.168251e-04	1.029968e-04	1.104212e-02	1.886487e-02	1.130104e-04
	3.900528e-04	2.100468e-04	2.300739e-04	1.158714e-04	1.010895e-04	4.999638e-04	1.888299e-02	1.070499e-04
Tempo medio	4.3358802e-4	2.2029876e-4	1.51205064e-3	1.4355183e-4	9.992124e-5	1.5442135e-3	1.71741253e-2	1.0967256e-4
Speed up	1	1.96818184542	0.28675495947	3.02042837071	4.33929783097	0.28078243066	0.02524658533	3.95347769761
Efficienza	1	0.98409092271	0.09558498649	0.75510709267	0.86785956619	0.04679707177	0.00360665504	0.4941847122

Tabella 5.4: Strategia I con $N = 10^5$

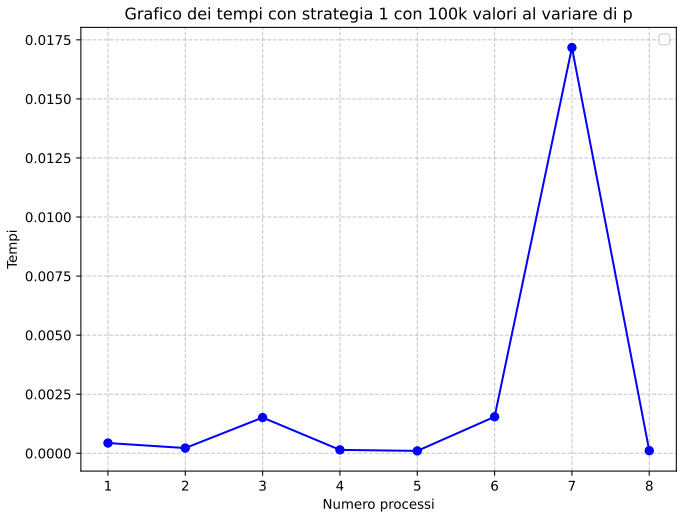


Figura 5.10: Plot *I strategia* del tempo al variare del numero dei processori per $N = 10^5$

Con 100 mila, invece, i tempi sono leggermente peggiori, in quanto N è più grande, ma speed up ed efficienza sono più alti in alcuni casi implicando che il parallelismo diventa più determinante.

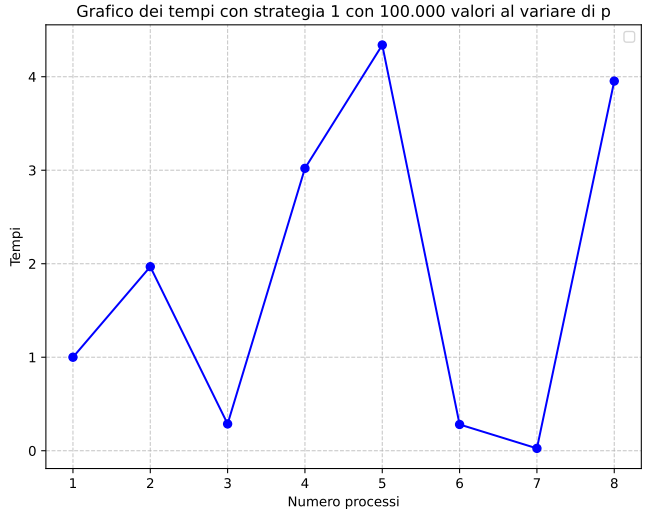


Figura 5.11: Plot *I strategia* dello speed up al variare del numero dei processori per $N = 10^5$

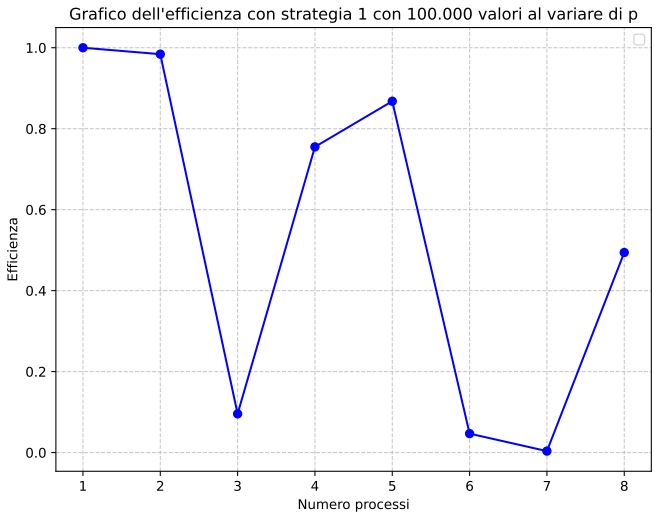


Figura 5.12: Plot *I strategia* dell'efficienza al variare del numero dei processori per $N = 10^5$

5.2.2 Analisi II strategia

N. processori	P1	P2	P4	P8
	4.529953e-04	2.119541e-04	1.440048e-04	7.891655e-05
	4.568100e-04	2.431870e-04	1.339912e-04	8.106232e-05
	4.518032e-04	2.100468e-04	1.330376e-04	8.606911e-05
	5.600452e-04	2.441406e-04	1.327991e-04	9.012222e-05
	3.919601e-04	2.338886e-04	1.330376e-04	8.082390e-05
	3.881454e-04	2.379417e-04	1.301765e-04	8.201599e-05
	4.680157e-04	2.110004e-04	1.311302e-04	8.296967e-05
	3.890991e-04	2.098083e-04	1.311302e-04	7.891655e-05
	3.869534e-04	2.398491e-04	1.339912e-04	7.390976e-05
	3.900528e-04	2.419949e-04	1.339912e-04	8.296967e-05
Tempo medio	4.3358802e-04	2.2838115e-04	1.3372896e-04	8.177757e-05
Speed up	1	1.89852805277	3.24228962	5.302040889
Efficienza	1	0.9492640	0.810572405	0.66275511

Tabella 5.5: Strategia II con $N = 10^5$

Figura 5.13: Plot *II strategia* del tempo al variare del numero dei processori per $N = 10^5$

In questi casi è possibile notare chiaramente come il tempo diminuisca all'aumentare del numero dei processori, lo speed up da quattro processori in poi si allontana sempre di più da quello ideale in quanto un numero così elevato di processori non è necessario per sommare tale quantità (100k). Quindi, rispetto ai casi precedenti dove aumentare il numero dei processori era uno spreco, in questo caso aumentare (di poco) porta effettivamente dei vantaggi in termini di tempo e speed up.

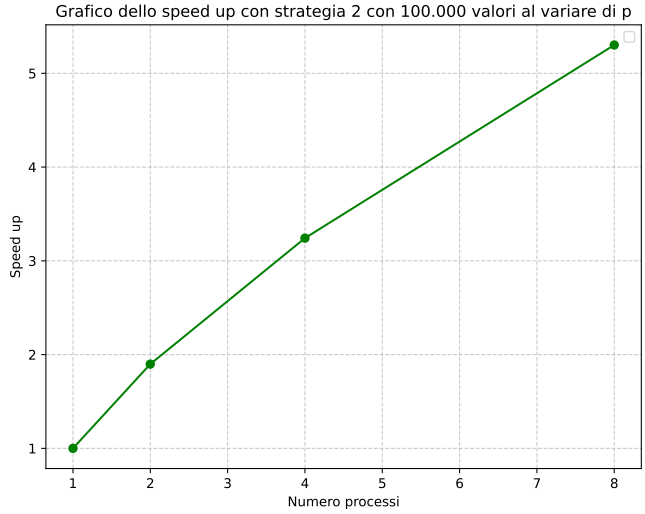


Figura 5.14: Plot *II strategia* dello speed up al variare del numero dei processori per $N = 10^5$

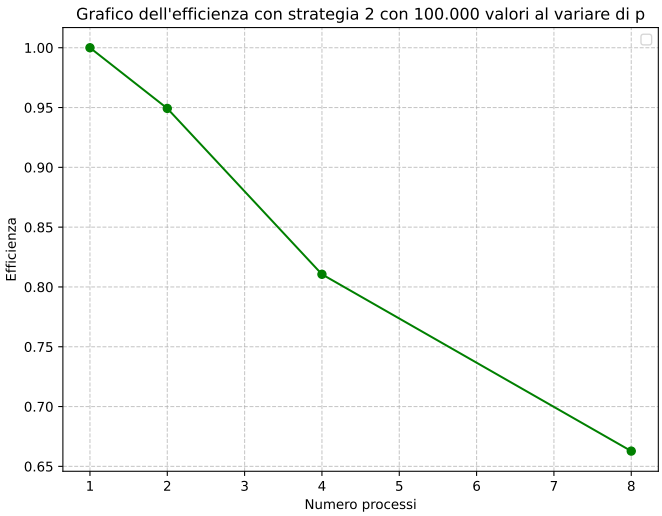


Figura 5.15: Plot *II strategia* dell'efficienza al variare del numero dei processori per $N = 10^5$

5.2.3 Analisi III strategia

N. processori	P1	P2	P4	P8
	4.529953e-04	2.179146e-04	1.449585e-04	9.012222e-05
	4.568100e-04	2.171993e-04	1.561642e-04	9.107590e-05
	4.518032e-04	2.160072e-04	1.609325e-04	9.012222e-05
	5.600452e-04	2.171993e-04	1.461506e-04	9.202957e-05
	3.919601e-04	2.191067e-04	1.528263e-04	9.894371e-05
	3.881454e-04	2.181530e-04	1.659393e-04	9.107590e-05
	4.680157e-04	2.219677e-04	1.530647e-04	10.01358e-05
	3.890991e-04	2.188683e-04	1.580715e-04	12.20703e-05
	3.869534e-04	2.181530e-04	1.652241e-04	10.01358e-05
	3.900528e-04	2.188683e-04	1.580710e-04	9.894370e-05
Tempo medio	4.3358802e-04	2.1834374e-04	1.5614027e-04	9.7465512e-05
Speed up	1	1.985804676	2.776913476581	4.448630198
Efficienza	1	0.992902338	0.694228369	0.55607877475

Tabella 5.6: Strategia III con $N = 10^5$

Figura 5.16: Plot *III strategia* del tempo al variare del numero dei processori per $N = 10^5$

Per quanto riguarda il tempo, questo è abbastanza simile alla strategia 2 peggiorando leggermente all'aumentare del numero dei processori in quanto vi sono più comunicazioni da fare. Per lo speed up e l'efficienza, queste sono piuttosto simili a quelli della strategia 2, con la differenza che con 2 processori ci si avvicina di più a speed up ed efficienza ideali. Ma questo potrebbe essere una casualità in quanto risultano abbastanza simili.

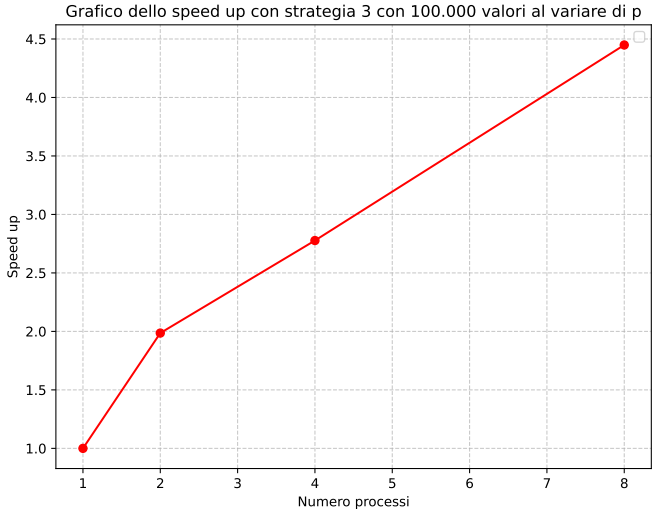


Figura 5.17: Plot *III strategia* dello speed up al variare del numero dei processori per $N = 10^5$

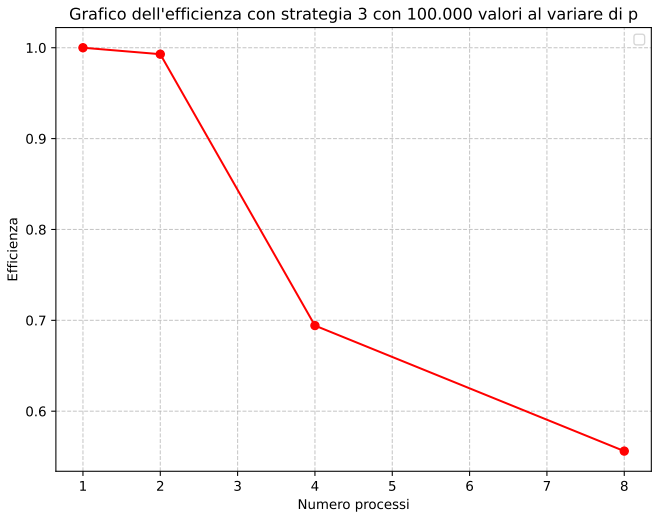


Figura 5.18: Plot *III strategia* dell'efficienza al variare del numero dei processori per $N = 10^5$

5.3 Analisi con *un milione*

5.3.1 Analisi I strategia

N. processori	P1	P2	P3	P4	P5	P6	P7	P8
	3.951073e-03	1.995087e-03	1.549006e-03	1.003981e-03	8.111000e-04	6.790161e-04	1.897407e-02	6.291866e-04
	4.580975e-03	1.997948e-03	1.559973e-03	1.011133e-03	8.070469e-04	6.799698e-04	1.909995e-02	5.128384e-04
	4.550934e-03	1.996994e-03	1.410007e-03	1.029968e-03	8.099079e-04	6.790161e-04	1.921701e-02	5.538464e-04
	4.202843e-03	1.991034e-03	1.410007e-03	1.006842e-03	8.080006e-04	6.790161e-04	1.879811e-02	5.578995e-04
	3.927946e-03	1.991034e-03	1.409054e-03	1.008034e-03	8.080006e-04	6.818771e-04	1.066303e-02	5.578995e-04
	3.957987e-03	1.993179e-03	1.424074e-03	1.008034e-03	8.108616e-04	6.818771e-04	1.911688e-02	5.140305e-04
	3.955126e-03	1.994133e-03	1.513004e-03	1.009941e-03	8.189678e-04	6.821156e-04	1.954699e-02	5.648136e-04
	3.928900e-03	1.992226e-03	1.421928e-03	1.008987e-03	8.099079e-04	6.811619e-04	1.919603e-02	5.629063e-04
	3.927946e-03	1.992941e-03	1.409054e-03	1.008987e-03	8.139610e-04	6.799698e-04	9.655952e-03	5.578995e-04
	3.926992e-03	2.002001e-03	1.415014e-03	1.008034e-03	8.158684e-04	6.809235e-04	1.927805e-02	5.559921e-04
Tempo medio	3.9728733e-03	1.9946577e-3	1.4521121e-3	1.0103941e-3	8.1136227e-4	6.8049431e-04	1.73546072e-2	5.5673124e-4
Speed up	1	1.99175693153	2.73592741222	3.93200366075	4.89654676696	5.83821678097	0.22892326252	7.13607035955
Efficienza	1	0.99587846576	0.91197580407	0.98300091518	0.97930935339	0.97303613016	0.03270332321	0.89200879494

Tabella 5.7: Strategia I con $N = 10^6$

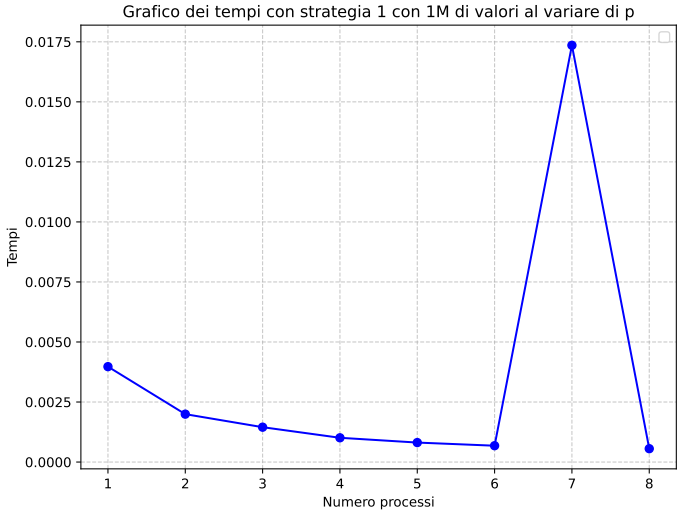


Figura 5.19: Plot *I strategia* del tempo al variare del numero dei processori per $N = 10^6$

Il tempo, ovviamente, diminuisce all'aumentare del numero dei processori. Lo speed up fino al 6 processore è vicino a quello ideale, mentre dal sesto in poi tende ad allontanarsi e questo si riflette anche sull'efficienza. Occorre notare che anche in questo caso con 7 processori si ottengono valori anomali.

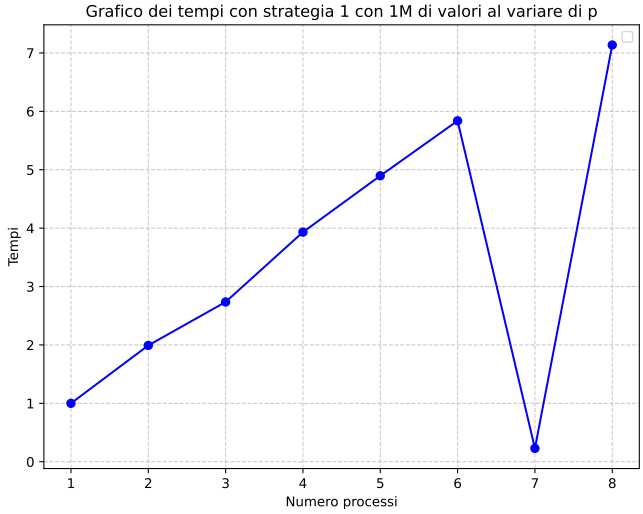


Figura 5.20: Plot *I strategia* dello speed up al variare del numero dei processori per $N = 10^6$

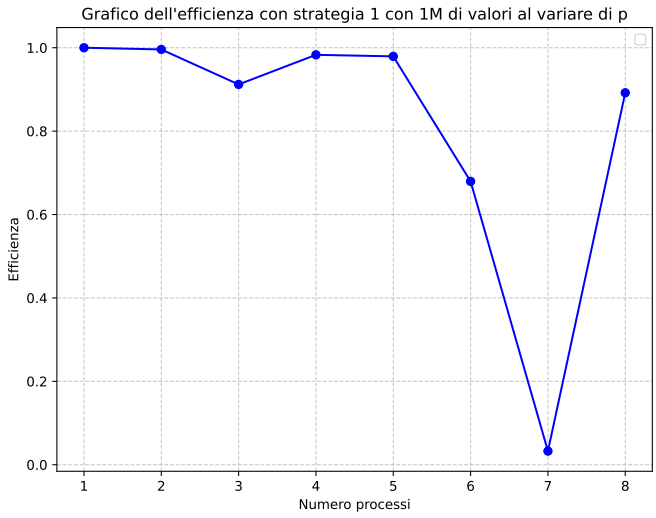


Figura 5.21: Plot *I strategia* dell'efficienza al variare del numero dei processori per $N = 10^6$

5.3.2 Analisi II strategia

N. processori	P1	P2	P4	P8
	3.951073e-03	2.301931e-03	1.014948e-03	5.300045e-04
	4.580975e-03	2.018929e-03	1.014948e-03	5.249977e-04
	4.550934e-03	2.010107e-03	1.013994e-03	5.278587e-04
	4.202843e-03	2.008200e-03	1.014948e-03	5.240440e-04
	3.927946e-03	2.043009e-03	1.017094e-03	5.240440e-04
	3.957987e-03	2.016068e-03	1.014948e-03	5.261898e-04
	3.955126e-03	2.002001e-03	1.014948e-03	5.259514e-04
	3.928900e-03	2.014160e-03	1.015902e-03	5.259514e-04
	3.927946e-03	2.074957e-03	1.009941e-03	5.259514e-04
	3.926992e-03	2.324104e-03	1.015902e-03	5.249977e-04
Tempo medio	3.9728733e-03	2.0813466e-03	1.0147573e-03	5.2626344e-04
Speed up	1	1.90879947626	3.91509703847	7.54921014464
Efficienza	1	0.95439973813	0.97877425961	0.94365126808

Tabella 5.8: Strategia II con $N = 10^6$

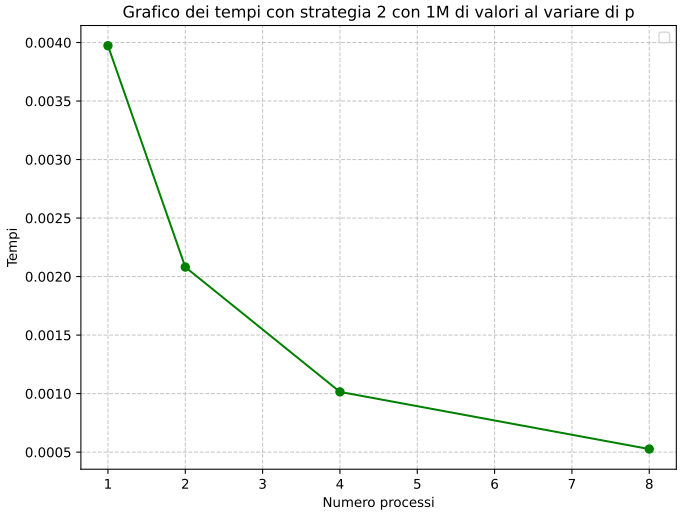


Figura 5.22: Plot *II strategia* del tempo al variare del numero dei processori per $N = 10^6$

Tempo e speed up risultano coerenti con l'obiettivo di diminuire il tempo all'aumentare del numero dei processori con speed up prossimi a quelli ideali. Per l'efficienza, pure abbiamo valori prossimi a quelli ideali, con un leggero calo con 2 e 8 processori implicando che con 4 si ha un'efficienza migliore, ergo è preferibile utilizzare 4 processori per ottenere prestazioni ottimali senza sprecare risorse.

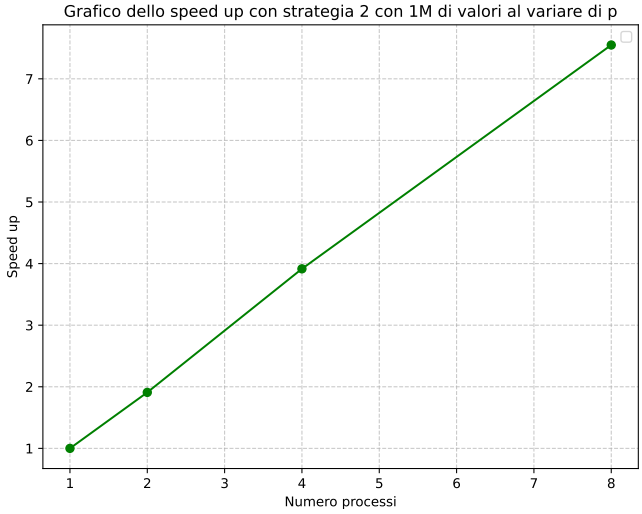


Figura 5.23: Plot *II strategia* dello speed up al variare del numero dei processori per $N = 10^6$

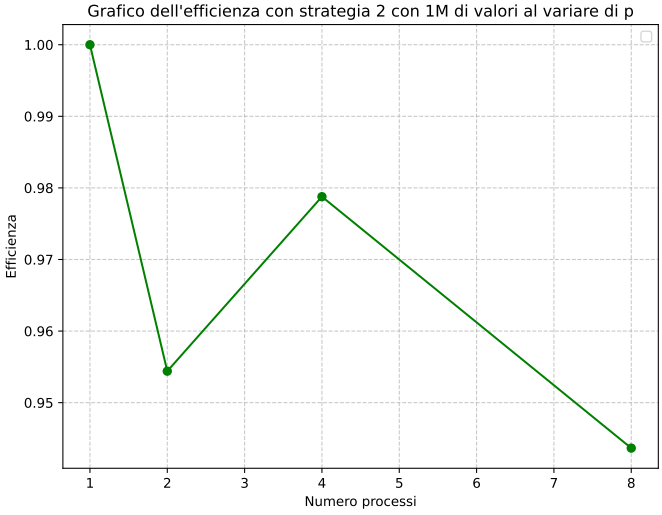


Figura 5.24: Plot *II strategia* dell'efficienza al variare del numero dei processori per $N = 10^6$

5.3.3 Analisi III strategia

N. processori	P1	P2	P4	P8
	3.951073e-03	2.278090e-03	1.206875e-03	6.599426e-04
	4.580975e-03	2.353191e-03	1.193047e-03	5.409718e-04
	4.550934e-03	2.027988e-03	1.031876e-03	5.989075e-04
	4.202843e-03	2.024889e-03	1.188040e-03	6.129742e-04
	3.927946e-03	2.382994e-03	1.184940e-03	5.331039e-04
	3.957987e-03	2.061129e-03	1.178980e-03	5.331039e-04
	3.955126e-03	2.059937e-03	1.235008e-03	5.328655e-04
	3.928900e-03	2.330065e-03	1.216888e-03	5.388260e-04
	3.927946e-03	2.066851e-03	1.184940e-03	5.340576e-04
	3.926992e-03	2.279043e-03	1.066923e-03	5.331039e-04
Tempo medio	3.9728733e-03	2.1254087e-03	1.1687517e-03	5.6178569e-04
Speed up	1	1.86920896	3.70457565	7.70708328
Efficienza	1	0.93460448	0.92614391	0.96338541

Tabella 5.9: Strategia III con $N = 10^6$

Figura 5.25: Plot *III strategia* del tempo al variare del numero dei processori per $N = 10^6$

Per la terza vale lo stesso discorso fatto prima per la seconda con la differenza che il tempo è leggermente maggiore (sempre a causa delle comunicazioni), mentre l'efficienza risulta "stranamente" migliore con 8 processori. Stranamente perchè essendo quella precedente più efficiente con 4 è strano che, in questo caso, con 8 sia migliore a causa delle maggiori comunicazioni da fare.

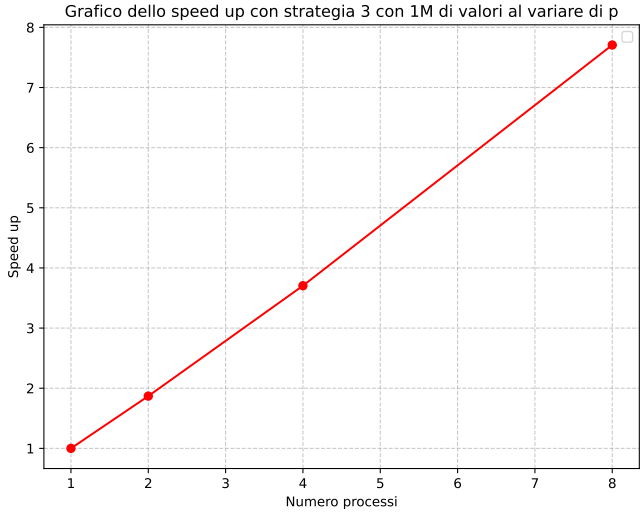


Figura 5.26: Plot *III strategia* dello speed up al variare del numero dei processori per $N = 10^6$

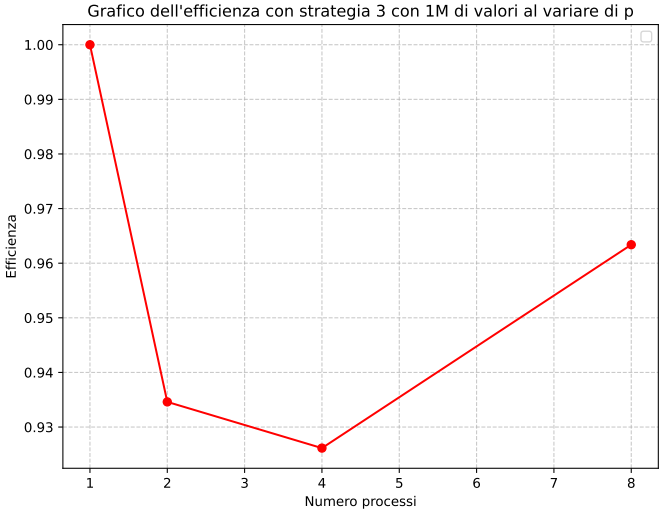


Figura 5.27: Plot *III strategia* dell'efficienza al variare del numero dei processori per $N = 10^6$

5.4 Analisi con *dieci milioni*

5.4.1 Analisi I strategia

N. processori	P1	P2	P3	P4	P5	P6	P7	P8
	3.959417e-02	2.048206e-02	1.330519e-02	9.963989e-03	8.024931e-03	6.659985e-03	1.773810e-02	5.017996e-03
	3.959513e-02	2.089810e-02	1.324797e-02	9.962082e-03	8.026123e-03	7.974148e-03	1.703596e-02	5.033016e-03
	3.962588e-02	1.985812e-02	1.326704e-02	9.954214e-03	8.033991e-03	6.703854e-03	2.257109e-02	5.146027e-03
	3.960299e-02	1.984787e-02	1.329994e-02	9.952068e-03	8.026123e-03	6.660938e-03	1.767898e-02	5.839825e-03
	3.962207e-02	1.988387e-02	1.329780e-02	9.954929e-03	7.977962e-03	6.654024e-03	1.712394e-02	5.017996e-03
	3.963208e-02	1.985788e-02	1.327300e-02	9.959936e-03	8.025885e-03	6.657124e-03	1.716113e-02	5.131006e-03
	3.962302e-02	1.984787e-02	1.327586e-02	9.985924e-03	7.982016e-03	6.657124e-03	2.614689e-02	5.027056e-03
	3.994894e-02	1.988387e-02	1.326799e-02	9.990931e-03	8.023024e-03	6.659985e-03	1.741099e-02	5.017042e-03
	3.959608e-02	1.985502e-02	1.325893e-02	9.963036e-03	8.044004e-03	6.639004e-03	1.701713e-02	5.046129e-03
	3.953505e-02	1.984811e-02	1.326299e-02	9.944916e-03	8.029938e-03	6.647825e-03	2.579594e-02	5.015135e-03
Tempo medio	3.9637541e-02	2.0026277e-02	1.3275671e-02	9.9632025e-03	8.0193997e-03	6.7914011e-03	1.9568015e-02	5.1291228e-03
Speed up	1	1.97927657747	2.985727877	3.97839359383	4.94270674	5.83643057	2.02562911	7.72793761148
Efficienza	1	0.98963828873	0.9952426256	0.99459839845	0.988541348	0.9727384283	0.28937558	0.96599220143

Tabella 5.10: Strategia I con $N = 10^7$

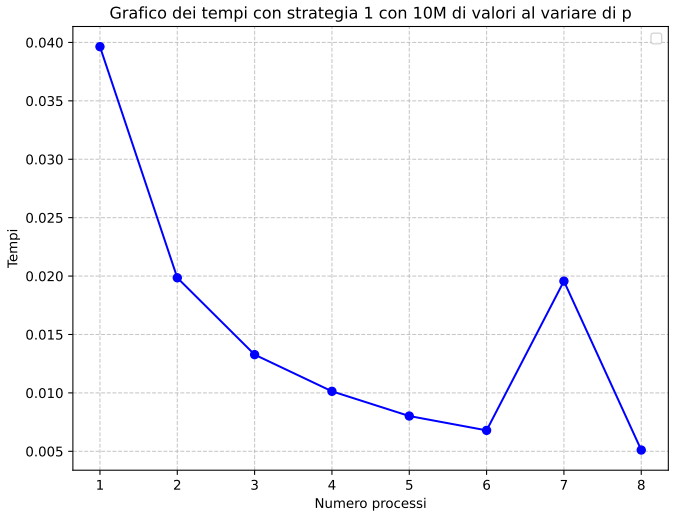


Figura 5.28: Plot *I strategia* del tempo al variare del numero dei processori per $N = 10^7$

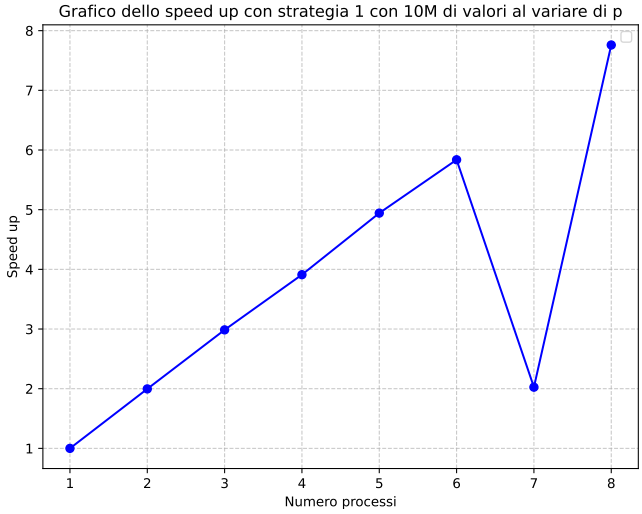


Figura 5.29: Plot *I strategia* dello speed up al variare del numero dei processori per $N = 10^7$

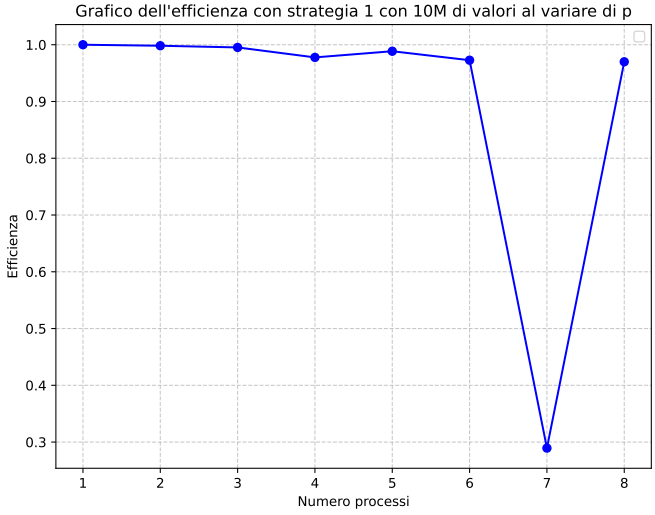


Figura 5.30: Plot *I strategia* dell'efficienza al variare del numero dei processori per $N = 10^7$

5.4.2 Analisi II strategia

N. processori	P1	P2	P4	P8
	3.959417e-02	2.048206e-02	9.963989e-03	5.017996e-03
	3.959513e-02	2.089810e-02	9.962082e-03	5.033016e-03
	3.962588e-02	1.985812e-02	9.954214e-03	5.146027e-03
	3.960299e-02	1.984787e-02	9.952068e-03	5.839825e-03
	3.962207e-02	1.988387e-02	9.954929e-03	5.017996e-03
	3.963208e-02	1.985788e-02	9.959936e-03	5.131006e-03
	3.962302e-02	1.984787e-02	9.985924e-03	5.027056e-03
	3.994894e-02	1.988387e-02	9.990931e-03	5.017042e-03
	3.959608e-02	1.985502e-02	9.963036e-03	5.046129e-03
	3.953505e-02	1.984811e-02	9.944916e-03	5.015135e-03
Tempo medio	3.9637541e-02	2.0026277e-02	9.9632025e-03	5.1291228e-03
Speed up	1	1.97927657747	3.97839359383	7.72793761148
Efficienza	1	0.98963828873	0.99459839845	0.96599220143

Tabella 5.11: Strategia II con $N = 10^7$

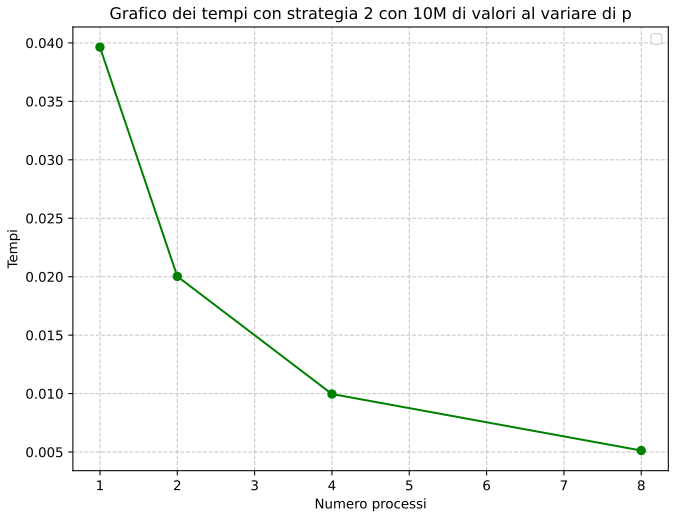


Figura 5.31: Plot *II strategia* del tempo al variare del numero dei processori per $N = 10^7$

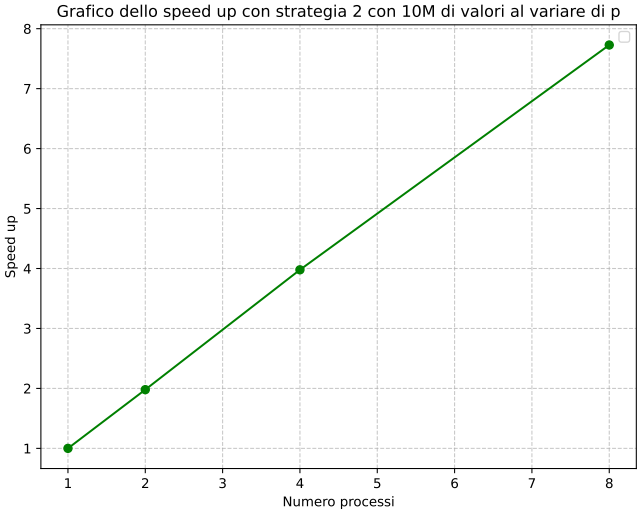


Figura 5.32: Plot *II strategia* dello speed up al variare del numero dei processori per $N = 10^7$

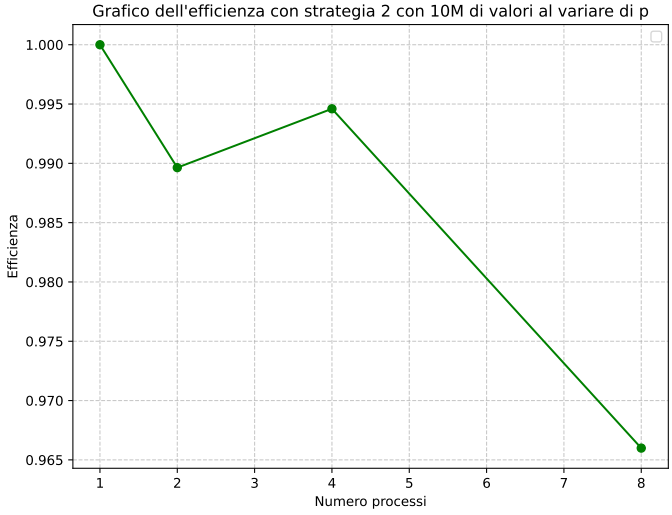


Figura 5.33: Plot *II strategia* dell'efficienza al variare del numero dei processori per $N = 10^7$

5.4.3 Analisi III strategia

N. processori	P1	P2	P4	P8
	3.959417e-02	1.996303e-02	9.971142e-03	5.018950e-03
	3.959513e-02	1.987791e-02	9.980917e-03	5.026102e-03
	3.962588e-02	1.988316e-02	9.977818e-03	5.029917e-03
	3.960299e-02	1.989198e-02	9.979963e-03	5.047083e-03
	3.962207e-02	1.987410e-02	9.982109e-03	5.023003e-03
	3.963208e-02	1.986694e-02	10.01406e-03	5.033016e-03
	3.962302e-02	1.987600e-02	9.973049e-03	5.028009e-03
	3.994894e-02	1.989913e-02	9.979963e-03	5.763054e-03
	3.959608e-02	1.990294e-02	9.986162e-03	5.147934e-03
	3.953505e-02	1.986909e-02	9.968996e-03	5.150080e-03
Tempo medio	3.9637541e-02	1.9890428e-02	9.9814179e-03	5.1267148e-03
Speed up	1	1.9927947754	3.97113329961	7.7315673967
Efficienza	1	0.9963973877	0.9927833249	0.9664459245

Tabella 5.12: Strategia III con $N = 10^7$

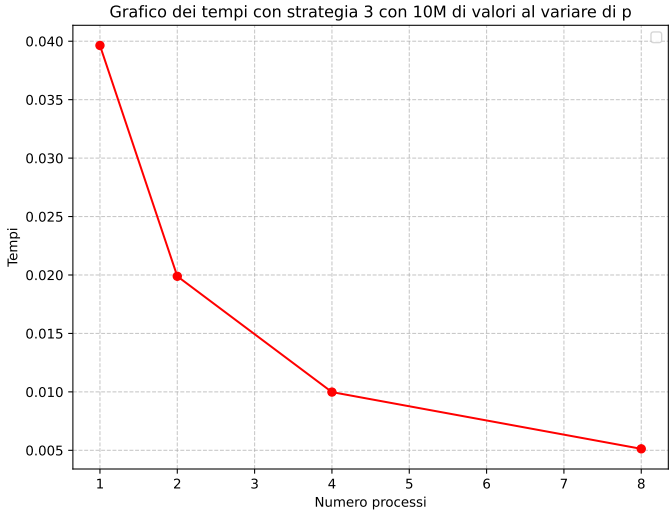


Figura 5.34: Plot *III strategia* del tempo al variare del numero dei processori per $N = 10^7$

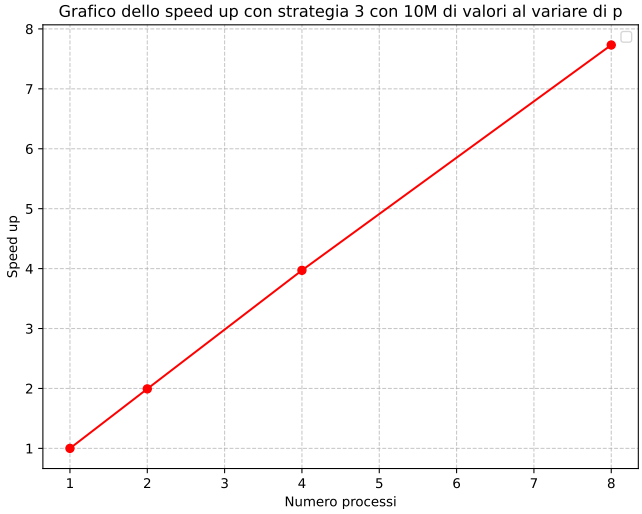


Figura 5.35: Plot *III strategia* dello speed up al variare del numero dei processori per $N = 10^7$

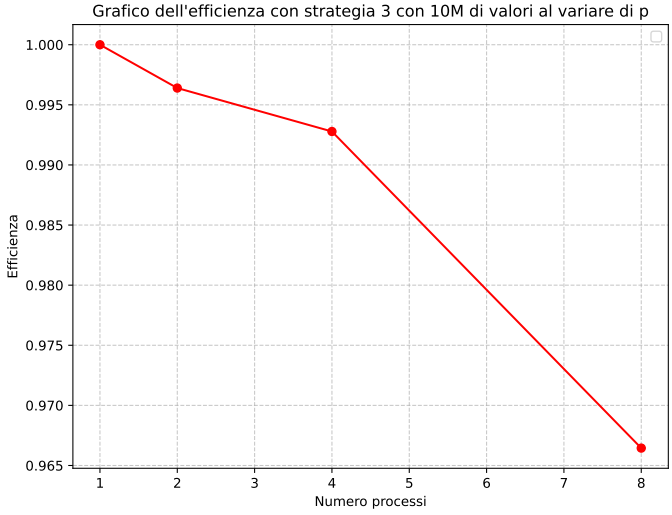


Figura 5.36: Plot *III strategia* dell'efficienza al variare del numero dei processori per $N = 10^7$

5.4.4 Confronto fra strategie

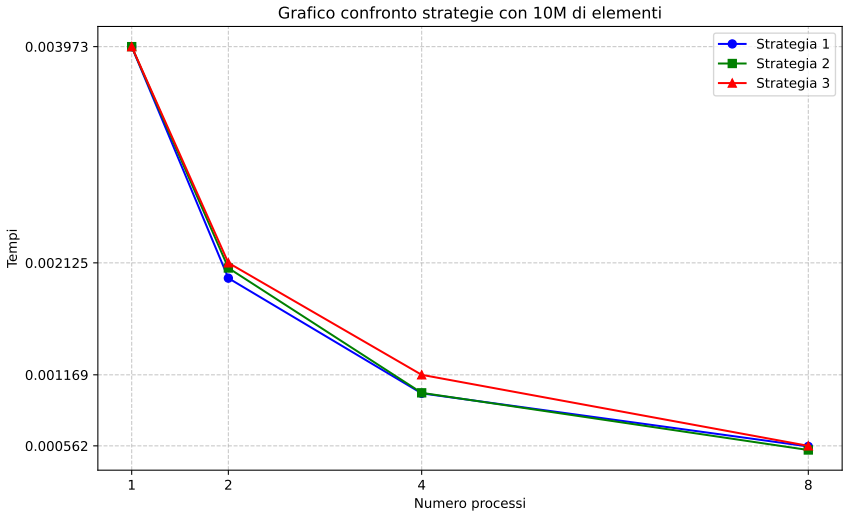


Figura 5.37: Plot della differenza fra strategie in termini di tempo al variare del numero dei processori per $N = 10^7$

Infine, con 10 milioni di valori possiamo notare che i tempi siano piuttosto simili (con la terza strategia che impiega di più, come ci aspettiamo a causa del numero elevato di comunicazioni). Ovviamente questo si riflette sullo speed up in quanto abbiamo valori prossimi a quelli ideali. Per l'efficienza pure abbiamo valori prossimi a quelli ideali con la differenza che con la prima strategia è più efficiente con 2 e 8 processori, mentre per la seconda e terza abbiamo differenze centesimali.

In definitiva: per input alti la strategia adottata è abbastanza ininfluente, mentre per input più bassi risulta evidente l'overhead all'aumentare del numero dei processori.

Con $N > 10^6$ otteniamo, quindi, sempre di più speed up prossimi a quello ideale, e quindi anche più efficienti.

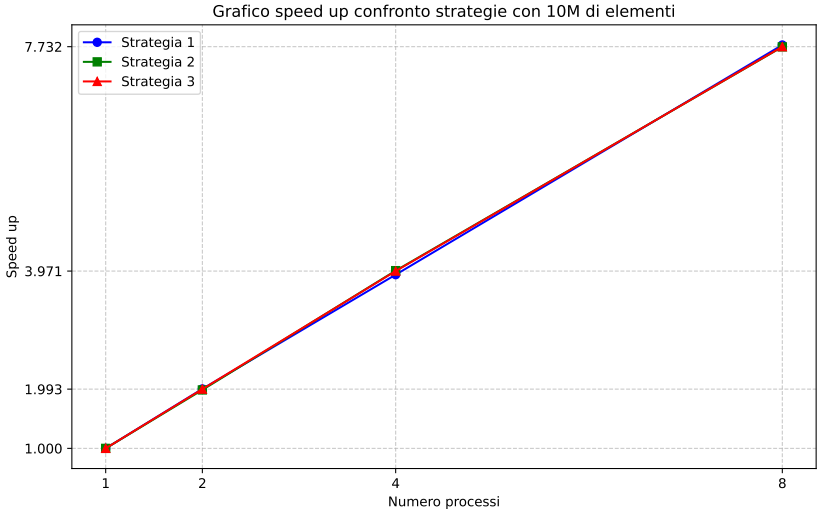


Figura 5.38: Plot della differenza fra strategie in termini di speed up al variare del numero dei processori per $N = 10^7$

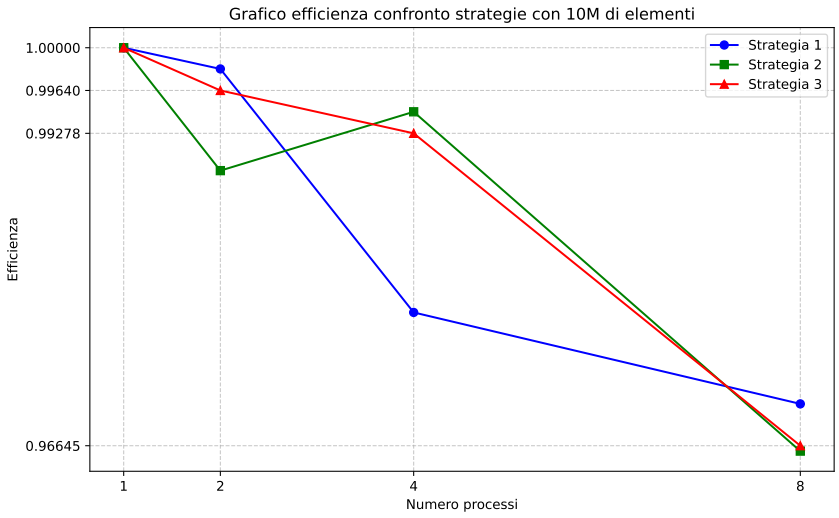


Figura 5.39: Plot della differenza fra strategie in termini di efficienza al variare del numero dei processori per $N = 10^7$

5.5 Analisi *N variabile*

5.5.1 Analisi I strategia

N. processori	**P2** $N_0 = 10^5$	**P4** $N_1 = 4 * 10^5$	**P8** $N_1 = 1.2 * 10^6$
	2.241135e-04	4.251003e-04	6.508827e-04
	2.119541e-04	4.239082e-04	6.132126e-04
	2.100468e-04	4.119873e-04	6.539822e-04
	2.140999e-04	4.761219e-04	7.081032e-04
	2.431870e-04	4.758835e-04	6.539822e-04
	2.140999e-04	4.148483e-04	6.539822e-04
	2.138615e-04	4.789829e-04	6.580353e-04
	2.100468e-04	4.179478e-04	6.160736e-04
	2.110004e-04	4.131794e-04	6.539822e-04
	2.331734e-04	4.110336e-04	6.141663e-04
Tempo medio	2.1855833e-04	4.3489932e-04	6.4764025e-04
Speed up Scalato	1	2.01019702675	4.04962471063
Efficienza Scalata	0.5	0.50254925668	0.50620308882

Tabella 5.13: Strategia I con N Scalato

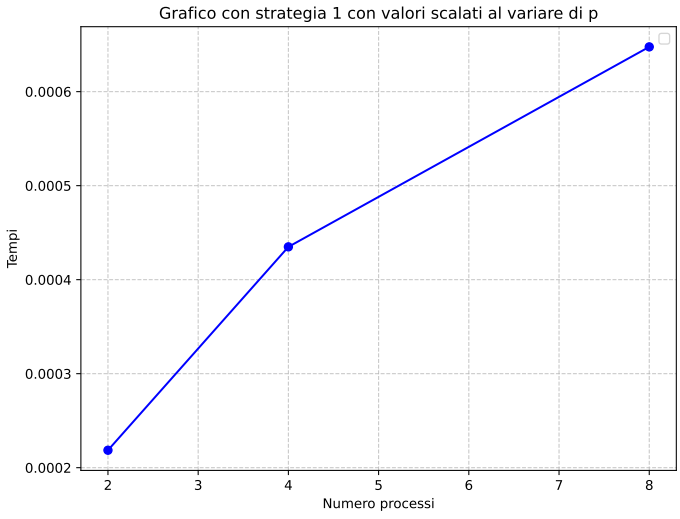


Figura 5.40: Plot *I strategia* del tempo scalato

Figura 5.41: Plot *I strategia* dello speed up scalato

Figura 5.42: Plot *I strategia* dell'efficienza scalata

5.5.2 Analisi II strategia

N. processori	**P2** $N_0 = 5 * 10^5$	**P4** $N_1 = 2 * 10^6$	**P8** $N_1 = 6 * 10^6$
	1.007080e-03	2.012968e-03	3.014088e-03
	1.013041e-03	2.003908e-03	3.017902e-03
	1.282930e-03	2.005100e-03	3.011942e-03
	1.021147e-03	2.004147e-03	3.012896e-03
	1.004934e-03	2.014875e-03	3.015995e-03
	1.011133e-03	2.004147e-03	3.010988e-03
	1.003981e-03	2.017021e-03	3.010035e-03
	1.003027e-03	2.011061e-03	3.015995e-03
	1.004934e-03	2.007008e-03	3.808975e-03
	1.003981e-03	2.011061e-03	3.010988e-03
Tempo medio	1.0356188e-03	2.0095349e-03	3.0929804e-03
Speed up Scalato	1	2.06182577769	4.0179451509
Efficienza Scalata	0.5	0.51545644442	0.50224314386

Tabella 5.14: Strategia II con N Scalato

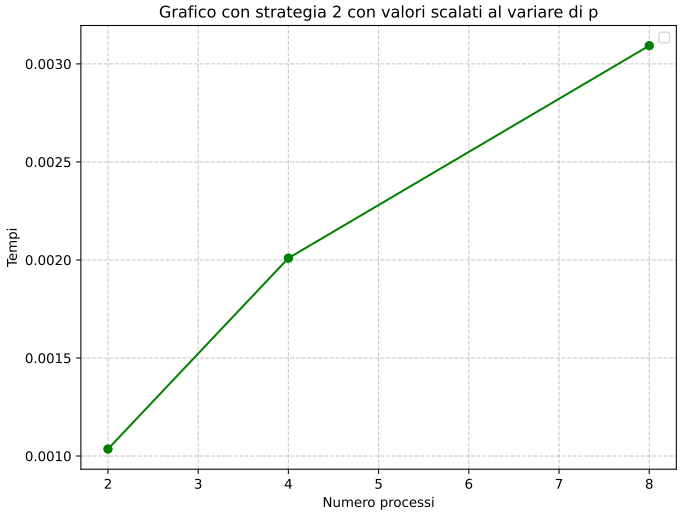


Figura 5.43: Plot *II strategia* del tempo scalato

Figura 5.44: Plot *II strategia* dello speed up scalato

Figura 5.45: Plot *II strategia* dell'efficienza scalata

5.5.3 Analisi III strategia

N. processori	**P2** $N_0 = 10^6$	**P4** $N_1 = 4 * 10^6$	**P8** $N_1 = 12 * 10^6$
	1.996040e-03	4.653931e-03	6.064177e-03
	2.006054e-03	4.215002e-03	6.029129e-03
	2.037048e-03	4.836798e-03	6.025076e-03
	2.161980e-03	5.012035e-03	6.026983e-03
	2.004147e-03	4.010916e-03	6.025076e-03
	2.214909e-03	4.011154e-03	6.021023e-03
	2.218008e-03	4.009008e-03	6.032944e-03
	2.038956e-03	4.009962e-03	6.030083e-03
	2.336979e-03	4.038095e-03	6.048918e-03
	2.634048e-03	4.035950e-03	6.032944e-03
Tempo medio	2.1648169e-03	4.2832851e-03	6.0336353e-03
Speed up Scalato	1	2.02164166004	4.3054976823
Efficienza Scalata	0.5	0.50541041501	0.53818721028

Tabella 5.15: Strategia III con N Scalato

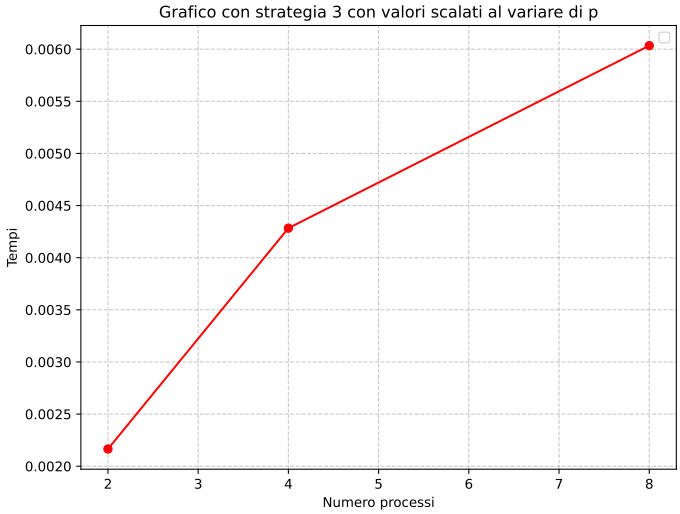


Figura 5.46: Plot *III strategia* del tempo scalato

Figura 5.47: Plot *III strategia* dello speed up scalato

Figura 5.48: Plot *III strategia* dell'efficienza scalata

5.5.4 Confronto fra strategie

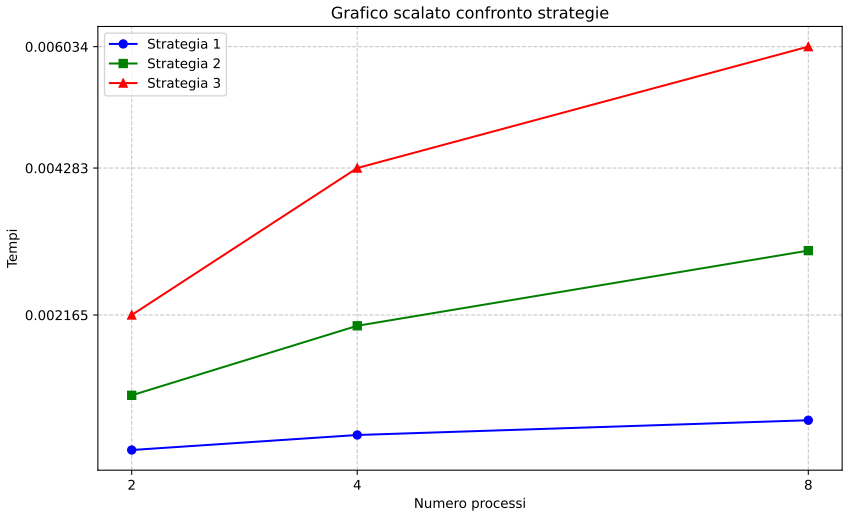


Figura 5.49: Plot della differenza fra strategie in termini di tempo scalato

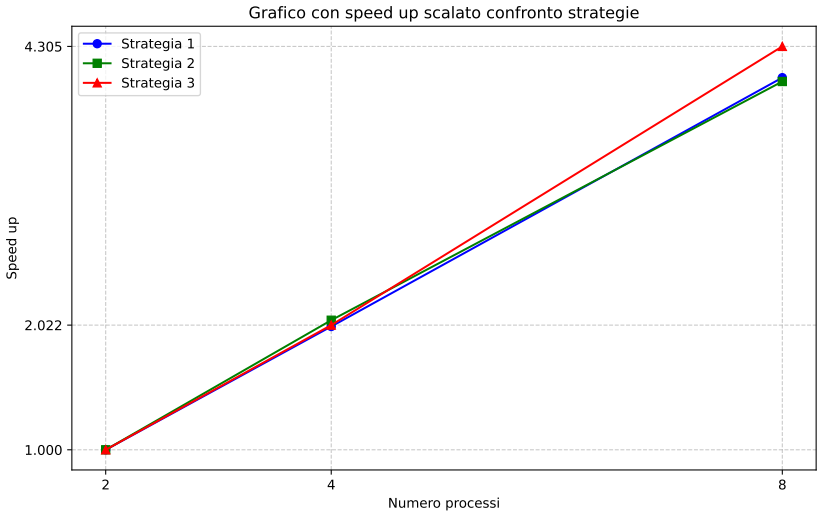


Figura 5.50: Plot della differenza fra strategie in termini di speed up scalato

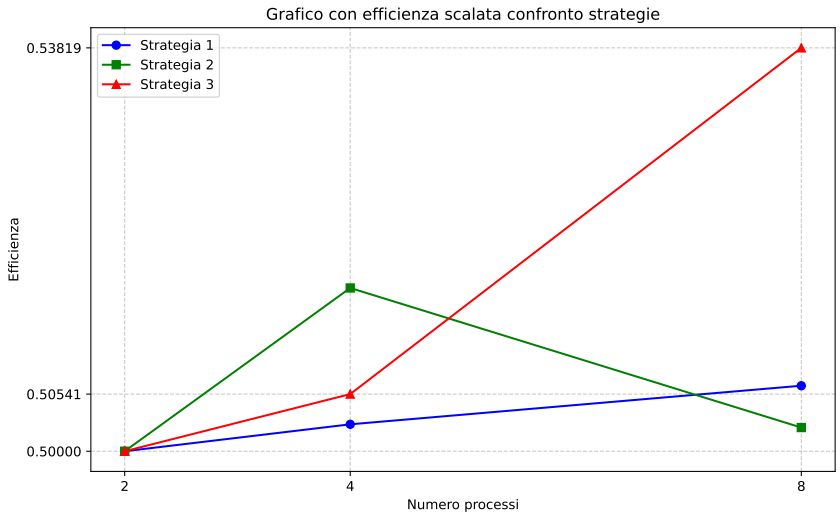


Figura 5.51: Plot della differenza fra strategie in termini di efficienza scalata

Nel caso scalato, invece, risulta abbastanza evidente come tempi, speed up ed efficienza siano costanti all'aumentare del numero dei processori e della quantità dei valori da sommare. Questo perchè l'N viene calcolato proprio in base al numero dei processori che vorremmo usare in parallelo.

Capitolo 6

Source code

6.1 *Main.c*

```
1  /**
2   * @author Fabrizio Vitale
3   * @author Giovanni Falcone
4   * @author Luigi Mangiacapra
5  */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10
11 #include "mpi.h"
12 #include "Strategy.h"
13 #include "Utils.h"
14
15 # define STRATEGY_1 1
16 # define STRATEGY_2 2
17 # define STRATEGY_3 3
18
19 int main(int argc, char *argv[]){
20     int menum;                      // id del processore
21     int nproc;                      // numero processori
22     int N;                          // numero di elementi da sommare
23     int sum;                        // somma totale da stampare
24     int logNproc;                   // numero di passi da effettuare ↩
       per la II, III strategia
25     int strategy;                   // strategia con cui sommare
26     int nloc;                       // numero di elementi che ciascun ↩
       processore deve sommare
27     int rest;                       // resto della divisione
28     int *elements;                  // array completo
29     int *elements_loc;              // vettore di elementi locale
30     int *array_of_powers_of_two;    // vettore di potenze di 2
31     double end_time;
32     double start_time;
33     double timetot = 0;
34
```

```
35      if(argc < 3){
36          fprintf(stderr, "Utilizzo: <numeri da sommare> <tipo di ↩
        strategia> <numeri da sommare se N>\n");
37          return EXIT_FAILURE;
38      }
39
40      // MPI initialization
41      MPI_Init(&argc, &argv);
42      MPI_Comm_rank(MPI_COMM_WORLD, &menum);
43      MPI_Comm_size(MPI_COMM_WORLD, &nproc);
44
45      if(menum == 0){
46          // convert to integer the number to sum and strategy to apply
47          N = atoi(argv[1]);
48          strategy = atoi(argv[2]);
49
50          if(check_if_inputs_are_valid(argc, N, strategy) != 0){
51              printf("Input non valido: chiusura del programma.\n");
52              MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
53          }
54
55          // array malloc
56          elements = (int *)malloc(sizeof(int) * N);
57          if(elements == NULL){
58              fprintf(stderr, "Errore nell'allocazione della memoria per ↩
        l'array 'elements'!\n");
59              return EXIT_FAILURE;
60          }
61
62          // fill array with N elements
63          fill_array(elements, N, argv);
64
65          // Verifica se la strategia 2 (o 3) e' applicabile: se il ↩
        numero dei processori non e' potenza di 2 applica la strategia 1
66          if(!strategy_2_OR_3_are_applicable(strategy, nproc)){
67              strategy = STRATEGY_1;
68              printf("Applico la prima strategia.\n");
69          }
70      }
71
72      // send data to all other processors
73      MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
74      MPI_Bcast(&strategy, 1, MPI_INT, 0, MPI_COMM_WORLD);
75
76      // compute the logarithm and powers of two for second and third ↩
        strategy
77      if(strategy == STRATEGY_2 || strategy == STRATEGY_3) {
78          // get number of steps
79          logNproc = log2(nproc);
80          // allocation of array of powers of 2
81          array_of_powers_of_two = (int *)malloc(sizeof(int) * logNproc);
82          if(array_of_powers_of_two == NULL){
83              fprintf(stderr, "Errore nell'allocazione della memoria per ↩
        l'array 'array_of_powers_of_two'!\n");
```

```
84              return EXIT_FAILURE;
85          }
86          // fill the array with powers of 2
87          compute_power_of_two(logNproc, array_of_powers_of_two);
88      }
89
90
91      // in order to check how many elements each processor must sum
92      nloc = N / nproc;
93      rest = N % nproc;
94
95      if(menum < rest){
96          nloc = nloc + 1;
97      }
98
99      // allocation of local array for each processor
100     elements_loc = (int *)malloc(sizeof(int) * nloc);
101     if(elements_loc == NULL){
102         fprintf(stderr, "Errore nell'allocazione della memoria per l'↩
        array 'elements'!\n");
103         return EXIT_FAILURE;
104     }
105
106     // invia elementi da sommare agli altri processori
107     operand_distribution(menum, elements, elements_loc, nloc, nproc, ↩
        rest);
108
109     // attendiamo che i processi si sincronizzino
110     MPI_Barrier(MPI_COMM_WORLD);
111     start_time = MPI_Wtime();
112
113     sum = 0;
114     // first step: each processor performs the first partial sum
115     sum = sequential_sum(elements_loc, nloc);
116
117     // check the strategy to apply
118     if(strategy == STRATEGY_1){
119         sum = first_strategy(menum, nproc, sum);
120     }else if(strategy == STRATEGY_2){
121         sum = second_strategy(menum, logNproc, array_of_powers_of_two, ↩
        sum);
122     } else{ // third_strategy
123         sum = third_strategy(menum, logNproc, array_of_powers_of_two, ↩
        sum);
124     }
125
126     end_time = MPI_Wtime();
127
128     double timeP = end_time - start_time;
129     printf("Il tempo impiegato da %d e' di %e s\n", menum, timeP);
130
131     // compute total time
132     MPI_Reduce(&timeP, &timetot, 1, MPI_DOUBLE, MPI_MAX, 0, ↩
        MPI_COMM_WORLD);
```

```
133
134     // print sum total and partial sum of each processor and the time
135     print_result(menum, strategy, sum, timetot);
136
137     // freeing memory before program termination
138     if(menum == 0){
139         free(elements);
140         free(elements_loc);
141         free(array_of_powers_of_two);
142     }else{
143         free(elements_loc);
144         free(array_of_powers_of_two);
145     }
146
147     MPI_Finalize();
148     return 0;
149 }
```

6.2 *Strategy.c*

```
1  /**
2   * @author Fabrizio Vitale
3   * @author Giovanni Falcone
4   * @author Luigi Mangiacapra
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <time.h>
10 #include <math.h>
11
12 #include "mpi.h"
13
14
15 int first_strategy(int menum, int nproc, int sum){
16     int sum_parz = 0;
17     int tag;
18     MPI_Status status;
19
20     if(menum == 0){
21         for(int i = 1; i < nproc; i++){
22             tag = 80 + i;
23             MPI_Recv(&sum_parz, 1, MPI_INT, i, tag, MPI_COMM_WORLD, &↩
       status);
24             sum += sum_parz;
25         }
26     }else{
27         tag = menum + 80;
28         MPI_Send(&sum, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
29     }
30
31     return sum;
```

```
32 }
33
34 int second_strategy(int menum, int logNproc, int *array, int sum){
35     int sum_parz = 0;
36     int tag;
37     int partner;
38
39     int power_for_partecipation;
40     int does_processor_partecipate;
41
42     int power_for_communication;
43     int does_processor_receive;
44
45     MPI_Status status;
46
47     for(int i = 0; i < logNproc; i++){
48         power_for_partecipation =  array[i];
49         does_processor_partecipate = (menum % power_for_partecipation) ↩
    == 0;
50
51         if(does_processor_partecipate){
52             power_for_communication = array[i + 1];
53             does_processor_receive = (menum % power_for_communication) ↩
    == 0;
54
55             if (does_processor_receive){
56                 partner = menum + power_for_partecipation;
57                 tag = 60 + i;
58                 MPI_Recv(&sum_parz, 1, MPI_INT, partner, tag, ↩
    MPI_COMM_WORLD, &status);
59                 sum += sum_parz;
60             }
61             else{
62                 partner = menum - power_for_partecipation;
63                 tag = 60 + i;
64                 MPI_Send(&sum, 1, MPI_INT, partner, tag, MPI_COMM_WORLD↩
    );
65             }
66         }
67     }
68
69     return sum;
70 }
71
72 int third_strategy(int menum, int logNproc, int *array, int sum){
73     int partner;
74     int send_tag;
75     int recv_tag;
76     int sum_parz;
77     MPI_Status status;
78
79     sum_parz = 0;
80     for(int i = 0; i < logNproc; i++){
81         if ((menum % array[i + 1]) < array[i]) {
```

```
82              partner = menum + array[i];
83              send_tag = 40 + i;
84              recv_tag = 40 + i;
85
86              // Invia la somma locale al processo partner
87              MPI_Send(&sum, 1, MPI_INT, partner, send_tag, ↩
    MPI_COMM_WORLD);
88
89              // Ricevi la somma del processo partner
90              MPI_Recv(&sum_parz, 1, MPI_INT, partner, recv_tag, ↩
    MPI_COMM_WORLD, &status);
91
92              // Aggiorna la variabile 'sum' con la somma ricevuta
93              sum += sum_parz;
94          } else {
95              partner = menum - array[i];
96              send_tag = 40 + i;
97              recv_tag = 40 + i;
98
99              // Ricevi la somma dal processo partner
100             MPI_Recv(&sum_parz, 1, MPI_INT, partner, recv_tag, ↩
    MPI_COMM_WORLD, &status);
101
102             // Invia la somma locale al processo partner
103             MPI_Send(&sum, 1, MPI_INT, partner, send_tag, ↩
    MPI_COMM_WORLD);
104             sum += sum_parz;
105         }
106     }
107
108     return sum;
109 }
```

6.3 *Strategy.h*

```
1  #ifndef STRATEGY_H
2  #define STRATEGY_H
3
4  /**
5   * @brief apply the first strategy
6   *
7   * @param menum id of the processor
8   * @param nproc number of processor
9   * @param sum partial sum performed at the first step
10  * @return int total sum
11  */
12 int first_strategy(int menum, int nproc, int sum);
13
14 /**
15  * @brief apply the second strategy
16  *
17  * @param menum id of the processor
```

```
 * @param logNproc number of steps
 * @param array the array of powers of two
 * @param sum  partial sum performed at the first step
 * @return int total sum
 */
int second_strategy(int menum, int logNproc, int *array, int sum);

/**
 * @brief apply the third strategy
 *
 * @param menum id of the processor
 * @param logNproc number of steps
 * @param array the array of powers of two
 * @param sum  partial sum performed at the first step
 * @return int total sum
 */
int third_strategy(int menum, int logNproc, int *array, int sum);

#endif
```

6.4 *Utils.c*

```
/**
 * @author Fabrizio Vitale
 * @author Giovanni Falcone
 * @author Luigi Mangiacapra
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

# include "mpi.h"

/* ************************************************************** */
/*                        SUPPORT FUNCTION                        */
/* ************************************************************** */

static void fill_array_randomly(int *elements, int N);

static void fill_array_by_argv(int *elements, int N, char *argv[]);

/**
 * @brief check if strategy is a number between 1 and 3
 *
 * @param strategy the integer
 *
 * @return 0 if it's valid, 1 otherwise
 *
*/
static int strategy_is_valid(int strategy);
```

```
31
32 /* ************************************************************** */
33 /* ************************************************************** */
34 /* ************************************************************** */
35
36 int check_if_inputs_are_valid(int argc, int N, int strategy){
37     if(N <= 0){
38         fprintf(stderr, "Inserire un numero maggiore di 0!\n");
39         return EXIT_FAILURE;
40     }
41
42     if(strategy_is_valid(strategy) != 0){
43         fprintf(stderr, "Strategia non valida: inserire un numero ↩
    compreso fra 1 e 3!\n");
44         return EXIT_FAILURE;
45     }
46
47     if(N <= 20 && argc - 3 != N){
48         fprintf(stderr, "Il numero di elementi inserito non corrisponde↩
     ad N!\n");
49         return EXIT_FAILURE;
50     }
51
52     return 0; // valid input
53 }
54
55 int strategy_is_valid(int strategy){
56     if(strategy < 1 || strategy > 3)
57         return EXIT_FAILURE;
58
59     // it's valid
60     return 0;
61 }
62
63 void fill_array(int *elements, int N, char *argv[]){
64     if(N > 20)
65         fill_array_randomly(elements, N);
66     else
67         fill_array_by_argv(elements, N, argv);
68 }
69
70 void fill_array_randomly(int *elements, int N){
71     srand(time(NULL));
72
73     printf("Generazione numeri randomici...\n");
74
75     for(int i = 0; i < N; i++){
76         elements[i] = rand() % 100;
77     }
78 }
79
80 void fill_array_by_argv(int *elements, int N, char *argv[]){
81     printf("Inserimento dei numeri forniti da terminale...\n");
82
```

```
83      for(int i = 0; i < N; i++){
84          elements[i] = atoi(argv[i + 3]);    // 0: name src; 1: N; 2: ↩
    strategy; starting from 3 we have all numbers
85      }
86  }
87
88  int strategy_2_OR_3_are_applicable(int strategy, int nproc){
89      return !(((strategy == 2 || strategy == 3) && ((nproc & (nproc - 1)↩
    ) != 0)) || (nproc == 1)) ? 1 : 0;
90  }
91
92  int sequential_sum(int *array, int n){
93      int sum = 0;
94
95      for(int i = 0; i < n; i++){
96          sum += array[i];
97      }
98
99      return sum;
100 }
101
102 void operand_distribution(int menum, int *elements, int *elements_loc, ↩
    int nloc, int nproc, int rest){
103     int tag;
104     MPI_Status status;
105
106     if (menum == 0){
107         for (int i = 0; i < nloc; i++){
108             elements_loc[i] = elements[i];
109         }
110
111         int tmp = nloc;
112         int start = 0;
113         for (int i = 1; i < nproc; i++){
114             start += tmp;
115             tag = 22 + i;
116             if (i == rest)
117                 tmp -= 1;
118
119             MPI_Send(&elements[start], tmp, MPI_INT, i, tag, ↩
    MPI_COMM_WORLD);
120         }
121     } else {
122         tag = 22 + menum;
123         MPI_Recv(elements_loc, nloc, MPI_INT, 0, tag, MPI_COMM_WORLD, &↩
    status);
124     }
125 }
126
127 void print_result(int menum, int strategy, int sum, double timetot){
128     if(strategy == 1){
129         if(menum == 0)
130             printf("La somma totale e' %d e l'algoritmo, per calcolarla↩
    , ha impiegato %e.\n", sum, timetot);
```

```
131     } else {
132         printf("\n Sono il processo %d e la somma totale e' %d\n", ↩
    menum, sum);
133         if(menum == 0)
134             printf("Tempo totale impiegato per l'algoritmo: %e\n", ↩
    timetot);
135     }
136 }
137 
138 void compute_power_of_two(int logNproc, int *array){
139     for(int i = 0; i < logNproc + 1; i++){
140         array[i] = (int) pow(2, i);
141     }
142 }
```

6.5 *Utils.h*

```
1  #ifndef UTILS_H
2  #define UTILS_H
3  
4  /**
5   * @brief check if the inpurs are correct in order to sum
6   *
7   * @param argc number of parameters
8   * @param N number of elements to sum
9   * @param strategy the strategy to apply
10  */
11 int check_if_inputs_are_valid(int argc, int N, int strategy);
12 
13 /**
14  * @brief fill the array randomly if N is greater than 20, from argv ↩
     otherwise
15  *
16  * @param elements the array of integers
17  * @param N capacity of array
18  * @param argv the elements to insert into the array
19  */
20 void fill_array(int *elements, int N, char *argv[]);
21 
22 /**
23  * @brief check if the strategy 2 (or 3) is applicable: the number of ↩
     processor must be a power of 2
24  *
25  * @param strategy the strategy to apply (2 or 3)
26  * @param nproc the number of processor
27  * @return int 1 if it's applicable, 0 otherwise
28  */
29 int strategy_2_OR_3_are_applicable(int strategy, int nproc);
30 
31 /**
32  * @brief performs the sum of each array value and returns it
33  *
```

```
34  * @param array the array of integer
35  * @param n size of array
36  * @return int the sum
37  */
38 int sequential_sum(int *array, int n);
39
40 /**
41  * @brief the processor with id 0 send the elements to sum to the other processor
42  *
43  * @param menum id of the processor
44  * @param elements array of all integers
45  * @param elements_loc local array for each processor
46  * @param nloc number of elements to sum for each processor
47  * @param nproc number of processor
48  * @param rest the rest of division between the all numbers to sum and number of processor
49  */
50 void operand_distribution(int menum, int *elements, int *elements_loc, int nloc, int nproc, int rest);
51
52 /**
53  * @brief print results: print the partial sum for each processor, the time spent for each partial sum,
54  * the total sum and total time spent for the total sum
55  *
56  * @param menum id of processor
57  * @param strategy the strategy applied
58  * @param sum the result to print
59  * @param timetot time taken
60  */
61 void print_result(int menum, int strategy, int sum, double timetot);
62
63 /**
64  * @brief compute the powers of 2 for second and third strategy
65  *
66  * @param logNproc number of steps
67  * @param array array to fill
68  */
69 void compute_power_of_two(int logNproc, int *array);
70
71 #endif
```