# Task 4: Fork() and Pipes

Jana Saleh 900204192
Mariam Dahab 900192441
Muhammad Azzazy 900202821

# TABLE OF CONTENTS

01

## Roles

Roles of every team member

02

## Pseudocode

Descriptions and pseudocode of the two compute functions

03

## Examples

Examples showing the compute functions producing correct results

04

## Figure

A figure illustrating processes and pipes with the flow of data

05

## Test Design

How the two tests were designed and what are the ranges chosen

06

## Curve Smoothing

Method of curve smoothing

# TABLE OF CONTENTS

07

## Comparison & Answers

Comparison results and
answers to the two questions

01

# Roles

Roles of every team member

| **Name** | **Role** |
|---|---|
| Muhammad Azzazy | Implemented **sequential_compute** |
| Mariam Dahab | Implemented **parallel_compute** |
| Jana Saleh | Implemented time computation and did the graphs and Microsoft Excel sheets |

02

# Pseudocode

Description and pseudocode of the two compute functions

# sequential_compute

```
SEQUENTIALCOMPUTE(filepath, f)
        N ← 0
    fh ← fopen(filepath, ‘r’)
    if fh == NULL
        perror(‘Unable to open the file’)
        exit(1)
    else
        while (!feof(fh))
                do fscanf(fh, “%d”, &arr[N])
                N ← N + 1
        fclose(fh)
        if N > 1
                then if f == add
                        then result ← add(arr[N-1], arr[N])
                                for i ← N-2 to 1
                                        do result ← add(arr[i],
    result)
                        else if f == multiply
                                result ← multiply(arr[N-1], arr[N])
                                for i ← N-2 to 1
                                        do result ← multiply(arr[i],
    result)
        else if N == 1
                then if f == add
                        then            result ← 0
                    result ← add(arr[N], result)
                        else if f == multiply
                                result ← 1
                                result ← multiply(arr[N], result)
    return result
```

# parallel_compute

After reading from file and checking for all the edge cases

A for loop is used to create n_prco -1 child process each is responsible for calculating x integers and writing the result to its pipe

After all child processes are done the parent process reads from each pipe and calculates the final result

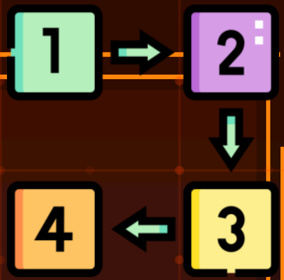
```
parallel_compute(filepath,n_proc,fun_point) {
N->0
//open file
  while (!feof(infile))
    Read from file
     N++;
 x<- N/n_proc -1
  for i=0  to n_proc -1
  {
    //create pipe

    ps_id[i] <- fork();

    if ps_id[i]== 0
      start = i * x;
      if (i != n_proc - 2)
        stop = start + x
      else
        stop <- N
     Partcial_result <- data[start]
     for z=start+1 to stop
        Partcial_result <-
       fun_ptr(partcial_result, data[z])
    Write partial result -> pipe[i]
    exit()
}
wait()
  for (int j = 0; j < n_proc-1; j++)
      Read pipe[j] -> total[i];
    if(j==0)
      result=total[j];
    else
      result=(*fun_ptr)(result,total[j]);
  printf("Final Result : %f \n",result);
}
```

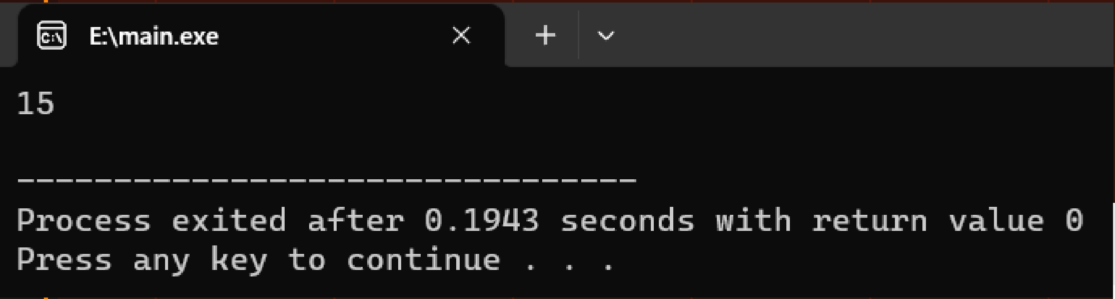
03

# Examples

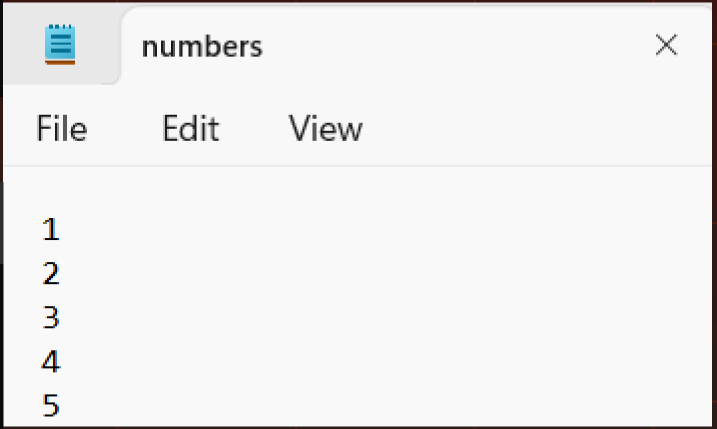
Examples showing the compute functions producing correct results

# sequential_compute

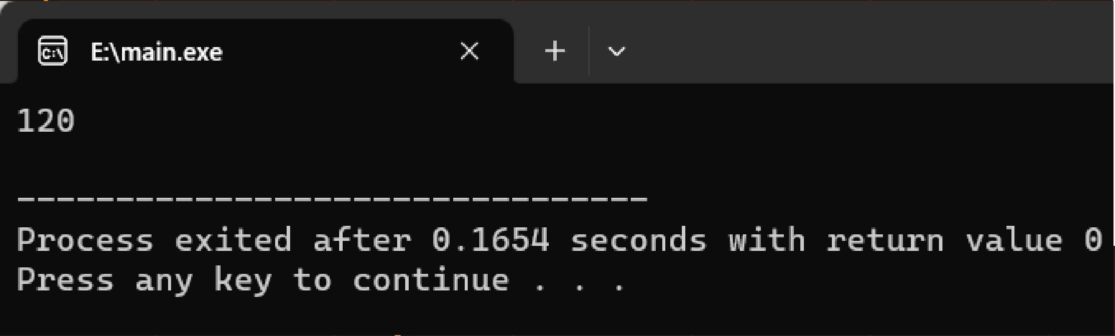




```
15

--------------------------------
Process exited after 0.1943 seconds with return value 0
Press any key to continue . . .
```

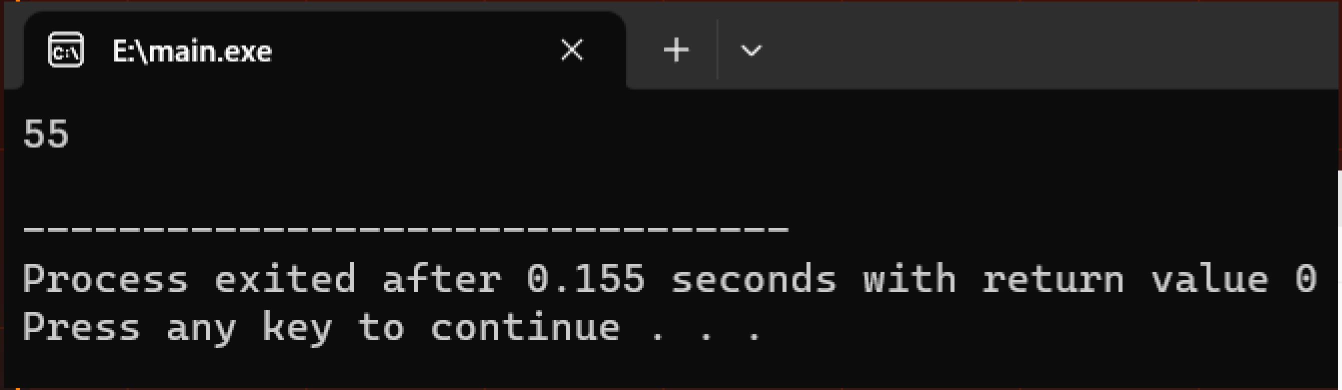


```
1
2
3
4
5
```

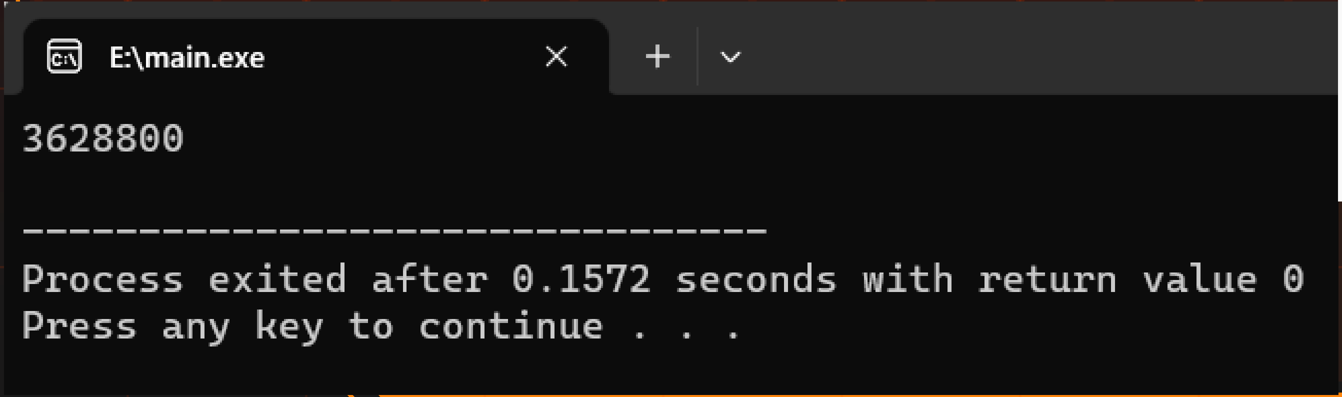


```
120

--------------------------------
Process exited after 0.1654 seconds with return value 0
Press any key to continue . . .
```

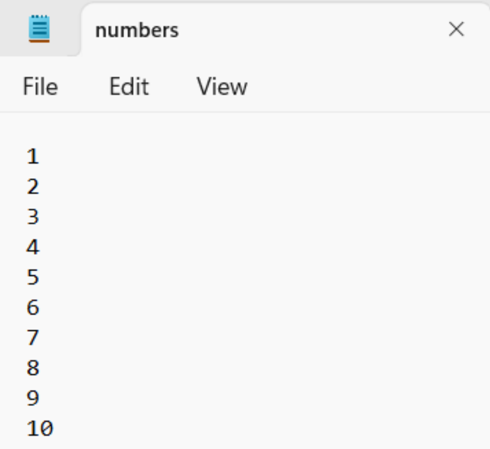
# sequential_compute



```
55

--------------------------------
Process exited after 0.155 seconds with return value 0
Press any key to continue . . .
```



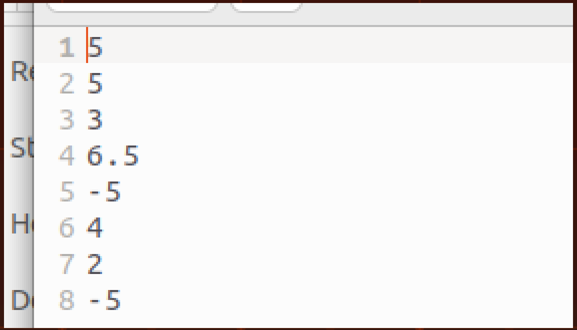
```
3628800

--------------------------------
Process exited after 0.1572 seconds with return value 0
Press any key to continue . . .
```



```
1
2
3
4
5
6
7
8
9
10
```

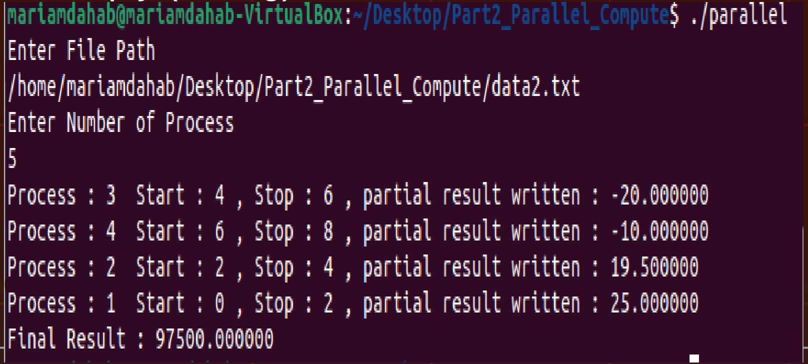
# parallel_compute

data2.txt



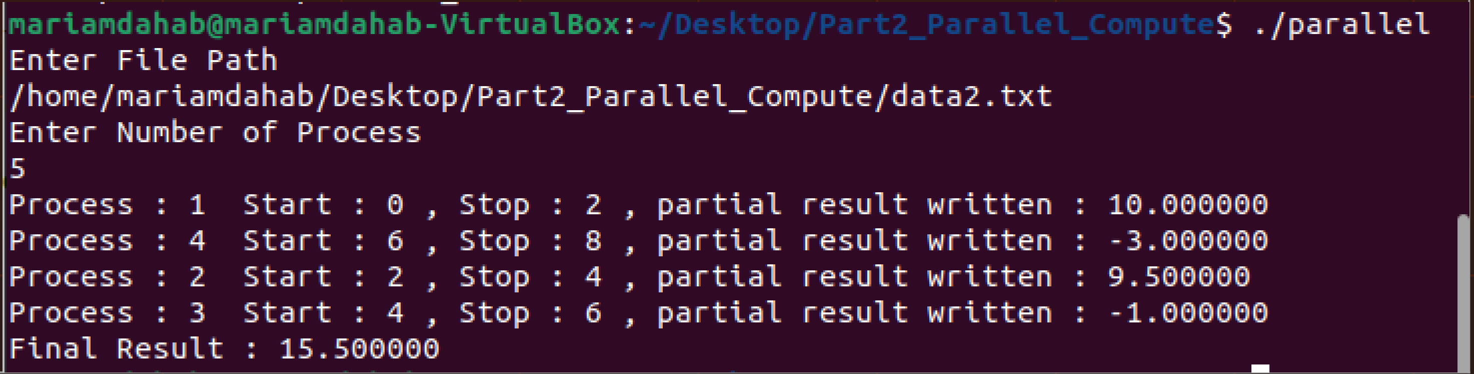
```
5
5
3
6.5
-5
4
2
-5
```

f=multiply (debug)



```
mariamdahab@mariamdahab-VirtualBox:~/Desktop/Part2_Parallel_Compute$ ./parallel
Enter File Path
/home/mariamdahab/Desktop/Part2_Parallel_Compute/data2.txt
Enter Number of Process
5
Process : 3  Start : 4 , Stop : 6 , partial result written : -20.000000
Process : 4  Start : 6 , Stop : 8 , partial result written : -10.000000
Process : 2  Start : 2 , Stop : 4 , partial result written : 19.500000
Process : 1  Start : 0 , Stop : 2 , partial result written : 25.000000
Final Result : 97500.000000
```

f=add (debug)



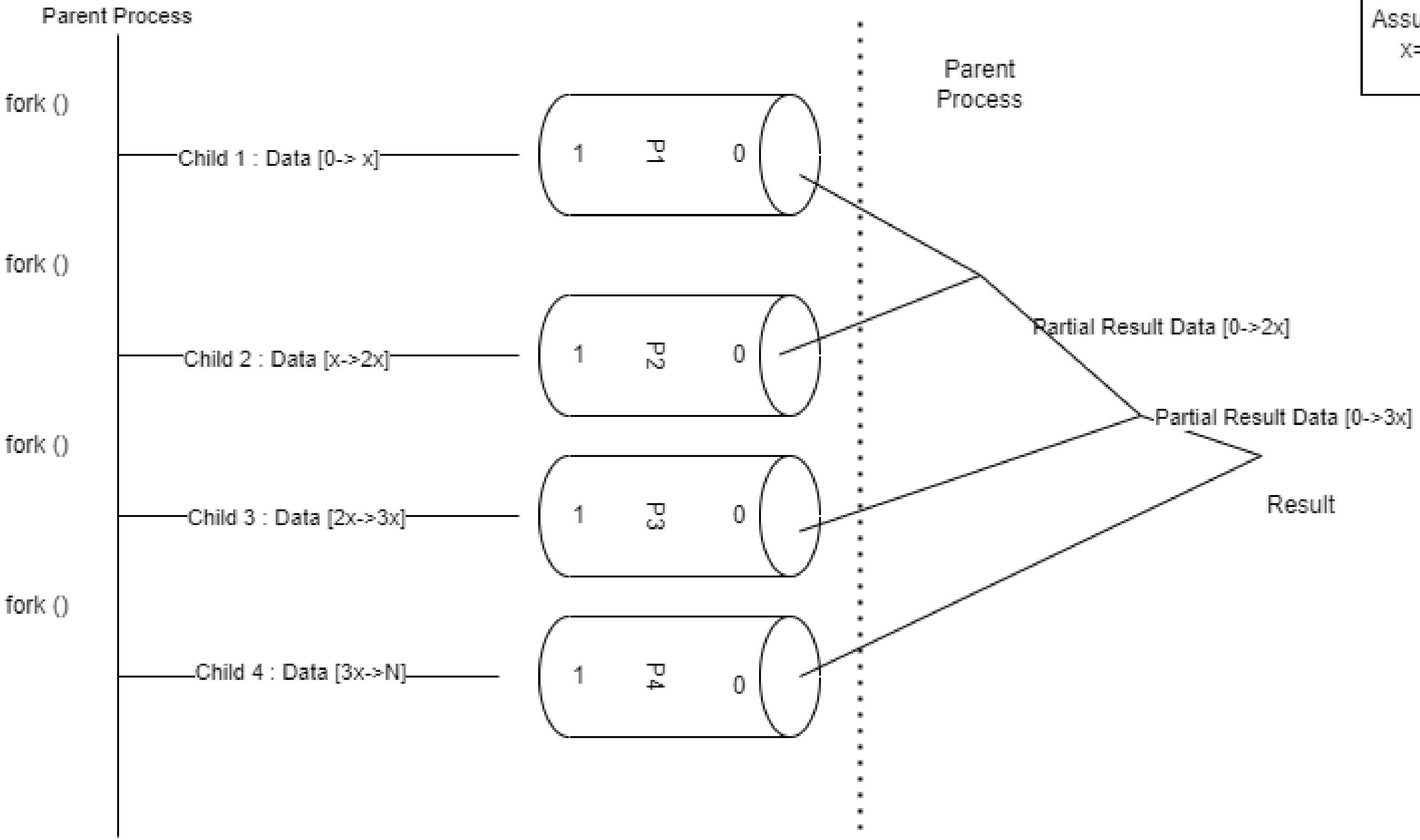
```
mariamdahab@mariamdahab-VirtualBox:~/Desktop/Part2_Parallel_Compute$ ./parallel
Enter File Path
/home/mariamdahab/Desktop/Part2_Parallel_Compute/data2.txt
Enter Number of Process
5
Process : 1  Start : 0 , Stop : 2 , partial result written : 10.000000
Process : 4  Start : 6 , Stop : 8 , partial result written : -3.000000
Process : 2  Start : 2 , Stop : 4 , partial result written : 9.500000
Process : 3  Start : 4 , Stop : 6 , partial result written : -1.000000
Final Result : 15.500000
```

04

# Figure

A figure illustrating processes and pipes with the flow of data

Assuming n_proc =5
x=N/(n_proc-1)



Parent Process

fork ()

Child 1 : Data [0-> x]

fork ()

Child 2 : Data [x->2x]

fork ()

Child 3 : Data [2x->3x]

fork ()

Child 4 : Data [3x->N]

Parent Process

Partial Result Data [0->2x]

Partial Result Data [0->3x]

Result

05

# Test Design

How the two tests were designed and what are the ranges chosen

# Chosen Ranges

## First Graph

- N Ranges from 20 to 900
- Fix numbers of process to 20

## Second Graph

- Number of process ranges from 1 to 990 processes
- N fixed to 300

06

# Curve Smoothing

Method of curve smoothing

# Curve Smoothing

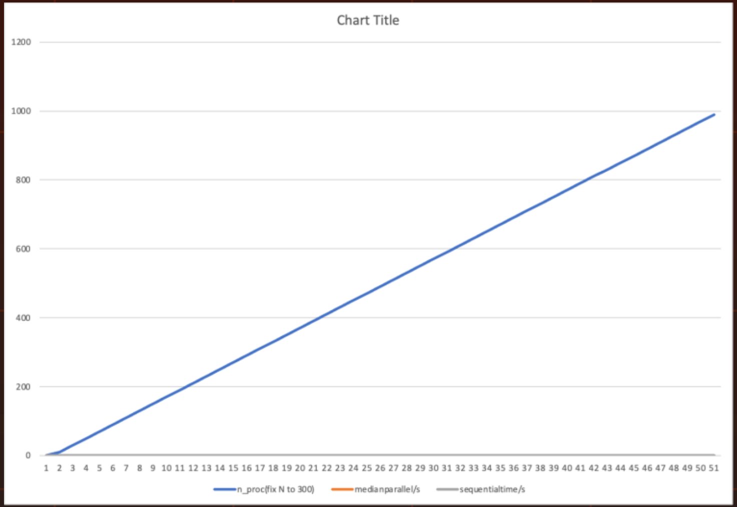
- Smoothing curve using the 4 point median technique by adding the last 4 test results, then getting the average of the median result
- It gives an indication of the overall trend of the results
- Appling the smoothing curve option in Microsoft Excel

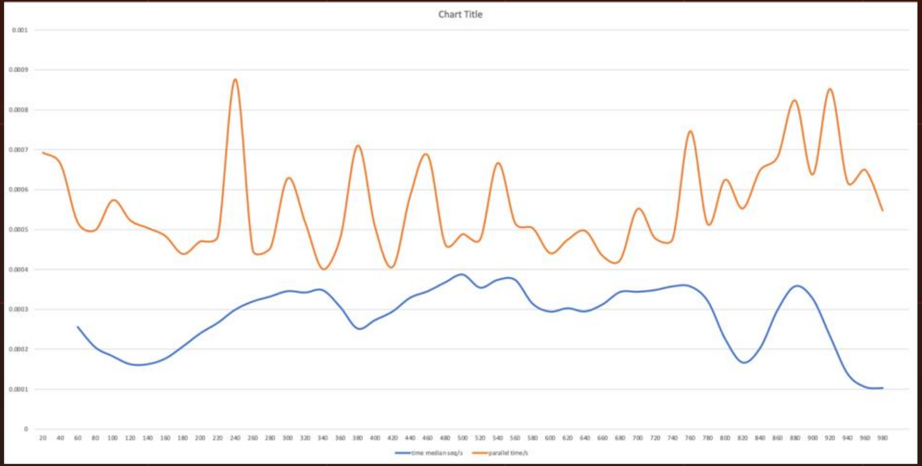
07

# Comparison & Answers

Comparison results and answers to the two questions

# Comparison Results





# Answers to the Questions

- We fixed nproc to 2 in the first question as when we increased it the performance did not increase when it was greater than 2
- The expected results were that parallel outperform the sequential when N increases, however the results were close
- We fixed N to 300 and nproces ranges from 10 to 900, it was expected that as the number of process increases the performances increases but that was not the case

# THANKS

Do you have any questions?
janasaleh@aucegypt.edu
mariamhdahab@aucegypt.edu
muhammad-azzazy@aucegypt.edu

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon** and infographics & images by **Freepik**

Please keep this slide for attribution