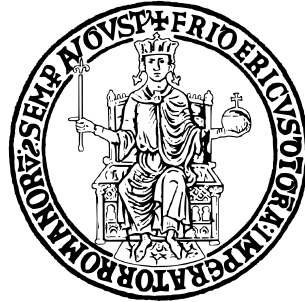


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE 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 N97/0000
Giovanni FALCONE N97/1111
Luigi MANGIACAPRA N97/2222

Anno Accademico 2023-2024

Indice

1	Descrizione del problema	2
2	Descrizione algoritmo	3
2.1	Struttura del programma	3
3	Descrizione routine	5
3.1	Funzioni <i>MPI</i> utilizzate	5
3.2	Funzioni <i>ausiliare</i> utilizzate	5
4	Testing	6
5	Analisi performance	7
6	Source code	8

Capitolo 1

Descrizione del problema

Il goal del problema è calcolare la somma di N numeri in ambiente *parallelo* su architettura **MIMD** (**M**ultiple **I**nstruction **M**ultiple **D**ata) a memoria distribuita, utilizzando la libreria MPI. 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.

Il linguaggio scelto per lo sviluppo dell'algoritmo è il C.

Capitolo 2

Descrizione algoritmo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

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
├── Cmake file
└── ..
dove
```

- `Main.c` è file principale che contiene il main del programma il quale si occupa di verificare se i valori forniti in input sono validi, le inizializzazioni per utilizzare MPI, richiamare l'opportuna funzione dall'header file in base alla strategia scelta e, in fine, mostrare a schermo il risultato ottenuto dalla somma.
- `Strategy.c` è il file che contiene tutte le funzioni necessarie per la somma degli N valori in base alla strategia scelta.

- `Strategy.h` è il file che contiene i vari prototipi.
- ...

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

```
void foo(int a, int b)
```

Descrizione: aaa

Input: bbb

Output: ccc

3.2 Funzioni *ausiliare* utilizzate

Capitolo 4

Testing

Capitolo 5

Analisi performance

Capitolo 6

Source code

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 # include <unistd.h>
4 # include <fcntl.h>
5 # include <sys/types.h>
6 # include <string.h>
7
8 # define BUFFER_SIZE 4096
9
10 static void err_sys(char *msg);
11
12 /**
13  * @brief Create a file object
14  *
15  * @param buffer
16  * @param file_name
17  */
18 static void create_file(char *buffer, char *file_name);
19
20 int main(int argc, char **argv){
21     char first_buffer[BUFFER_SIZE];
22     char second_buffer[BUFFER_SIZE];
23     int fd;
24     ssize_t n_read;
25
26     if(argc != 3){
27         fprintf(stderr, "Usage %s: <file_name> <n>\n", argv[0]);
28         return EXIT_FAILURE;
29     }
30
31     char *FILENAME = argv[1];
32     int N = atoi(argv[2]);
33
34     if((fd = open(FILENAME, O_RDONLY | O_RDONLY)) < 0)
35         err_sys("open error");
36
37     if((n_read = read(fd, first_buffer, N)) < 0)
38         err_sys("read error");
39     first_buffer[n_read] = '\0';
```

```

40     printf("first buffer:\n%s\n\n", first_buffer);
41
42     // in questo modo evito di fare le altre chiamate ed esco ←
direttamente
43     if(N >= lseek(fd, 0, SEEK_END)){
44         printf("The file has been read in its entirety\n");
45         // creo il primo file
46         create_file(first_buffer, "part1");
47         return EXIT_SUCCESS;
48     }
49
50     // se non ho raggiunto la fine del file allora setto il cursore per ←
leggere la seconda parte
51     if(lseek(fd, N, SEEK_SET) < 0)
52         err_sys("lseek error");
53
54     // leggo finche' e' possibile
55     while((n_read = read(fd, second_buffer, BUFFER_SIZE)) == 0);
56
57     if(n_read < 0)
58         err_sys("read error");
59
60     printf("second buffer:\n%s\n\n", second_buffer);
61
62     close(fd);
63
64     // creo il primo file
65     create_file(first_buffer, "part1");
66     // creo il secondo file (nessun controllo perche' non sono uscito ←
dal programma nel primo if)
67     create_file(second_buffer, "part2");
68
69     return 0;
70 }
71
72
73 void create_file(char *buffer, char *file_name){
74     int fd;
75
76     if((fd = open(file_name, O_RDWR | O_CREAT | O_TRUNC, S_IRWXU)) < 0)
77         err_sys("open error");
78
79     if(write(fd, buffer, strlen(buffer)) != strlen(buffer))
80         err_sys("write error");
81
82     fprintf(stdout, "File %s created!\n", file_name);
83
84     close(fd);
85 }
86
87
88 void err_sys(char *error) {
89     perror(error);
90     exit(1);

```

```
91 }  
92
```