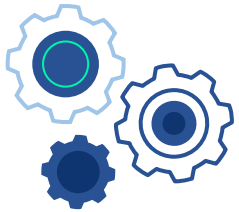
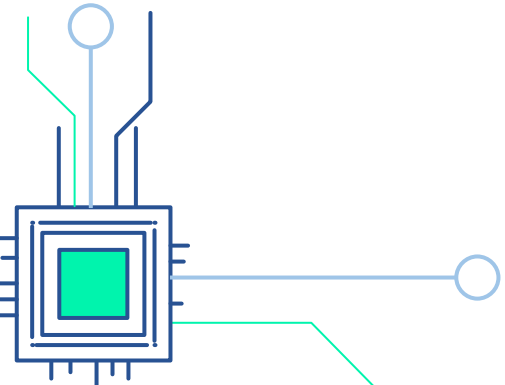


OS TASK 4 - INTERPROCESS COMMUNICATION USING PIPES

Farah Kabesh - 900191706
Karim Sherif - 900191474



PREVIOUS MISTAKES

- Missing Descriptions.
- Only 1 pipe used.
- Too much text per slide.
- n_proc was fixed to a value not correctly associated with machine's CPU.
- No valid justification for time decreases as N increases.

ROLES OF TEAM MEMBERS

Farah

- ❏ Sum
- ❏ Parallel Sum
- ❏ Presentation

Karim

- ❏ Sum
- ❏ Parallel Sum
- ❏ Presentation

PSEUDO-CODE OF SUM

```
sum(n)
    s=0
    for i=1 to i ← n
        s=s+i
```

DESCRIPTION OF SUM

- Function takes positive integer N and returns the sum of numbers from 1 to N.
- S is initialized to 0.
- A for loop is used to calculate the sum using s = s+i.

PSEUDO CODE OF PARALLEL SUM

PART 1

```
parallel_sum_pipes(n_proc, n)

    Initialize:

    parent_pid = getpid()

    pid_t child_pid, wpid

    s = 0, inpipe_sum = 0

    status = 0
```

PSEUDO CODE OF PARALLEL SUM

PART 2

```
Initialize:

interval = n/n_proc
start = 1
end = interval
p[n_proc][2]

for i=0 to i ← n_proc

    if pipe p[i] is equal to -1

        print 'Failed to create pipe'
        exit
```

PSEUDO CODE OF PARALLEL SUM

PART 3

```
for i=0 to i ← n_proc

    if getpid() is equal to parent_pid AND child_pid = fork()
    is equal to 0

        if i+1 is equal to n_proc
            end = n

        inpipe_sum = 0
```

PSEUDO CODE OF PARALLEL SUM

PART 4

```
    for start ← end
        inpipe_sum = inpipe_sum + start

    close(p[i][0]

    write(p[i][1], &inpipe_sum, sizeof(unsigned long long int))
start = end + 1

end = end + interval
```

PSEUDO CODE OF PARALLEL SUM

PART 5

```
if get_pid is equal to parent_pid
    while wpid = wait(&status) is greater than 0
    for i=0 to i ← n_proc
        read(p[i][0], &inpipe_sum, sizeof(unsigned long long int))
        s = s + inpipe_sum
        close(p[i][0]), close(p[i][1])
else exit
```

DESCRIPTION OF PARALLEL SUM

- Function takes number of processes (n_proc) and a positive integer N.
- Interval is equal to N/n_proc.
- 2 pipes are used.
- Equal intervals are divided among partial sums.
- The partial sum for each interval is calculated.
- The total sum is calculated by summing all partial sums.

HOW TEST 1 WAS DESIGNED & WHY?

FIXING N_PROC

- Machine Name: Intel® Core™ i7-6700 CPU @ 3.40GHz × 8.
- Machine Specifications:
  - Number of cores = 4.
  - Number of threads = 8.

- Therefore n_proc = 3

- In order to obtain readings:
  - for (n = 1; n < 1000000; n+=5000)

HOW TEST 2 WAS DESIGNED & WHY?

FIXING N

- n = 100000.

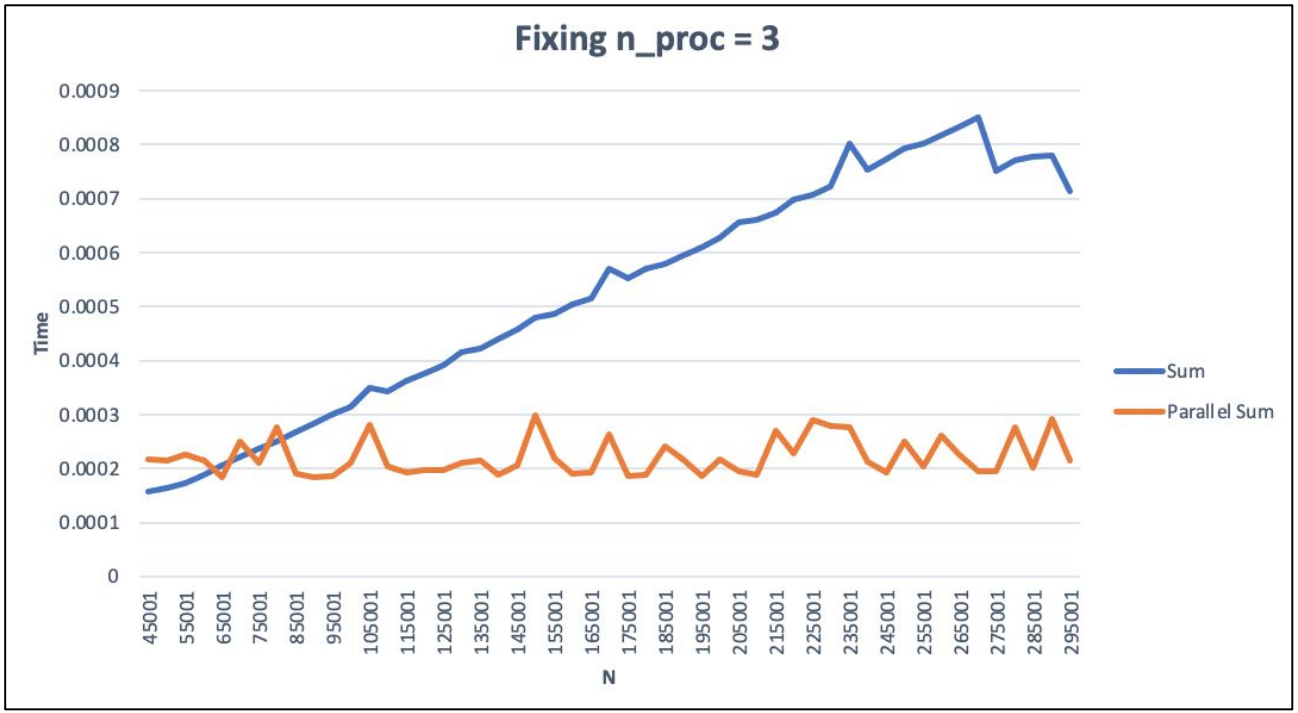
- In order to obtain readings:
  - for (j = 1; j< 150; j++)

Note: j represents number of processes.

QUESTION 1

If we fix *n_proc*, then at which value of N does parallel_sum outperform sum?

GRAPH 1


Fixing n_proc = 3
0.0009
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
Time
45001
55001
65001
75001
85001
95001
105001
115001
125001
135001
145001
155001
165001
175001
185001
195001
205001
215001
225001
235001
245001
255001
265001
275001
285001
295001
N
Sum
Parallel Sum

ANSWER TO QUESTION 1

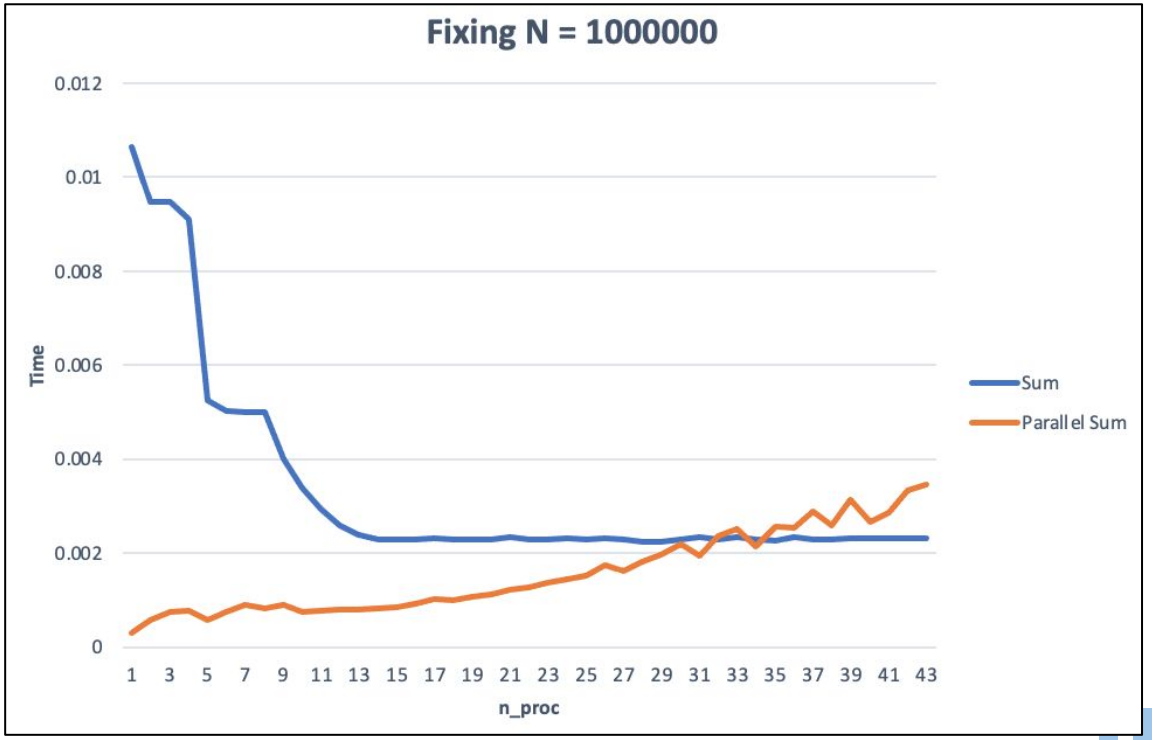
- N = 85001.

- Since N increases and the number of processes is the same, each process will have to handle a larger interval each time meaning that more time is needed to calculate the sum.
- Therefore, as the interval increases, time taken for execution using pipes is less than time taken using sum.

QUESTION 2

If we fix *N* to a large number, then at which value of n_proc does sum outperform parallel_sum?

GRAPH 2


Fixing N = 1000000
0.012
0.01
0.008
0.006
0.004
0.002
0
Time
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43
n_proc
Sum
Parallel Sum

ANSWER TO QUESTION 2

- n_proc = 32.

- As n_proc increases, the overhead of creating new processes increases which leads to more time needed to compute the sums. Therefore, time taken for execution using sum is takes the least time; followed by memory, pipes, and finally threads.