**Project 1: Report**

**Swapnil U. Shinde**

Implementation of Shearsort algorithm with block decomposition:

Followed the implementation details and pseudo code given in reference research paper.

Input : Input is a 1 dimensional array which needs to be mapped into 2d mesh.

**BLOCK DECOMPOSITION:**

Given n1/2\*n1/2 mesh with p processors available, each processor will get n/p processors. I divided them into p blocks.

One node works as a master node which coordinates with all p-1 slave nodes.

If number of elements are not possible to divide into perfect square matrix, then I divided into 2\*(n/2) matrix. If number of elements are odd and cannot be mapped into matrix then it is padded with '-1' elements.

Formula for # of padding elements e - (p - n%p )

p- # of processors and n - # of elements

after adding these padding elements total number of elements in array becomes n+e.

this n+e array is divided for p-1 slave nodes for parallel shear sort processing.

Each array receives message from master node with elements to be sorted locally and number of rows, columns. (TAG type = 0)

Step 2:

**SHEARSORT INDIVIDUAL BLOCKS:**

Each slave node receives element array,# of rows and # of columns. Slave node executes shear sort using odd even sort for sorting rows and columns. Once the shear sort iterations are over, every node prepares message destined to master node with sorted element array.

Step 3:

**2-D Merge**

Master node waits for receiving response from all slave nodes and when it receives, it starts with process of

2-D merging. I implemented the pseudo code for 2D merge given in research paper.

1) Columns sort (I used same even odd sorting algo. as given in requirement)

2) Two iterations of shear sort algorithm.

It gives sorted output in snake indexing pattern. I have converted it back into 1-D array in ascending order.

I removed all padding elements (-1) which were added for shear sort implementation.

Output -

Sorted array in both snake indexed format and linear array format.

**Implementation details -**

**matrix\_decompose(array,count,numSlaves,&row,&col) -** Receives linear array and map it into 2D mesh. It adds padding elements if needed. Returns array ordered as per the sub matrices(tasks).It determines the number of rows and columns in each sub matrix.

Messages passed -

MPI\_Send(row\_col\_info,2,MPI\_INT,i,0,comm);

Row col information to each slave processor.

MPI\_Send(sendbuff,row\*col,MPI\_INT,i,1,comm);

row\*col elements to sort locally destined to every slave node.

MPI\_Send(rcvbuff,rcv\_count,MPI\_INT,0,2,comm);

sorted sub matrix created at slave node destined to master node.

**sheer\_sort\_function(int \*ptr,int nRows,int nColumns) -** Recceives array pointer which needs to sort at slave node.

**odd\_even\_function(int \*ptr,int length) -** Perform odd even sort which is used by slave nodes for sorting purpose.

**merge\_2D\_function(int \*ptr,int nRows,int nColumns)** - It performs 2D merge pseudo code as explained above.

Que #1

α = 0.000035 s

β = 0.000072 s

Program for this is ping\_pong\_V2.c and output is stored in alpha\_beta\_output.txt

Que # 3

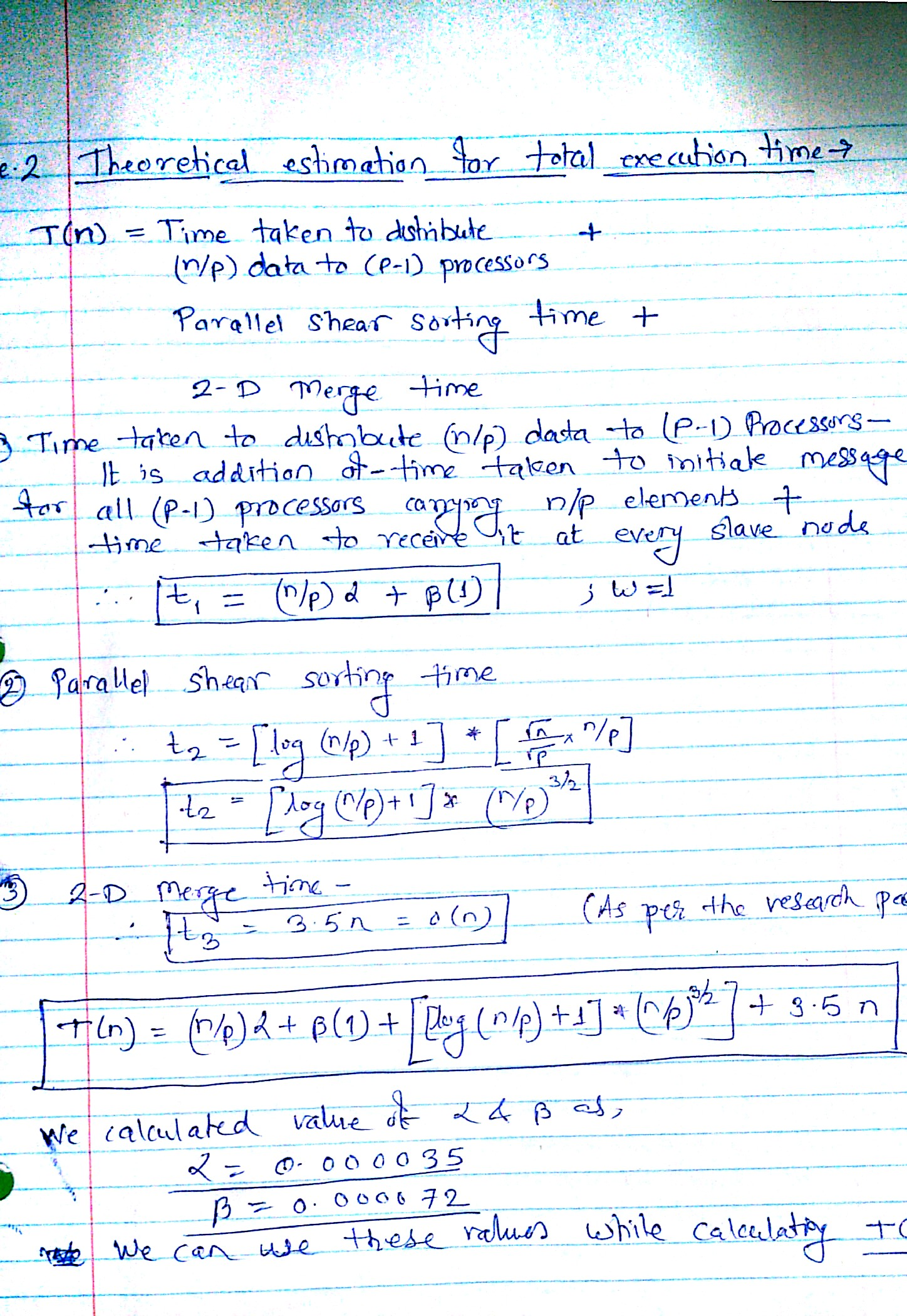
Program name is main2Copy.c

Que # 4

Processors

|  |  |  |  |
| --- | --- | --- | --- |
|  | **4** | **9** | **16** |
| **16** | **3684 µs** | 17416 µs | 61689 µs |
| **64** | **2990 µs** | 24069 µs | 61975 µs |
| **256** | **5080 µs** | 19364 µs | 73689 µs |
| **1024** | **10217 µs** | 17155 µs | 72273 µs |
| **4096** | 92274 µs | **74860 µs** | 94204 µs |
| **16384** | 8559433 µs | **383449 µs** | 439616 µs |

The least time consuming processor is highlighted.



Que # 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Processors** |  |  |  |
|  | **4** |  | **9** |  | **16** |  |
|  | Estimated no. of **steps** | Measured | Estimated no. of **steps** | Measured | Estimated no. of **steps** | Measured |
| **16** | 80 | 3684 | 60.3 | 17416 | 57 | 61689 |
|  |  |  |  |  |  |  |
| **64** | 544 | 2990 | 296.6 | 24069 | 248 | 61975 |
|  |  |  |  |  |  |  |
| **256** | 4480 | 5080 | 1780 | 19364 | 1216 | 73689 |
|  |  |  |  |  |  |  |
| **1024** | 40448 | 10217 | 13086.8 | 17155 | 7168 | 72273 |
|  |  |  |  |  |  |  |
| **4096** | 374784 | 92274 | 109776 | 74860 | 51200 | 74204 |
|  |  |  |  |  |  |  |
| **16384** | 3465216 | 855943 | 976213 | 383449 | 417792 | 439616 |

This table includes estimated steps calculated from time complexity derived in Q2.It shows that as estimated steps from theoretical model increases my measured time is also increases. So my results are consistent with theoretical results.

We could assume the cycle frequency of machine and could approximate theoretical time with certain limits. Doing so we can directly compare theoretical value with experimental results.

Que # 6

My answers are consistent with theoretical estimates. Time taken by my implementation takes a bit longer than in theory but it is consistent for all results.

Table in Q4 clearly shows that P should not be very large as it affects the overall performance. Time spent in distributing tasks and merging results is much expensive than performing sequentially.

The ideal values of p increases slowly with high increase in 'n'. (Best p values for different n are shoen in bold). Therefore p should be as small as possible.