UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

SCUOLA INTERDIPARTIMENTALE DELLE SCIENZE,

DELL'INGEGNERIA E DELLA SALUTE

INFORMATICA

Progetto di Calcolo Parallelo e Distribuito

Proponenti:

Mungari Alfredo 0124002134

Giordano Orsini Massimiliano 0124002214

Ferraro Dominick 0124002048

Data di Consegna:

14/06/2022

Anno Accademico:

2021 – 2022

Categoria:

Matrice x Vettore

Descrizione generale del progetto

La traccia richiede l'implementazione dell'algoritmo parallelo (p processori) per il calcolo del prodotto tra una matrice A di dimensione nxm e un vettore x di dimensione m, secondo la prima strategia di parallelizzazione ovvero partizionando la matrice per blocchi di righe monodimensionali. L'algoritmo è sviluppato in ambiente *MPI_Docker*.

Il calcolo del prodotto matrice-vettore è definito come segue:

$$Ax = y, A \in R^{nxm}, x \in R^{m}, y \in R^{n}$$

ove A è la matrice di dimensioni nxm, x è un vettore di dimensione m e y è un vettore di dimensione n. Si ricorda che tale prodotto è possibile solo se il numero di colonne della matrice A è esattamente pari al numero di righe del vettore x. Tale prodotto genera un vettore y di dimensione n.

L'algoritmo sequenziale prevede il calcolo del vettore y componente per componente:

$$y_i = \sum_{j=1}^{n} a_{i,j} * x_j, \ per \ \ i = 1, \ldots, n$$

Di seguito è riportato un esempio di algoritmo sequenziale per il prodotto matrice-vettore in pseudo-codice.

```
1.   procedure MAT_VECT ( A, x, y)
2.   begin
3.      for i := 0 to n - 1 do
4.      begin
5.         y[i]:=0;
6.         for j := 0 to n - 1 do
7.             y[i] := y[i] + A[i, j] x x[j];
8.      endfor;
9.   end MAT_VECT
```

Figura 1. Pseudo-codice dell'algoritmo sequenziale per prodotto matrice-vettore

Descrizione dell'approccio parallelo

E' possibile interpretare il prodotto matrice-vettore come una serie di prodotti scalari indipendenti tra le righe della matrice ed il vettore delle incognite.

Sia A una matrice di dimensioni nxm, x un vettore di dimensione m e y un vettore di dimensione n, la componente i-esima del vettore y è ottenuta come il prodotto scalare tra la riga i-esima della matrice A ed il vettore x.

La matrice A può essere distribuita ai processori del cluster con diverse strategie; nel nostro caso consideriamo la prima la prima strategia, la quale effettua la suddivisione più naturale poiché deriva direttamente dalla definizione del prodotto matrice per vettore, come illustrata in precedenza.

Supponendo che il processore *master* sia l'unico a contenere sia gli elementi della matrice A sia gli elementi del vettore x, è necessario che questi siano distribuiti tra i vari processori per il calcolo delle rispettive componenti di y.

La prima strategia prevede la decomposizione della matrice A in **blocchi di righe.**

Il processore *master* distribuisce a tutti i processori $\frac{n}{p}$ vettori di lunghezza m, se n è esattamente divisibile per p, altrimenti $\frac{n}{p} + 1$ vettori di lunghezza m.
Analogamente, il processore *master* distribuisce a tutti i processori l'intero vettore x di lunghezza m, siccome ogni processore necessita dell'intero vettore per il calcolo del prodotto scalare.

Al termine della fase di calcolo locale completamente parallela, è possibile ipotizzare che ciascun processore stampi le componenti calcolate localmente del vettore y, senza che queste risiedano quindi un'unica area di memoria, tipicamente del processore *master*, o nell'area memoria di ciascun processore, oppure che tali componenti vengono inviate ad un processore in particolare, ad esempio il processore *master*, il quale stamperà in sequenziale il vettore finale di lunghezza m.

Nel caso di non esatta divisibilità del numero di righe, è necessario distribuire le righe in eccesso tra i vari processori in qualche maniera. E' possibile ipotizzare in maniera arbitraria la distribuzione delle righe in eccesso tra eventuali processori che conseguentemente calcoleranno un prodotto scalare in più di lunghezza m.
Tutti gli altri processori attenderanno il completamento di questa fase.
Alternativamente, è possibile aggiungere un certo numero di righe nulle nel calcolo del prodotto matrice-vettore per ricondursi nel caso di esatta divisibilità.

Di seguito è riportata un'illustrazione grafica del prodotto matrice-vettore

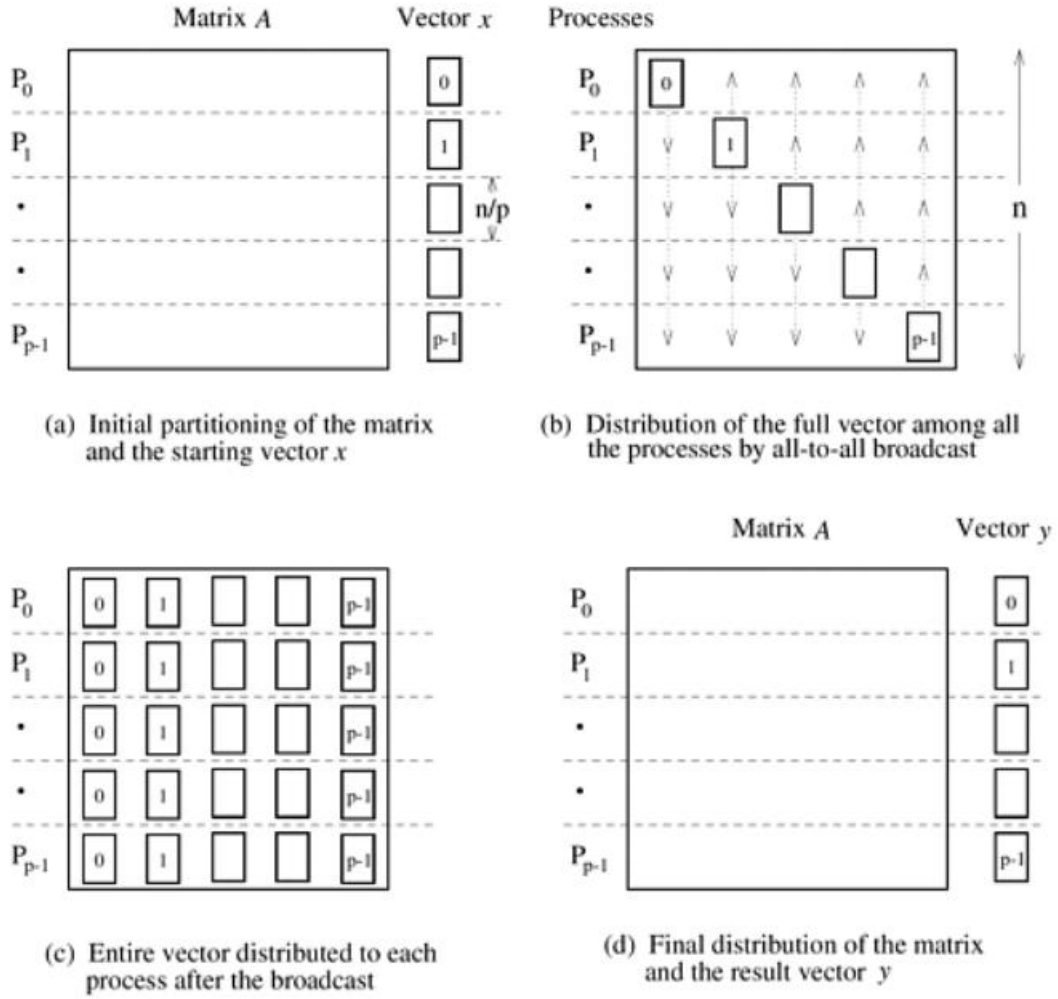

Figura 2. Distribuzione delle righe della matrice A

CALCOLO DI SPEED-UP, OVERHEAD ED EFFICIENZA

L'algoritmo sequenziale prevede n prodotti scalari di lunghezza m, cioè:

$$n[m\ molt. + (m-1)\ add. \xrightarrow[molt \sim add]{} n[2m-1]\ operazioni$$

La complessità di tempo sequenziale dell'algoritmo parallelo è pari a:

$$T_1(n \times m) = n[2m-1]\ t_{calc}$$

Si ricorda che tale complessità esprime il numero di operazioni eseguite dall'algoritmo parallelo eseguito da un'unica unità processante per il calcolo del prodotto matrice-vettore.

Nel caso di *MIMD-DM*, ogni riga della matrice A è assegnata a ciascun processore, quindi ogni processore avrà $\frac{n}{p}$ righe.

Il vettore x è presente in memoria per tutti i processori; pertanto, con p processori:

$$\dim[A_i] = \left(\frac{n}{p}\right) \times m$$
$$\dim[b] = m$$

L'unica fase della prima strategia di parallelizzazione prevede il calcolo in parallelo di $\frac{n}{p}$ prodotti scalari di lunghezza m, cioè:

$$T_p(n \times m) = \frac{n}{p}\ [2m-1]\ t_{calc}$$

SPEED-UP

Lo speed-up è il rapporto tra l'algoritmo parallelo eseguito con un processore e l'algoritmo parallelo eseguito con p processori; misura la riduzione del tempo di esecuzione dell'algoritmo sequenziale rispetto al tempo di esecuzione dell'algoritmo parallelo.

$$S_p(n \times m) = \frac{T_1(n \times m)}{T_P(n \times m)} = \frac{n[2m-1]}{\frac{n}{p}[2m-1]} = p$$

Nel caso di prodotto-matrice vettore secondo la prima strategia di parallelizzazione, lo speed-up è pari allo speed-up ideale.

OVERHEAD

L'overhead totale misura quanto lo speed-up differisce da quello ideale, ovvero la quantità di operazioni che non è possibile parallelizzare.

$$O_h = pT_p(n \times m) - T_1(n \times m) = p\left(\frac{n}{p}[2m-1]\right)t_{calc} - n[2m-1] = 0$$

L'algoritmo parallelo del calcolo matrice-vettore secondo il partizionamento per blocchi di righe della matrice non produce overhead.

EFFICIENZA

L'efficienza è il rapporto tra lo speed-up ed il numero di processori.

$$E_p(n \times m) = \frac{S_P(n \times m)}{p} = \frac{T_1(n \times m)}{T_P(n \times m)} = \frac{n[2m-1]}{\frac{n}{p}[2m-1]} = \frac{p}{p} = 1$$

Nel nostro caso, l'efficienza è pari a quella ideale.

Dalle valutazioni di speed-up, overhead totale ed efficienza, l'algoritmo parallelo per il calcolo matrice-vettore secondo la prima strategia di parallelizzazione è considerato un algoritmo completamente parallelizzabile.

ISOEFFICIENZA

L'isoefficienza è la legge secondo cui si sceglie la nuova dimensione del problema, affinchè l'efficienza resti costante; è dimostrabile che è misurata dal rapporto tra gli overhead. Si ricorda che nel caso prodotto matrice-vettore, per n_o ed n_1 s'intendono rispettivamente le dimensioni iniziali della matrice e le dimensioni al passo successivo.

$$T_p(n_1) = \frac{O_h(n_1, p_1)}{O_h(n_0, p_0)} T_p(n_0)$$

Siccome l'overhead totale O_h è 0 e l'efficienza è basata sull'overhead, si ottiene che

$$I = \frac{O_h(n_1, p_1)}{O_h(n_0, p_0)} = forma\ indeterminata$$

Per convenzione, l'isoefficienza è posta uguale ad ∞, ovvero è possibile considerare qualsiasi costante moltiplicativa per calcolare la dimensione n_1 e quindi controllare la scalabilità dell'algoritmo.

WARE-AMDHAL

Siccome è possibile distinguere la parte completamente parallela dalla parte completamente sequenziale, viene applicata la forma base della legge di Ware-Amdhal, per cui:

$$S_p(n \times m) = \frac{1}{\alpha + \frac{(1 - \alpha)}{p}}$$

Nella fase di calcolo parallelo ciascun processore effettua $\frac{n}{p}[2m - 1]$ prodotti scalari; vengono eseguite quindi $p\ \frac{n}{p}[2m - 1]$ delle $n[2m - 1]$ operazioni.
Si ottiene:

$$1 - \alpha = p\frac{n[2m - 1]}{pn[2m - 1]} = 1 \Rightarrow \frac{1 - \alpha}{p} = \frac{1}{p} \Rightarrow \alpha = 0$$

Lo speed-up secondo la legge di Ware-Amdhal è quindi pari a:

$$S_p(n \times m) = \frac{1}{\alpha + \frac{(1 - \alpha)}{p}} = \frac{1}{\frac{1}{p}} = p$$

CONSIDERAZIONI

Di seguito vengono riportati i conti nel caso in cui n non sia esattamente divisibile per p:

$$\dim[A_i] = \left(\frac{n}{p} + 1\right) \times m\ per\ 0 < i < mod(n, p)$$

$$\dim[A_i] = \left(\frac{n}{p}\right) \times m\ per\ mod(n, p) \leq i < n$$

$$\dim[b] = m$$

COMPLESSITA' DI TEMPO DELL'ALGORITMO PARALLELO

$$T_p(n \times m) = \left(\frac{n}{p} + 1\right)[2m - 1]\ t_{calc}$$

SPEED-UP

$$S_p(n \times m) = \frac{T_1(n \times m)}{T_P(n \times m)} = \frac{n[2m-1]}{\frac{\left(\frac{n}{p}+1\right)}{p}[2m-1]}$$

OVERHEAD

$$O_h = pT_p(n \times m) - T_1(n \times m) = p\left(\left(\frac{n}{p}+1\right)[2m-1]\right)t_{calc} - n[2m-1]$$

EFFICIENZA

$$E_p(n \times m) = \frac{S_P(n \times m)}{p} = \frac{T_1(n \times m)}{T_P(n \times m)} = \frac{n[2m-1]}{\left(\frac{n}{p}+1\right)[2m-1]}$$

WARE-AMDHAL

Siccome non è possibile distinguere la parte completamente parallela dalla parte completamente sequenziale, viene applicata la forma generalizzata della legge di Ware-Amdhal, per cui:

$$S_p(n \times m) = \frac{1}{\sum_{i=1}^{p} \frac{\alpha_k}{k}}$$

Ad ogni modo nella soluzione proposta, quando *n* non è esattamente divisibile per *p*, il numero delle righe diviene pari al prossimo numero divisibile per *p*. Ad esempio, se *p = 8, n = 12, m = 15,* la matrice assume dimensioni *16x15* piuttosto che *12x15*, aggiungendo righe nulle come elemento neutro del prodotto matrice-vettore e riconducendosi quindi sempre al caso di esatta divisibilità.

Descrizione dell'algoritmo parallelo

L'algoritmo proposto per il calcolo del prodotto matrice-vettore secondo il partizionamento per blocchi di righe della matrice prevede:

- la configurazione di una griglia di processori;
- l'acquisizione dei dati provenienti da un file di testo da parte del processore master;
- la comunicazione e la distribuzione dei dati acquisiti da file di testo da parte del processore master;
- la fase di calcolo parallelo dei prodotti scalari da parte di ciascun processore;
- la collezione e la stampa dei risultati da parte del processore master.

I processori vengono disposti secondo una griglia *px1* periodica, in maniera tale da far corrispondere a ciascun processore un certo numero di righe, coerentemente con la strategia di parallelizzazione.

Il processore *master* legge da file di testo, compilato con valori generati in maniera pseudo-casuale, il numero di righe della matrice *n*, il numero di colonne della matrice *m*, gli elementi della matrice *A* e gli elementi del vettore *x*.

Nel caso in cui il numero di righe *n* non sia esattamente divisibile per il numero di processori *p*, vengono aggiunte righe nulle per ricondursi nel caso di esatta divisibilità; pertanto, ciascun processore effettuerà lo stesso numero di prodotti scalari, pari a *n/p*, di lunghezza *m*.
Il processore *master* comunica le dimensioni della matrice *A* con la funzione *MPI_Bcast()* e le righe della matrice vengono distribuite con la funzione *MPI_Scatter()* tra i vari processori.
Allo stesso modo, il vettore *x* viene inviato per intero agli altri processori mediante la funzione *MPI_Bcast()*.

Ciascun processore calcola le componenti del vettore risultante *y* in maniera completamente parallela; successivamente queste vengono collezionate con la funzione *MPI_Gather()* dal processore *master*, che si occupa della stampa della matrice *A*, il vettore *x* ed il vettore *y*.

La matrice *A* e il vettore *x* sono implementati mediante array *stacked* 1-D di tipo intero.
Il vettore *y* è implementato mediante un array di tipo *long long int,* pari a 64 byte, per evitare overflow nel calcolo delle componenti del vettore risultante.

N.B.: la parte relativa alla collezione dei risultati è una comodità introdotta per la stampa dei risultati a schermo; è quindi esclusa dal calcolo della complessità di tempo dell'algoritmo parallelo, valutazione dei parametri di un algoritmo in ambiente parallelo e dalla misura dei tempi.

Input e Output

Per compilare il file sorgente, è necessario utilizzare il compilatore *mpicc,* definito in ambiente MPI.

Compilazione

```
mpicc MatrixVectorMultiplication_I_Strategy.c –o <executableFile>
```

Esecuzione

Lo script bash *employ.sh* prende in input il machinefile contenente il numero di unità processanti per ogni macchina, il numero di processori con cui si vuole eseguire il programma e il file eseguibile.

```
./employ.sh <machinefile> <nproc> <executableFile>
```

Il machinefile è compilato in maniera tale che ciascun nodo contenga 64 unità processanti, in maniera tale da

supportare l'elaborazione fino a 256 unità processanti.

Le dimensioni e gli elementi della matrice A e del vettore x sono contenute nel file di testo *inputFile.txt*, il quale viene compilato con valori generati in maniera pseudo-casuale. Tale file dev'essere strutturato in maniera tale che contenga sequenzialmente, una riga dietro l'altra, il numero di righe della matrice *n*, il numero di colonne della matrice *m*, gli elementi della matrice *A* e gli elementi del vettore *x*.

L'output a schermo mostra gli elementi della matrice *A*, il vettore *x* ed il vettore *y*.
In caso di situazione di errore previsto, viene mostrato un messaggio diagnostico dell'errore.

Routine utilizzate

Routine MPI

MPI_Scatter

Invia dati da un processore agli altri processori in un communicator

```
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype, int root,
               MPI_Comm comm)
```

Input Parameters

sendbuf
: indirizzo del buffer di invio

sendcount
: numero di elementi inviati ad ogni processore

sendtype
: tipo di dato dei dati da inviare

recvcount
: numero di elementi nel buffer di ricezione

recvtype
: tipo di dato dei dati ricevere

root
: rank del processore master

comm
: communicator

Output Parameters

recvbuf
: indirizzo del buffer di ricezione

MPI_Gather

Colleziona insieme i valori di un gruppo di processori

```
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                void *recvbuf, int recvcount, MPI_Datatype recvtype, int root,
 MPI_Comm comm)
```

Input Parameters

sendbuf
: indirizzo di partenza del buffer di invio

sendcount
: numero di elementi nel buffer di invio

sendtype
: tipo di dato degli elementi del buffer di invio

recvcount
: numero di elementi nel buffer di ricezione

recvtype
tipo di dato dei dati da ricevere
root
rank del processore master
comm
communicator

Output Parameters
recvbuf
indirizzo del buffer di ricezione

MPI_Bcast
Condivide un messaggio dal processore con rank "root" a tutti gli altri processori del communicator

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

Input/Output Parameters
buffer
indirizzo di partenza del buffer

Input Parameters
count
numero di elementi del buffer
datatype
tipo di dato nel buffer
root
rank del broadcast root
comm
communicator

MPI_Comm_rank
Determina il rank del processore chiamante nel communicator

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

Input Parameters
comm
communicator (handle)

Output Parameters
rank
rank del processore chiamante nel gruppo comm

MPI_Comm_size
Determina la dimensione del gruppo associato con un communicator

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

Input Parameters
comm
communicator

Output Parameters
size
numero di processori nel gruppo comm

MPI_Cart_create

Realizza un nuovo communicator organizzato in un particolare topologia

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[],
                    const int periods[], int reorder, MPI_Comm * comm_cart)
```

Input Parameters

comm_old
: input communicator

ndims
: numero di dimensioni della griglia cartesiana

dims
: array di tipo intero di dimensione ndims che specifica il numero di processori in ogni dimensione

periods
: array booleano di dimensione ndims che specifica se la griglia è periodica in ogni dimensione

reorder
: i ranking potrebbe essere riordinati o meno

MPI_Barrier

Blocca fino a quanto tutti i processori nel communicator hanno raggiunto questa routine

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

Input Parameters

comm
: communicator

MPI_Wtime

Ritorna il tempo elaborato sul processore chiamante.

```
double MPI_Wtime( void )
```

Return value

Tempo in secondi rispetto ad un istante arbitrario nel passato.

MPI_Init

Inizializza l'ambiente di esecuzione MPI

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

Input Parameters

argc
: Puntatore al numero di argomenti

argv
: Puntatore al vettore degli argomenti

MPI_Finalize

Termina l'ambiente di esecuzione MPI

```
int MPI_Finalize( void )
```

Routine implementate

readMatrix

Legge la matrice di input.

```
int *readMatrix(FILE *text, int *rows, int *cols, int nproc, int *mod)
```

Input Parameters

text
Puntatore al file di testo

nproc
numero di processori

Output Parameters

rows
righe della matrice

cols
colonne della matrice

mod
resto della divisione

a
puntatore alla matrice A

readVector

Legge la matrice di input.

```
int *readVector(FILE *text, int cols)
```

Input Parameters

text
Puntatore al file di testo

cols
colonne della matrice

Output Parameters

x
puntatore al vettore x

printInput

Stampa a schermo i dati di input ovvero la matrice A e il vettore x

```
void printInput(int *a, int *x, int rows, int cols)
```

Input Parameters

a
puntatore alla matrice A

x
puntatore al vettore x

rows
numero delle righe della matrice

cols

numero delle colonne della matrice e del vettore

printOutput

Stampa a schermo i dati di output ovvero il vettore y

```
void printOutput(long long int* y, int size)
```

Input/Output Parameters

y

puntatore al vettore y

Input Parameters

size

numero delle colonne del vettore

MatrixVectorMultiplication

Realizza il prodotto matrice-vettore

```
long long int* MatrixVectorMultiplication(int nloc, int cols, int *aloc, int *x)
```

Input Parameters

nloc

Numero di righe del vettore aloc

cols

Numero di colonne del sotto-vettore aloc e vettore x

aloc

Puntatore al sotto-vettore aloc

x

Puntatore al vettore x

Output Parameters

y

Puntatore al vettore y

Analisi delle performance

Analizziamo il nostro algoritmo valutando in dettaglio alcune caratteristiche atte a specificare le prestazioni di un software parallelo.

Queste consentiranno all'utente di capire in quale situazione è più opportuno utilizzare l'algoritmo e quando invece il suo utilizzo non reca alcun palese vantaggio.

VALUTAZIONE DEI TEMPI

Siccome lavoriamo in ambiente Docker, definibile come un ambiente "simulato", le valutazioni dei tempi sono riportate a scopo illustrativo, precisando che tali misure andrebbero effettuate con cluster che mettono effettivamente a disposizione un elevato numero di unità processanti. Tali misure sono ovviamente influenzate dal numero delle unità processanti messe a disposizione dalla macchina che ha eseguito i test.

Fissata la dimensione N del problema, il tempo effettivo $\tau_p(N)$ è dato dalla complessità di tempo $T_p(N)$, moltiplicato per un fattore k, che dipende dallo stato attuale della macchina in esecuzione.

$$\tau_p(N) = k \cdot T_p(N)$$

La complessità di tempo $T_p(N)$ ingloba al suo interno il fattore $\mu = t_{calc}$, ovvero il tempo di esecuzione di un'operazione floating point, dipendente dalla macchina utilizzata.

Di seguito viene riportata la tabella che raffigura i tempi d'esecuzione dell'algoritmo parallelo, considerando un numero di processori che va da 1 a 256 e matrici quadrate 1x1 fino a 256x256 (65.536 elementi). I tempi sono stati misurati attraverso l'utilizzo della funzione *MPI_Wtime()*; sono riportati in secondi.

τ_p	Numero di unità processanti p									
Dimensione del problema N		1	2	4	8	16	32	64	128	256
	1	0.000001	0.000005	0.000019	0.0000016	0.089686	0.226548	0.490117	1.409912	3.139867
	4	0.000001	0.000011	0.000015	0.002613	0.089779	0.129935	0.590096	1.200898	2.759386
	16	0.000002	0.000012	0.000015	0.000067	0.105505	0.184393	0.498763	1.289974	2.849856
	64	0.000002	0.000012	0.000019	0.011402	0.079882	0.179952	0.519872	1.139997	2.529788
	256	0.000005	0.000006	0.000011	0.011402	0.079927	0.209921	0.410003	1.199876	2.620002
	1024	0.000009	0.000009	0.000022	0.000025	0.079999	0.199976	0.509997	1.169807	3.216928
	4096	0.000074	0.000042	0.000062	0.000288	0.057384	0.199924	0.559978	1.289416	3.229667
	16384	0.000185	0.000316	0.0000109	0.000187	0.069513	0.159987	0.530127	1.319995	3.317676
	65536	0.000402	0.000375	0.000267	0.000631	0.074142	0.159915	0.500392	1.039916	3.222357

Tabella 1. Tabella dei tempi effettivi di esecuzione dell'algoritmo parallelo

Di seguito viene riportato il grafico dei tempi effettivi di esecuzione dell'algoritmo proposto.

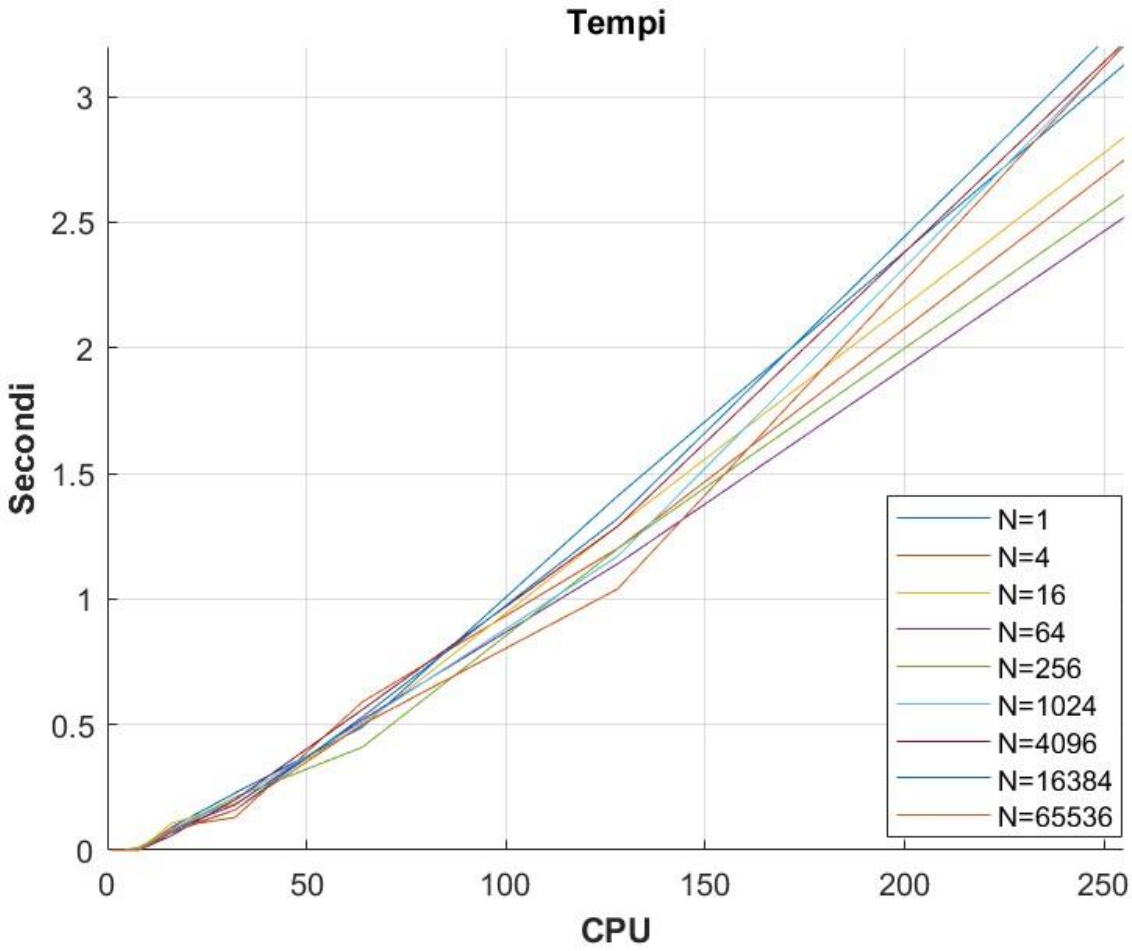

Figura 3. Grafico dei tempi misurati per l'algoritmo proposto.

Dal grafico si evince come all'aumentare del numero delle unità processanti e al crescere della dimensione del problema, aumentano i tempi effettivi di esecuzione, come prevedibile.

SPEED-UP
Data la dimensione del problema N, lo speed-up misurato è dato dal rapporto tra il tempo effettivo di esecuzione dell'algoritmo parallelo, eseguito da una sola unità processante e il tempo effettivo di esecuzione con p processori.

$$S_{t_p}(N) = \frac{\tau_1(N)}{\tau_p(N)}$$

Di seguito viene riportato il grafico relativo allo speed-up.

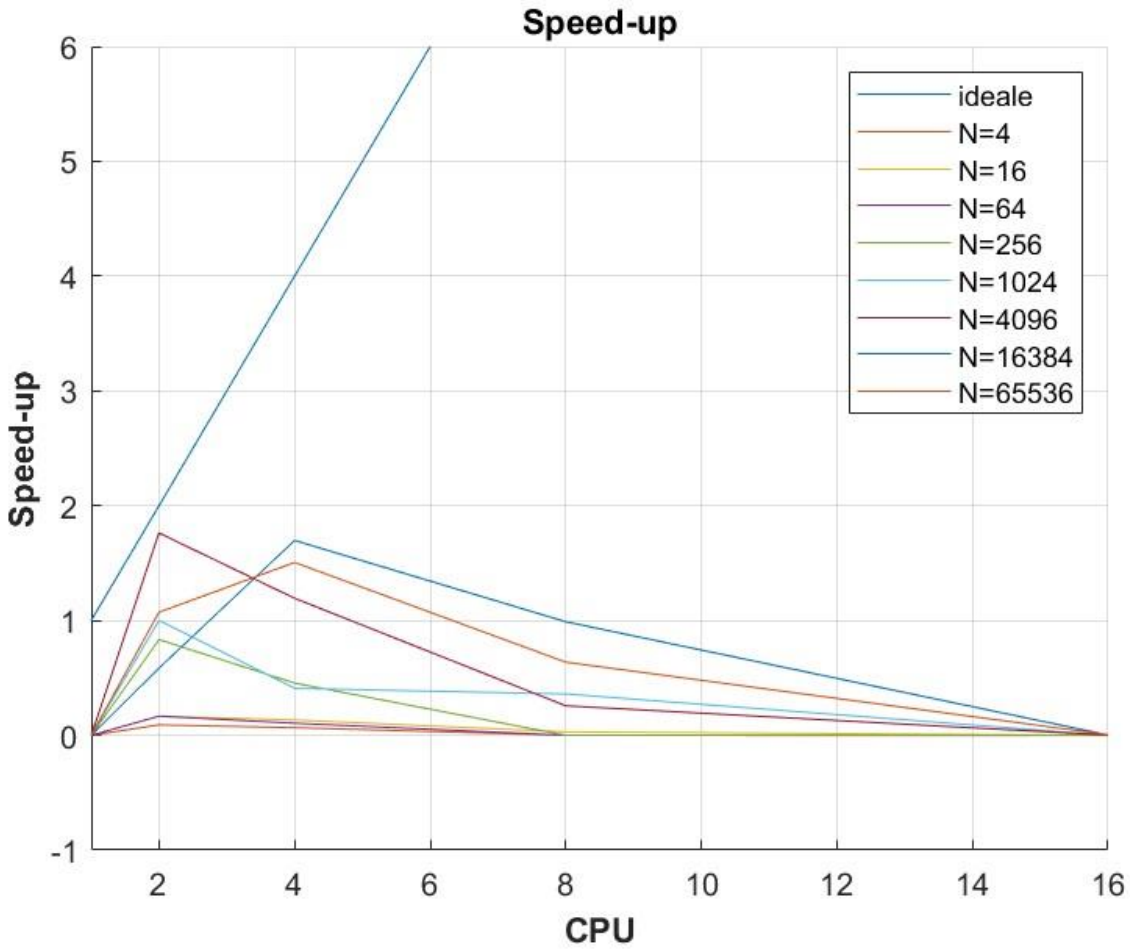

Figura 4. Speed-up misurato per l'algoritmo proposto

Come mostra il grafico, all'aumentare del numero delle unità processanti, lo speed-up tende a 0.
Per questioni di leggibilità, il grafico è stato limitato lungo l'asse x fino a 16 unità processanti.
Sebbene dal punto di vista teorico, il prodotto matrice-vettore secondo il partizionamento in blocchi di righe della matrice A tra i processori preveda uno speed-up ideale secondo la complessità di tempo $T(N)$, dall'ambiente simulato lo speed-up misurato sui tempi effettivi di esecuzione $\tau_p(N)$ non raggiunge lo speed-up ideale, in quanto incidono negativamente il fattore k e il fattore μ.

EFFICIENZA
Data la dimensione del problema N, l'efficienza misurata è dato dal rapporto tra lo speed-up misurato considerando il tempo effettivo di esecuzione $S_{\tau_p}(N)$ e il numero di processori p.

$$E_{t_p}(N) = \frac{S_{\tau_p}(N)}{p}$$

Analogamente per l'efficienza misurata secondo il tempo effettivo di esecuzione $\tau_p(N)$, vale quanto detto per lo speed-up $S_{t_p}(N)$.
Di seguito viene riportato il grafico dell'efficienza.

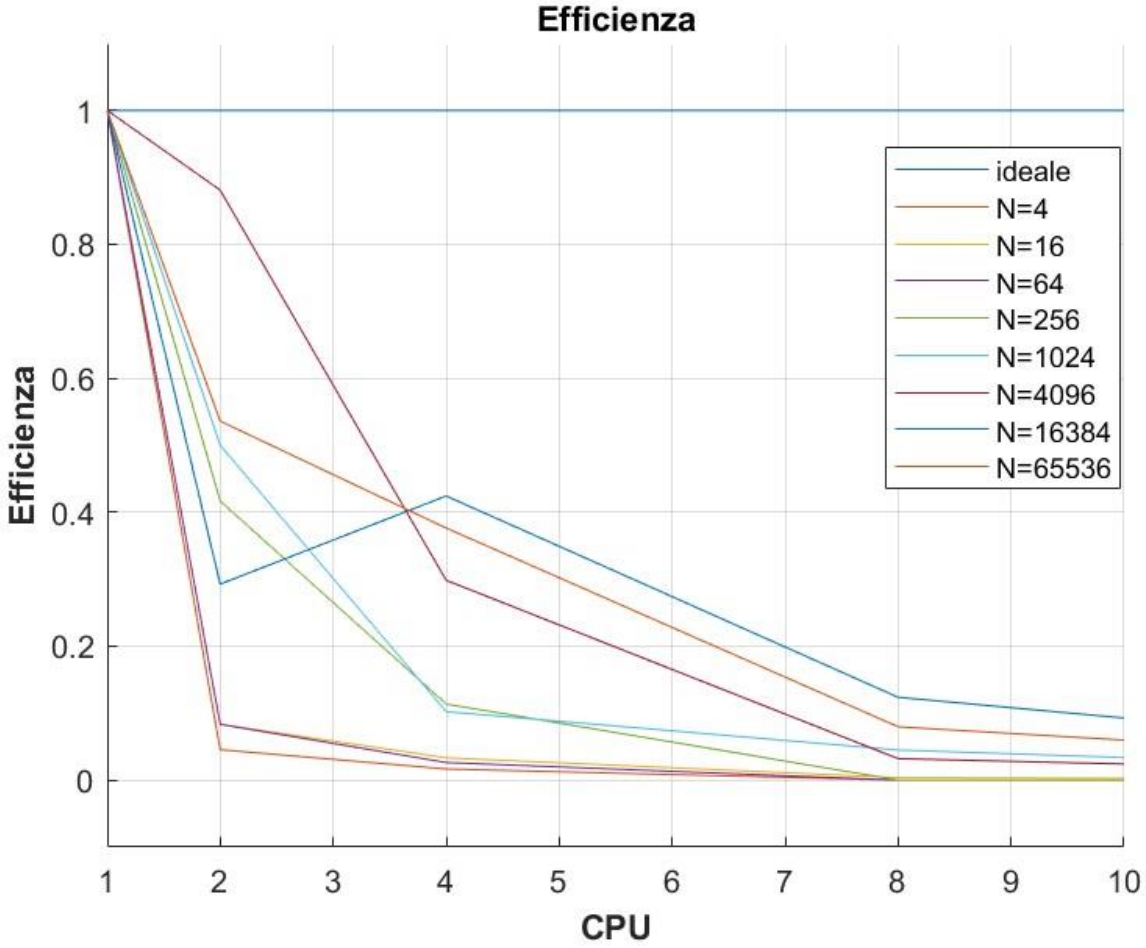

Figura 5. Efficienza misurata per l'algoritmo proposto

Come mostra la figura, all'aumentare del numero di processori p e al crescere della dimensione del problema, l'efficienza tende a 0, provocando il degrado delle prestazioni.
Per questioni di leggibilità, il grafico è stato limitato lungo l'asse x fino a 16 unità processanti.

Esempi d'uso

Esempio 1

Dopo aver compilato come mostrato **Input e Output**, viene lanciato lo script bash employ.sh, passando il machinefile, il numero di processori pari a 4, il nome del file eseguibile mat, numero di righe pari a 10 e numero di colonne pari a 9.

```
cpd2021@8c584b9e4120:/Docker_MPI/Matrix-Vector-Multiplication-OPEN_MPI/Source$ ./employ.sh machinefile 4 mat 10 9
Starting MPI employing


A =
 164     152     163     86      84      191     242     253     250
 122     108     83      21      20      234     227     46      82
 143     32      87      9       88      40      168     6       213
 209     83      131     120     118     28      154     75      238
 89      62      107     83      55      214     37      75      105
 143     174     150     96      61      54      182     197     141
 222     109     18      51      62      227     181     181     90
 208     207     36      62      40      224     168     250     151
 253     31      225     103     45      15      124     141     203
 49      67      144     189     160     124     207     210     185



x=
 178     136     237     139     215     188     47      22      100


y=
 196357  223442  217343  159539  142566  166346  143190  170233  184543  168794
Effective execution time: 0.000013
Effective execution time: 0.000013
Effective execution time: 0.000013
Effective execution time: 0.000013
```

Per scopi illustrativi, viene mostrato a schermo il vettore risultante, osservando il comportamento del programma quando il numero delle righe non è esattamente divisibile per il numero di processori.

Esempio 2
Dopo aver compilato come mostrato **Input e Output**, viene lanciato lo script bash employ.sh, passando il machinefile, il numero di processori pari a 8, il nome del file eseguibile mat ed una matrice quadrata 8x8.

```
cpd2021@8c584b9e4120:/Docker_MPI/Matrix-Vector-Multiplication-OPEN_MPI/Source$ ./employ.sh machinefile 8 mat 8 8
Starting MPI employing


A =
 164     152     163     86      84      191     242     253
 250     122     108     83      21      20      234     227
 46      82      143     32      87      9       88      40
 168     6       213     209     83      131     120     118
 28      154     75      238     89      62      107     83
 55      214     37      75      105     143     174     150
 96      61      54      182     197     141     222     109
 18      51      62      227     181     181     90      208



x=
 207     36      62      40      224     168     250     151


y=
 202573  166999  71660  144976  95145  138077  172471  143346
Effective execution time: 0.000029
Effective execution time: 0.000029
Effective execution time: 0.000029
Effective execution time: 0.000029
Effective execution time: 0.000029
Effective execution time: 0.000029
Effective execution time: 0.000029
Effective execution time: 0.000029
```

Viene quindi mostrato a schermo, per scopi illustrativi, il comportamento del programma quando il numero delle righe è esattamente divisibile per il numero di processori.
In entrambi i casi l'algoritmo risulta portare a termine l'esecuzione con successo.

Riferimenti bibliografici

Gupta, A., Karpis, G., Kumar, V., & Grama, A. (2003). *Introduction to Parallel Computing*. Pearson College Div.

Appendice

Il codice sorgente, la documentazione e le immagini sono riportate nella GitHub repository al link: https://github.com/dom0000D/Matrix-Vector-Multiplication-OPEN_MPI

Glossario

Il glossario ha lo scopo fondamentale di chiarire il gergo tecnico usato e di evidenziare eventuali sinonimie e omonimie. Trattandosi di un contesto **informatico**, la maggioranza dei termini riguardano tale ambito, le informazioni riportate valgono per lo stato italiano. È possibile che in altri Paesi, tali termini tradotti letteralmente possono essere utilizzati in contesti che differiscono da quelli di nostro interesse.

Termine	Descrizione	Sinonimi	Omonimi
Processore	Tipo di dispositivo hardware di un computer che si contraddistingue per essere dedicato all'esecuzione di istruzioni, a partire da un instruction set.	Unità di elaborazione	-
Cluster	Insieme di computer connessi tra loro tramite una rete telematica.	Gruppi	-
MIMD-DM	"Multiple Istruction Multiple Data – Distribuited Memory". Ambiente distribuito con cluster di processori che utilizzano una sola unità di controllo.		-
Docker	Software libero progettato per eseguire processi informatici in ambienti isolabili, minimali e facilmente distribuibili chiamati container Linux.		-
Machinefile			-
Mpicc	Comando usato per compilare e linkare programmi MPI scritti in linguaggio C.		-
		-	-