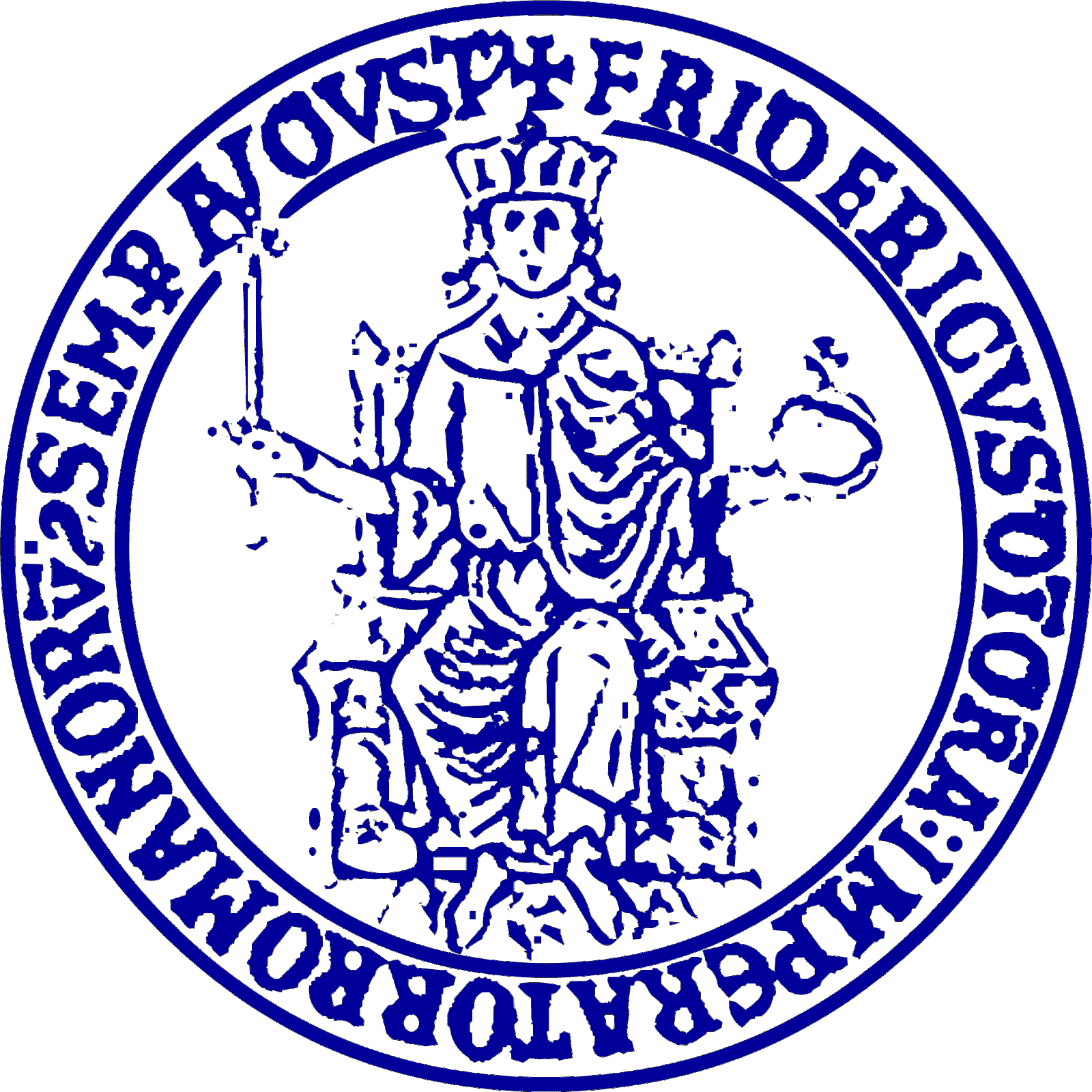
**UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II**

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL’INFORMAZIONE



CORSO DI LAUREA IN MAGISTRALE INFORMATICA

Documentazione esterna Parallel Distributed Computing per il calcolo della somma in parallelo

Studente: **Giuseppe Piccolo**

Matricola: **N97000415**

Anno accademico 2022/2023

Sommario

[1 Definizione e analisi del problema 5](#_Toc117323315)

[2 Descrizione algoritmo 6](#_Toc117323316)

[2.1 Gestione input 7](#_Toc117323317)

[2.3 Calcolo somma 8](#_Toc117323318)

[2.4 Strategia 1 10](#_Toc117323319)

[2.5 Strategia 2 11](#_Toc117323320)

[2.6 Strategia 3 12](#_Toc117323321)

[2.7 Stampa somma e tempo 13](#_Toc117323322)

[3 Input e Output 13](#_Toc117323323)

[4 Indicatori di errori 14](#_Toc117323324)

[5 Subroutine 14](#_Toc117323325)

[5.1 Subroutine librerie esterne 14](#_Toc117323326)

[5.2 Subroutine personali 17](#_Toc117323327)

[6 Analisi dei tempi 23](#_Toc117323328)

[6.1 Analisi preliminare con n = 20 - Strategia 1 23](#_Toc117323329)

[6.2 Analisi preliminare con n = 20 - Strategia 2 24](#_Toc117323330)

[6.2 Analisi preliminare con n = 20 - Strategia 3 24](#_Toc117323331)

[6.2 Strategia 1 – 200000000 numeri 25](#_Toc117323332)

[6.3 Strategia 2 – 200000000 numeri 26](#_Toc117323333)

[6.4 Strategia 3 – 200000000 numeri 27](#_Toc117323334)

[6.5 Strategia 1 – 300000000 numeri 28](#_Toc117323335)

[6.6 Stategia 2 – 300000000 numeri 29](#_Toc117323336)

[6.7 Stategia 3 – 300000000 numeri 30](#_Toc117323337)

[7 Esempi d’uso 30](#_Toc117323338)

# 1 Definizione e analisi del problema

Ci è stato chiesto di sviluppare un software che permettesse di eseguire la somma di più numeri, utilizzando le varie tecniche e strategie del calcolo parallelo. In questo caso specifico, per calcolare la somma di più numeri, è stato possibile implementare diversi tipi di strategie, nello stesso software. In particolare, è possibile decidere, prima di eseguire il programma, una delle tre strategie implementate, a patto che gli ulteriori parametri dati in input siano corretti. Nello specifico l'applicativo deve essere in grado di prendere in input il quantitativo di numeri da sommare, se questo valore è minore o uguale a 20, l'utente è obbligato a passare questi valori in input, altrimenti i numeri sono generati casualmente. Successivamente deve prendere in input un numero compreso tra 1 e 3, per la scelta della strategia da eseguire e infine, l'ultimo valore da prendere in input è un numero compreso tra -1 e il numero di processori utilizzati, se il valore è -1, tutti i processori stamperanno il risultato e il tempo trascorso altrimenti solo il processore con l'id numerico uguale al valore passato in input stamperà il risultato e il tempo impiegato. Visto che il programma deve gestire molte casistiche di dati in input, una delle complicazioni dello sviluppo dell'applicativo, è quello di gestire gli errori che possono partire da dati sbagliati passati in ingresso, per esempio quando si vuole eseguire la somma dei numeri con la strategia 2 utilizzando un numero di processori che non è una potenza di 2. Un'altra difficoltà affrontata è stata quella dell'implementazione degli algoritmi per eseguire la somma in parallelo, come per esempio la gestione della distribuzione dei numeri dal processore radice agli altri processori e dell'implementazione delle varie strategie che permettono di eseguire il calcolo delle somme parziali in parallelo fino ad arrivare al risultato finale. Tutte queste difficoltà elencate sono state affrontate e risolte in vari modi che saranno spiegate nelle sezioni successive in questa documentazione.

# 2 Descrizione algoritmo

Essenzialmente, l’algoritmo implementato può essere diviso in tre principali blocchi:

* Il primo blocco si occupa della gestione dei valori passati in input tramite l’array argv, dell’inizializzazioni delle variabili che servono all’algoritmo per funzionare e della gestione di eventuali errori, nel caso in cui gli argomenti passati in ingresso non siano corretti;
* Il secondo blocco si occupa dell’esecuzione della somma dei numeri. In particolare, il processore radice si occuperà di distribuire i numeri, equamente, a tutti gli altri processori, poi ognuno di essi eseguirà la somma parziale dei numeri ricevuti e successivamente si utilizzerà la strategia scelta dall’utente per permettere di sommare i numeri passati in input;
* Il terzo blocco si occupa della stampa della somma totale e del tempo di esecuzione che l’algoritmo ha impiegato per sommare i numeri e gestisce la modalità di stampa che l’utente ha deciso di utilizzare.

L’architettura può essere riassunta con questo schema:

Gestione input

Calcolo Somma

Stampa somma e tempo

## 2.1 Gestione input

Come già spiegato, gli argomenti passati in input vengono ottenuti tramite l’array *argv*. Ogni posizione del vettore corrisponde a un argomento specifico. In particolare, l’argomento *argv[1]* corrisponde al parametro che specifica la quantità dei numeri da passare in input, se questo parametro è minore o uguale a 20, allora da *argv[2]* ad *argv[numbers\_size]* si troveranno i numeri passati in ingresso, altrimenti i numeri verranno generati casualmente e quindi non ci sarà alcun bisogno di passare i numeri in ingresso. Nella posizione *argv[argc - 2]*, dove *argc* è la grandezza di *argv*, si troverà un valore compreso tra 1 e 3 che rappresenta la strategia che si vorrà utilizzare, infine il valore nella posizione *argv[argc - 1],* che sarà compreso tra -1 e *number\_of\_processes,* rappresenta il processore che dovrà stampare la somma e il tempo impiegato, se il valore è -1 tutti i processori dovranno stampare il risultato.

Ricapitolando:

*argv[1] =* “quantità numeri da passare in ingresso”

*se argv[1]* <= 20 allora nell’intervallo [*argv[2]*, *argv[numbers\_size]*] = “numeri passati in input”

se argv[1] > 20 allora l’intervallo [*argv[2]*, *argv[numbers\_size]*] non esiste e i numeri verranno generati casualmente

*argv[argc - 2]* = “strategia da utilizzare (numero compreso tra 1 e 3)”

*argv[argc - 1]* = “processore che deve stampare il risultato (numero compreso tra 0 e number\_of\_processes) oppure stampano tutti i processori (valore -1)”

Sequenzialmente il processore zero va a ottenere, controllare e salvare i valori dall’array nelle apposite variabili. Appena il processore zero trova un errore negli argomenti passati, come per esempio la mancanza di uno degli argomenti obbligatori da passare, interrompe l’esecuzione, chiama le varie funzioni per rilasciare le risorse occupate dell’applicativo, viene stampato un messaggio di errore nello stream degli errori e tutto il programma viene terminato bruscamente.

Di seguito è mostrato lo pseudo-codice di questo blocco che viene eseguito dal processore zero:

if(id\_processor == 0)

if(parameters\_are\_not\_enough(argv))

throw\_error();

size\_numbers = get\_size\_numbers\_from\_argv(argv);

if((size\_numbers <= 20 and argc == (size\_numbers + NECESSARY\_PARAMETERS)) or

(size\_numbers > 20 and argc == NECESSARY\_PARAMETERS)) or

(size\_numbers > 1)

numbers\_to\_sum = get\_numbers\_to\_sum(argv, size\_numbers);

else

throw\_error();

strategy = get\_strategy(argv, argc);

if(strategy < 1 or strategy > 3)

throw\_error();

id\_printer = get\_id\_printer(argv, argc);

if(id\_printer < -1 or id\_printer > processors)

throw\_error();

## 2.3 Calcolo somma

In questo blocco viene eseguito il codice per eseguire il calcolo della somma di N numeri. A seconda della strategia da utilizzare scelta dall’utente, il programma cambierà metodo di esecuzione del calcolo della somma dei numeri. La parte iniziale è pressoché uguale per tutte e tre le strategie. Essa consiste nel distribuire i numeri, in modo equo, a tutti i processori. Per fare ciò, il processore zero invia il quantitativo di numeri da sommare a ogni processore, ognuno di essi dividerà questo numero per il numero di processori, e se il quantitativo di numeri non è un multiplo del numero di processori, si incrementerà di uno la quantità di numeri da avere localmente ai primi processori per avere una distribuzione equa dei numeri. Una volta fatto ciò il processore zero si occuperà di inviare i numeri corretti che gli altri processori devono avere per eseguire la somma parziale. Gli altri processori, una volta ottenuti i numeri, li allocheranno e li salveranno localmente. Fatto ciò, localmente ogni processore eseguirà la somma parziale dei numeri ricevuti e successivamente, a seconda della strategia che si sta utilizzando, si passa all’esecuzione della somma totale dei numeri. La strategia 1 può essere eseguita con qualsiasi numero di processori, mentre le strategie 2 e 3, possono essere eseguite se e solo se il numero di processori è un multiplo di 2. Nel caso in cui il numero di processori è incompatibile con la strategia utilizzata, la strategia predefinita utilizzata è la 1.

Di seguito è mostrato lo pseudo-codice che si occupa di eseguire la somma totale dei numeri:

Broadcast(size\_numbers\_to\_sum, 0);

local\_size\_numbers\_to\_sum = size\_numbers\_to\_sum / processors;

rest = size\_numbers\_to\_sum % processors;

if(id\_processors < rest)

++local\_size\_numbers\_to\_sum;

if(id\_processors == 0)

local\_numbers\_to\_sum = numbers\_to\_sum;

send\_numbers(local\_numbers\_to\_sum, local\_size\_numbers\_to\_sum, processors, rest);

else

local\_numbers\_to\_sum = alloc(local\_size\_numbers\_to\_sum);

get\_numbers(local\_numbers\_to\_sum, local\_size\_numbers\_to\_sum, id\_processor);

Broadcast(strategy, 0);

execute\_strategy(stategy, local\_numbers\_to\_sum, local\_size\_numbers\_to\_sum);

## 2.4 Strategia 1

Nella strategia 1 ogni processore esegue il calcolo locale della somma parziale dei numeri che ha ricevuto precedentemente e ci sarà un processore prestabilito, che riceverà queste somme parziali e poi le sommerà tutte assieme, ottenendo il risultato della somma totale. In altre parole, ogni processore, una volta eseguito il calcolo della somma parziale, la invierà al processore prestabilito. Esso le riceverà e le sommerà con la somma parziale che ha calcolato localmente. Una volta sommate tutte le somme parziali il processore prestabilito avrà il risultato della somma totale. Questa strategia può essere eseguita con un qualsiasi numero di processori. Quindi non esistono vincoli di esecuzione per questa strategia. In generale, il processore prestabilito a ottenere tutte le somme parziali sarà quello che impiegherà più tempo per eseguire il calcolo, visto che sarà colui che si occupa di eseguire tutte le somme.

La strategia 1 con 4 processori può essere schematizzata in questo modo:

**P1**

**P2**

**P3**

**P0**

**SOMMA TOTALE**

**S0 + S1 + S2 + S4**

**S0 + S1 + S2**

**S0 + S1**

**Iterazione 1**

**Iterazione 2**

**Iterazione 3**

## 2.5 Strategia 2

Nella strategia 2, i processori lavoreranno a coppie. Essenzialmente, si va a formare un albero binario capovolto quando questa strategia viene eseguita, le somme parziali calcolate localmente da ogni processore rappresentano le foglie dell’albero. Finito il calcolo delle somme parziali, ci sarà un ciclo che avrà log2(number\_of\_processors) iterazioni. Come prima cosa si andrà a controllare se alla iterazione i-esima l’id del processore p è divisibile per 2i. Si esegue questo controllo per scartare tutti i processori che non devono inviare o ricevere nulla alla iterazione i-esima. Successivamente si controlla se l’id del processore è divisibile per 2i + 1, per distinguere i processori che devono ricevere da quelli che devono inviare. Se questa condizione è vera, il processore riceverà dal processore id\_current\_processor + 2i la somma parziale e la sommerà con la propria in locale. Si esegue questo procedimento fino a che i < log2(number\_of\_processors). Questa strategia può essere utilizzata se e solo se il numero di processori è una potenza di 2, visto che si va a formare un albero binario.

La strategia 2 può essere rappresentata con questo schema:

**S2 + S3**

**S0 + S1 + S2 + S3**

**S0 + S1**

**P0**

**P1**

**P2**

**P3**

**Iterazione 1**

**Iterazione 2**

**SOMMA TOTALE**

## 2.6 Strategia 3

Nella strategia 3, coppie di processori condividono la propria somma parziale, inizialmente i processori eseguono localmente il calcolo della somma, fatto ciò, con un ciclo for verranno eseguite log2(number\_of\_processors) iterazioni e viene controllato se il resto dell’id del processore sommato a 2i + 1 è minore di 2i dove i sta per la i-esima iterazione del ciclo, se la condizione è vera allora il processore dovrà ricevere la somma parziale dal processore con id processore corrente + 2i, inviare la propria somma parziale al processore con id processore corrente + 2i e calcolare la somma parziale locale con quella ricevuta. Altrimenti, il processore dovrà inviare la propria somma parziale al processore con id processore corrente – 2i, ricevere la somma parziale del processore con id corrente – 2i e sommarla con la propria. Quindi ogni processore sarà coinvolto nel calcolo della somma fino alla fine dell’esecuzione dell’algoritmo e a esecuzione finita, ogni processore avrà il risultato della somma totale. Quest’algoritmo potrà essere eseguito se e solo se il numero di processori è una potenza di 2

La strategia 2 può essere rappresentata con questo schema:

**P3**

**P2**

**P1**

**P0**

**Iterazione 1**

**S0 + S1**

**S2 + S3**

**S2 + S3**

**S0 + S1**

**Iterazione 2**

**S0 + S1 + S2 + S3**

**S0 + S1 + S2 + S3**

**S0 + S1 + S2 + S3**

**S0 + S1 + S2 + S3**

**SOMMA TOTALE**

**SOMMA TOTALE**

**SOMMA TOTALE**

**SOMMA TOTALE**

## 2.7 Stampa somma e tempo

In questo blocco finale, viene gestito la stampa dei risultati ottenuti. Nel caso in cui un processore specifico deve stampare il risultato non c’è bisogno di fare nulla, visto che il processore prestabilito ad avere la somma totale corrisponderà al processore che deve stampare l’output. Mentre se tutti i processori devono stampare, il processore zero si occuperà di inviare, tramite una broadcast, il risultato della somma e del tempo impiegato a tutti gli altri processori. Alla fine, verrà eseguita la stampa solo sul processore prestabilito oppure verrà fatto stampare il risultato a tutti i processori.

Di seguito è mostrato lo pseudo-codice che gestisce la stampa:

If(id\_printer\_processor == -1)

Broadcast(sum, 1)

If(id\_printer\_processor == -1)

Time\_elapsed(sum, 1)

If(id\_printer\_processor == -1 or id\_printer\_processor == id\_processor)

print\_result(sum, time\_elapsed);

# 3 Input e Output

L’applicativo, per essere eseguito, ha bisogno di alcuni parametri quando viene lanciato a riga di comando con mpiexec. Il primo parametro è il quantitativo di numero che l’utente vuole passare in input. Questo parametro deve essere maggiore di 1. Nel caso in cui è maggiore di 1 e minore uguale di 20, bisogna passare tanti numeri quanti espressi in questo primo parametro. Nel caso in cui è anche maggiore di 20, non bisogna inserire alcun numero, visto che verranno generati automaticamente. Come penultimo parametro, bisogna inserire la strategia che si vuole utilizzare per eseguire la somma dei numeri. Deve essere un valore compreso tra 1 e 3. Infine, come ultimo parametro, bisogna inserire il processore che dovrà stampare la somma finale e il tempo di esecuzione. Dovrà essere un valore compreso tra -1 e numero\_processori. Se il valore passato è -1, ogni processore stamperà la somma e il tempo impiegato. Se sarà compreso tra 0 e numero\_processori, il processore che avrà lo stesso id, stamperà la somma e il tempo impiegato. Se il valore è maggiore del numero dei processori, allora il programma stamperà un errore e questo porterà la terminazione del programma.

# 4 Indicatori di errori

Nel caso in cui si passi come primo argomento un numero <= di 1, il programma terminerà e stamperà nel file .err, questo messaggio di errore “Error number of arguments”. Se si passa un quantitativo diverso di numeri dichiarato dal primo argomento l’applicativo interromperà l’esecuzione e stamperà questo messaggio di errore: “Error size numbers input!” nel file degli errori. Se il parametro che indica la strategia da usare è minore di 1 oppure è maggiore di 3 l’esecuzione viene interrotta e viene stampato questo messaggio “Error strategy!” nello stream degli errori. Nel caso in cui l’ultimo parametro, che indica chi deve stampare il tutto, è < -1 oppure > numero\_processori, allora il programma smetterà di funzionare e stamperà negli errori questo messaggio: “Error id printer processor!”. Nel caso in cui si scelga la strategia 2 o 3, con un numero inappropriato di processori, l’applicativo stamperà che utilizzerà la strategia 1 e si proseguirà con il calcolo e la stampa della somma e del tempo di esecuzione.

# 5 Subroutine

## 5.1 Subroutine librerie esterne

**int MPI\_Init(int \*argc, char \*\*\*argv)**

**input:**

**argc**

Puntatore al numero di argomenti

**argv**

Puntatore al vettore parametri

**output:**

**IERROR**

solo per fortran

**int MPI\_Comm\_rank(MPI\_Comm comm, int \*rank)**

**input:**

**comm**

Comunicatore (gestore).

**output:**

**rank**

rango del gruppo di processi di comm

**IERROR**

solo per fortran

**int MPI\_Comm\_size(MPI\_Comm comm, int \*size)**

**input:**

**comm**

Comunicatore (gestore).

**output:**

**size**

numero di processori nel gruppo di comm

**IERROR**

solo per fortran

**int MPI\_Abort(MPI\_Comm comm, int errorcode)**

**input:**

**comm**

Comunicatore (gestore).

**errorcode**

Codice di errore da restituire all’ambiente di invocazione

**output:**

**IERROR**

solo per fortran

**int MPI\_Bcast** **(void \*buffer, int count, MPI\_Datatype datatype,**

**int root, MPI\_Comm comm)**

**input:**

**buffer**

Indirizzo iniziale del buffer

**count**

Numero di istanze nel buffer

**datatype**

Tipo di dati del buffer

**root**

Rango della radice per la trasmissione

**comm**

Comunicatore (gestore).

**errorcode**

Codice di errore da restituire all’ambiente di invocazione

**output:**

**request**

richiesta (gestore)

**IERROR**

solo per fortran

**int MPI\_Barrier(MPI\_Comm comm)**

**input:**

**comm**

Comunicatore (gestore).

**output:**

**request**

richiesta (gestore)

**IERROR**

solo per fortran

**double MPI\_Wtime()**

**output:**

**time**

tempo in secondi del tempo trascorso dall’inizio dell’esecuzione

**int MPI\_Send (const void \*buf, int count, MPI\_Datatype datatype, int dest,**

**int tag, MPI\_Comm comm)**

**input:**

**buffer**

Indirizzo iniziale del buffer

**count**

Numero di istanze nel buffer

**datatype**

Tipo di dati del buffer

**dest**

rango di destinazione

**tag**

etichetta del messaggio

**comm**

Comunicatore (gestore).

**output:**

**IERROR**

solo per fortran

**int MPI\_Recv (void \*buf, int count, MPI\_Datatype datatype,**

**int source, int tag, MPI\_Comm comm, MPI\_Status \*status)**

**input:**

**count**

Numero di istanze nel buffer

**datatype**

Tipo di dati del buffer

**source**

Rango del sorgente

**tag**

etichetta del messaggio

**comm**

Comunicatore (gestore).

**output:**

**buf**

Indirizzo iniziale del buffer ricevitore

**status**

Oggetto status

**IERROR**

solo per fortran

## 5.2 Subroutine personali

**int get\_size\_numbers\_to\_sum(char \*\*argv)**

**input:**

**argv**

vettore argomenti

**returned value:**

quantità numeri passati in input

Questa funzione ritorna la quantità di numeri passati in input dall’utente estraendola da argv

**float \*get\_random\_numbers(int size)**

**input:**

**size**

quantità numeri da generare

**returned value:**

numeri generati casualmente

Questa funzione costruirà un vettore con numeri generati casualmente

**float \*get\_numbers\_from\_argv(int size, char \*argv[])**

**input:**

**size**

quantità numeri da estrarre da argv

**argv**

vettore argomenti

**returned value:**

numeri passati in input

Questa funzione estrae i numeri passati in input dall’utente tramite argv

**float \*get\_numbers\_to\_sum(int size, char \*argv[])**

**input:**

**size**

quantità numeri da estrarre da argv da generare casualmente

**argv**

vettore argomenti

**returned value:**

vettore di numeri da sommare

Questa funzione, in base a size deciderà di estrarre i numeri da argv o di generarli casualmente

**float \*get\_strategy(char \*argv[], int argc)**

input:

**argv**

vettore parametri

**argc**

grandezza vettore parametri

**returned value:**

strategia passata in input

Questa funzione estrae la strategia da utilizzare decisa dall’utente

**float \*get\_id\_printer\_processor(char \*argv[], int argc)**

input:

**argv**

vettore parametri

**argc**

grandezza vettore parametri

**returned value:**

id processore printer passato in input

Questa funzione estrae l’id del processore da utilizzare per stampare il risultato deciso dall’utente

**void send\_numbers(float \*local\_numbers\_to\_sum, int local\_size\_numbers\_to\_sum, int processors, int rest)**

**input:**

**local\_numbers\_to\_sum**

vettore con i numeri

**local\_numbers\_to\_sum**

grandezza local\_numbers\_to\_sum

**processors**

numero processor

**rest**

resto dell’operazione grandezza\_numeri % numero\_processori

La funzione è utilizzata dal processore 0 per inviare e distribuire i numeri agli altri processori

**void get\_numbers(float \*local\_numbers\_to\_sum, int local\_size\_numbers\_to\_sum, int id\_process)**

**input:**

**local\_numbers\_to\_sum**

vettore con i numeri

**local\_numbers\_to\_sum**

grandezza local\_numbers\_to\_sum

**id\_processor**

id del processore che deve ricevere i numeri localmente

La funzione è utilizzata da un determinato processore per ricevere i numeri distribuiti

**void get\_sum(float \*local\_numbers\_to\_sum, int local\_size\_numbers\_to\_sum)**

**input:**

**local\_numbers\_to\_sum**

vettore con i numeri

**local\_numbers\_to\_sum**

grandezza local\_numbers\_to\_sum

**returned value:**

somma parziale di local\_numbers\_to\_sum

La funzione calcola la somma parziale di local\_numbers\_to\_sum e restituisce il risultato

**int is\_a\_power\_of\_2(int number)**

**input:**

**number:**

numero da controllare

**returned value:**

1 se number è una potenza di 2, 0 altrimenti

La funzione controlla se number è una potenza di 2

**void strategy1(float \*sum, int id\_processor, int processors)**

**input:**

**sum:**

somma parziale

**id\_processor:**

id processore corrente

**processors**

numero processori

La funzione esegue la strategia 1

**void strategy2(float \*sum, int id\_processor, int processors)**

**input:**

**sum:**

somma parziale

**id\_processor:**

id processore corrente

**processors:**

numero processori

La funzione esegue la strategia 2

**void strategy3(float \*sum, int id\_processor, int processors)**

**input:**

**sum:**

somma parziale

**id\_processor:**

id processore corrente

**processors:**

numero processori

La funzione esegue la strategia 3

**void sum\_numbers(int strategy, float \*sum, int size\_numbers\_to\_sum, int id\_processor, int processors)**

**input:**

**strategy:**

strategia da usare

**sum:**

somma parziale

**size\_numbers\_to\_sum:**

quantità numeri da sommare

**id\_processor:**

id processore corrente

**processors:**

numero processori

La funzione in base agli argomenti passati, eseguirà la giusta strategia e controllerà se il numero di processori è compatibile con la strategia

**void free\_memory(int id\_processor, float \*numbers\_to\_sum, float \*local\_numbers\_to\_sum)**

**input:**

**id\_processor:**

id processore corrente

**numbers\_to\_sum:**

array con tutti i numeri da sommare

**local\_numbers\_to\_sum**

array locale con i numeri da sommare

La funzione esegue la liberazione delle risorse occupate

**void print\_sum(int id\_processor, int id\_printer\_processor, float sum, double total\_time\_elapsed)**

**input:**

**id\_processor:**

id processore corrente

**id\_printer\_processor:**

id del processore che deve stampare il risultato

**sum**

somma da stampare

**total\_time\_elapsed:**

tempo da stampare

La funzione esegue la stampa della somma e del tempo impiegato

**void send\_sum\_if\_broadcast\_true(int id\_printer\_processor, int id\_ processor, float \*sum)**

**input:**

**id\_printer\_processor:**

id del processore che deve stampare il risultato

**id\_processor:**

id processore corrente

**sum**

somma da stampare

La funzione invierà a tutti i processori la somma da stampare se l’utente ha inserito come modalità di stampa -1

**void send\_total\_time\_elapsed(int id\_printer\_processor, int id\_ processor, double \*total\_time\_elapsed)**

**input:**

**id\_printer\_processor:**

id del processore che deve stampare il risultato

**id\_processor:**

id processore corrente

**total\_time\_elapsed**

tempo impiegato da stampare

La funzione invierà a tutti i processori il tempo impiegato da stampare se l’utente ha inserito come modalità di stampa -1

**void handle\_error(char \*error\_message)**

**input:**

**error\_message:**

errore da stampare

La funzione gestisce la terminazione nel caso in cui c’è un errore

**int rotate\_tree(int id\_processor, int id\_printer, int processors)**

**input:**

**id\_processor:**

id del processore che ha chiamato la funzione

id\_printer:

id del processore che deve stampare il risultato finale

processors:

numero dei processori attivi

returned value:

nuovo id del processore con id\_processor

ritorna l’id del processore corrente ruotato, in modo tal da far sì che il processore con id\_printer stampi il risultato finale

# 6 Analisi dei tempi

## 6.1 Analisi preliminare con n = 20 - Strategia 1

In questa sezione viene studiato quanto valgono i valori T(p), S(p) ed E(p) nel caso in cui si utilizzi la strategia 1 con 20 numeri. Sappiamo che T(p) può essere diviso in 2 parti, ovvero Ts e Tc. Il primo rappresenta il numero di istruzioni che vengono eseguite sequenzialmente. Il secondo rappresenta il numero di operazioni che vengono eseguite in parallelo da p processori. Nel caso della strategia 1 la quantità di operazioni svolte sequenzialmente sono pari a p – 1. Perché ci deve essere almeno un processore che riceve e somma le somme parziali calcolate dagli altri processori. Per esempio, nel caso in cui p = 4 e il processore 0 riceve le somme degli altri processori, quest’ultimo deve sommare le somme parziali dei processori 1, 2 e 3. Quindi in generale Ts(p) = p – 1. Mentre Tp dipende da p e da n. Non è vero che quando si aumenta p il numero di operazioni si dimezzano, soprattutto se si utilizzano molti processori per risolvere un problema poco complesso. Il valore S(p), che sta per speed up, indica il rapporto tra T(1) e T(p) che misura la riduzione del tempo di esecuzione rispetto a T(1). Ovviamente l’ideale sarebbe che T(1)/T(p) = p, ma ciò non è quasi mai vero, perché lo speed up aumenta ma non in egual modo a p. Mentre il valore E(p), che sta per efficienza, indica quanto l’algoritmo implementato riesce a sfruttare il parallelismo ed è uguale a S(p) / p. Il tempo viene detto ideale quando E(p) < 1. Questi sono i valori calcolati e messi in tabella:

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) = Ts + Tc | S(p) | E(p) |
| 1 Processore | 19 | 1 | 1 |
| 2 Processori | 1 + 9 | 1,9 | 0,95 |
| 3 Processori | 2 + 6 | 2,375 | 0,792 |
| 4 Processori | 3 + 4 | 2,714 | 0,6785 |
| 5 Processori | 4 + 3 | 2,714 | 0,5428 |
| 6 Processori | 5 + 3 | 2,375 | 0,396 |
| 7 Processori | 6 + 2 | 2,375 | 0,339 |
| 8 Processori | 7 + 2 | 2,375 | 0.296 |

## 6.2 Analisi preliminare con n = 20 - Strategia 2

Nel caso in cui bisogna sommare 20 numeri con la strategia 2, avremo che Ts sarà sempre uguale a 1, visto che l’unica operazione che non viene eseguita in parallelo è quando viene effettuata l’ultima somma parziale dal processore scelto che deve avere il risultato finale. Il resto delle operazioni sono tutte eseguite in parallelo, quindi Tc dipenderà dal problema p ed n. In generale T(p) = n/p – 1 + log2p. Questi sono i valori calcolati e messi in tabella:

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) = Ts + Tc | S(p) | E(p) |
| 1 Processore | 1 + 18 | 1 | 1 |
| 2 Processori | 1 + 9 | 1,9 | 0,95 |
| 4 Processori | 1 + 5 | 3,166 | 0,792 |
| 8 Processori | 1 + 4 | 3,8 | 0,475 |

## 6.2 Analisi preliminare con n = 20 - Strategia 3

Invece, nel caso della strategia 3, non esistono istruzioni che vengono eseguite sequenzialmente. Quindi, Ts sarà sempre uguale a 0 e T(p) sarà costituito solo da Tc. Essenzialmente, anche qui vale la formula utilizzata per la strategia 2 per il calcolo di T(p), ovvero, T(p) = n/p – 1 + log2p. L’unica differenza è che l’istruzione, che viene eseguita sequenzialmente, non c’è in questo caso. Essa viene eseguita in parallelo da tutti i processori, siccome nell’ultimo passo tutti i processori eseguono il calcolo della somma totale. Questi sono i valori calcolati e messi in tabella:

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) = Ts + Tc | S(p) | E(p) |
| 1 Processore | 19 | 1 | 1 |
| 2 Processori | 10 | 1,9 | 0,95 |
| 4 Processori | 6 | 3,166 | 0,792 |
| 8 Processori | 5 | 3,8 | 0,475 |

## 6.2 Strategia 1 – 200000000 numeri

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| 1 Processore | 0,827163 | 1 | 1 |
| 2 Processori | 0,412411 | 2,007 | 1,004 |
| 3 Processori | 0,276533 | 2,986 | 0,995 |
| 4 Processori | 0,207288 | 3,995 | 0,999 |
| 5 Processori | 0,165441 | 5,012 | 1,002 |
| 6 Processori | 0,139031 | 5,95 | 0,992 |
| 7 Processori | 0,130027 | 6,362 | 0,909 |
| 8 Processori | 0,103813 | 7,952 | 0,994 |

Da come si può vedere da grafici, all’aumentare del numero dei processori il tempo di esecuzione migliora, si passa da 0,827163 secondi a 0,103813 secondi, lo speed up migliora e l’efficienza rimane piuttosto costante. Bisogna anche considerare che è stato messo un quantitativo di numeri molto grande e quindi, le operazioni che vengono eseguite per il calcolo sono molto maggiori rispetto alle operazioni di comunicazioni tra processori.

## 6.3 Strategia 2 – 200000000 numeri

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| 1 Processore | 0,816409 | 1 | 1 |
| 2 Processori | 0,413718 | 1,971 | 0,986 |
| 4 Processori | 0,207586 | 3,923 | 0,981 |
| 8 Processori | 0,103906 | 7,846 | 0,981 |

In questo caso, sono solo stati utilizzati una quantità di processori che corrisponde a una potenza di 2 per eseguire la strategia 2. Anche in questo caso con l’aumentare del numero di processori, il tempo diminuisce di molto, lo speed up aumenta, tende quasi a raddoppiare e l’efficienza tende a essere uguale a 1. Quindi con un quantitativo di numeri molto alto si ottengono tempi molto buoni.

## 6.4 Strategia 3 – 200000000 numeri

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| 1 Processore | 0,82219 | 1 | 1 |
| 2 Processori | 0,41041 | 2 | 1 |
| 4 Processori | 0,206946 | 3,971 | 0,993 |
| 8 Processori | 0,103851 | 7,904 | 0,988 |

Anche in questo caso è stato possibile usare sono un numero di processori tale che siano una potenza di 2. I risultati sono molto simili alla strategia 2, anche perché l’algoritmo esegue lo stesso numero di istruzioni della strategia 2 e questo porta ad avere risultati molto simili. Lo speed up è quasi in rapporto 1:1 con il numero di processori e l’efficienza tende a essere uguale a 1.

## 6.5 Strategia 1 – 300000000 numeri

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| 1 Processore | 1,227958 | 1 | 1 |
| 2 Processori | 0,621007 | 1,977 | 0,989 |
| 3 Processori | 0,414118 | 2,966 | 0,989 |
| 4 Processori | 0,31044 | 3,961 | 0,990 |
| 5 Processori | 0,248766 | 4,932 | 0,986 |
| 6 Processori | 0,209429 | 5,835 | 0,973 |
| 7 Processori | 0,203388 | 5,876 | 0,839 |
| 8 Processori | 0,155515 | 7,872 | 0,984 |

Da come si può vedere, i risultati sono molto simili con 200000000 numeri, ovviamente il tempo di esecuzione aumenta, ma col crescere dei processori il tempo di esecuzione diminuisce, lo speed up aumenta e l’efficienza rimane quasi costante. Da come si può vedere dalla tabella, si ha un po’ di overhead nel caso in cui l’algoritmo è eseguito con 7 processori. Infatti, da come si nota dalla tabella lo speed up è molto simile con 6 processori e l’efficienza è di meno di 6 processori. Probabilmente ciò è dovuto da un rallentamento o da qualche problema del cluster.

## 6.6 Stategia 2 – 300000000 numeri

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| 1 Processore | 1,2433 | 1 | 1 |
| 2 Processori | 0,620652 | 2 | 1 |
| 4 Processori | 0,311031 | 3,996 | 0,999 |
| 8 Processori | 0,155621 | 7,968 | 0,996 |

Anche in questo caso, comparando i valori della strategia 2 con 200000000 numeri, si nota che i valori come lo speed up e l’efficienza rimangono piuttosto simili. Ovviamente il tempo di esecuzione aumenta. Però all’aumentare dei processori, speed up ed efficienza migliorano di molto.

## 6.7 Stategia 3 – 300000000 numeri

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| 1 Processore | 1,236038 | 1 | 1 |
| 2 Processori | 0,620976 | 1,990 | 0,995 |
| 4 Processori | 0,309699 | 3,987 | 0,997 |
| 8 Processori | 0,15573 | 7,923 | 0,99 |

Anche con la strategia 3, con l’aumentare del problema i tempi salgono ma i valori come lo speed up e l’efficienza rimangono pressoché simili con meno numeri e i tempi continuano a essere molto simili a quella della strategia 2.

# 7 Esempi d’uso

Per lanciare l’esecuzione del calcolo della somma basta aggiungere i parametri corrispondenti per il corretto funzionamento. Per esempio con:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N numbers\_120000000\_strategy\_3\_printer\_0

#PBS -o numbers\_120000000\_strategy\_3\_printer\_0.out

#PBS -e numbers\_120000000\_strategy\_3\_printer\_0.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/project\_sum\_of\_numbers

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/project\_sum\_of\_numbers $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/project\_sum\_of\_numbers $PBS\_O\_WORKDIR/main.c

echo "Eseguo:/usr/lib64/openmpi/1.4-gcc/bin/-machinefile $PBS\_NODEFILE -n 2 $PBS\_O\_WORKDIR/project\_sum\_of\_numbers"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 2 $PBS\_O\_WORKDIR/project\_sum\_of\_numbers 120000000 3 0

si sta indicando che si vuole sommare con 2 processori 12000000 numeri casuali con la strategia 3 e che il processore 0 sarà incaricato a stampare il risultato. Questo è il contenuto del file .out:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

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

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4003895.torque02.scope.unina.it

PBS: job name is numbers\_120000000\_strategy\_3\_printer\_0

PBS: node file is /var/spool/pbs/aux//4003895.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/numbers\_sum /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/main.c

Eseguo:/usr/lib64/openmpi/1.4-gcc/bin/-machinefile /var/spool/pbs/aux//4003895.torque02.scope.unina.it -n 2 /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/project\_sum\_of\_numbers

Executing with 2 processors

I'm the processor 0 - This is the sum 72057594037927936.000000 - Total time elapsed: 0.248866 seconds

In quest’altro esempio si può vedere che si vogliono sommare 100000000 numeri, utilizzando 8 processori, con la strategia 3 e che tutti i processori devono stampare il risultato:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N numbers\_100000000\_strategy\_1\_printer\_minus\_one

#PBS -o numbers\_100000000\_strategy\_1\_printer\_minus\_one.out

#PBS -e numbers\_100000000\_strategy\_1\_printer\_minus\_one.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/project\_sum\_of\_numbers

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/project\_sum\_of\_numbers $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/project\_sum\_of\_numbers $PBS\_O\_WORKDIR/main.c

echo "Eseguo:/usr/lib64/openmpi/1.4-gcc/bin/-machinefile $PBS\_NODEFILE -n 2 $PBS\_O\_WORKDIR/project\_sum\_of\_numbers"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 8 $PBS\_O\_WORKDIR/project\_sum\_of\_numbers 100000000 1 -1

questo è l’output generato:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

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

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4003886.torque02.scope.unina.it

PBS: job name is numbers\_100000000\_strategy\_1\_printer\_minus\_one

PBS: node file is /var/spool/pbs/aux//4003886.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/numbers\_sum /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/main.c

Eseguo:/usr/lib64/openmpi/1.4-gcc/bin/-machinefile /var/spool/pbs/aux//4003886.torque02.scope.unina.it -n 8 /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/project\_sum\_of\_numbers

Executing with 8 processors

I'm the processor 7 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 0 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 1 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 3 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 4 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 2 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 6 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

I'm the processor 5 - This is the sum 107372473003016192.000000 - Total time elapsed: 0.051912 seconds

In quest’ultimo esempio mostra il caso in cui si vogliono inserire manualmente 20 numeri che devono essere sommati con la strategia 1 con 8 processori e stampati dal processore 0. Quindi i numeri non verranno generati casualmente:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N numbers\_20\_strategy\_1\_printer\_0

#PBS -o numbers\_20\_strategy\_1\_printer\_0.out

#PBS -e numbers\_20\_strategy\_1\_printer\_0.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/project\_sum\_of\_numbers

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/project\_sum\_of\_numbers $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/project\_sum\_of\_numbers $PBS\_O\_WORKDIR/main.c

echo "Eseguo:/usr/lib64/openmpi/1.4-gcc/bin/-machinefile $PBS\_NODEFILE -n 2 $PBS\_O\_WORKDIR/project\_sum\_of\_numbers"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile $PBS\_NODEFILE -n 8 $PBS\_O\_WORKDIR/numbers\_sum 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 0

Questo è il file di output generato:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

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

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4004449.torque02.scope.unina.it

PBS: job name is numbers\_20\_strategy\_1\_printer\_0

PBS: node file is /var/spool/pbs/aux//4004449.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/project\_sum\_of\_numbers /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/main.c

Eseguo:/usr/lib64/openmpi/1.4-gcc/bin/-machinefile /var/spool/pbs/aux//4004449.torque02.scope.unina.it -n 2 /homes/DMA/PDC/2022/PCCGPP98C/project\_sum\_of\_numbers/project\_sum\_of\_numbers

Executing with 1 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.000002 seconds

Executing with 2 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.000014 seconds

Executing with 3 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.000092 seconds

Executing with 4 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.000021 seconds

Executing with 5 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.000022 seconds

Executing with 6 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.001800 seconds

Executing with 7 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.024941 seconds

Executing with 8 processors

I'm the processor 0 - This is the sum 55.000000 - Total time elapsed: 0.000061 seconds