Assignment 1

Question 1:

This program is designed to find a password that matches a given hash value using the brute force approach. Using MPI we divide the number of combinations between ‘n’ number of processes to speed up the process of finding the match. Without using MPI if the program is executed on 1 process, then only 1 combination is tried at a time so it would take a long time to check 64x64x64x64 numbers of combination. But if we distributed this over 40 processes then the process becomes 40 times faster as 40 combinations are being checked at a time.

To perform this, I dynamically selected the range of the outer loop based on the rank of the process.

Let total no. of total of combinations = X

Let number of processes = N

Divide the number of combinations among all the processes then each process gets X/N.

Let X/N = Y

Then,

for process 0 the range will be I = 0; I < Y * (1).

for process 1 the range will be I = Y; I < Y * (2).

for process 2 the range will be I = 2*Y; I < Y * (3).

So, if I had to generalize:

for process n it would be I = my_rank * Y; I < Y * (my_rank + 1).

To signal all the other processes that the password has been found I am using MPI_Allreduce() to perform a Logical OR operation and make the result available to all processes.

All the processes will keep a local version of the flag and the result will be stored in the global flag based on which I am exiting all the processes.

Job Details:

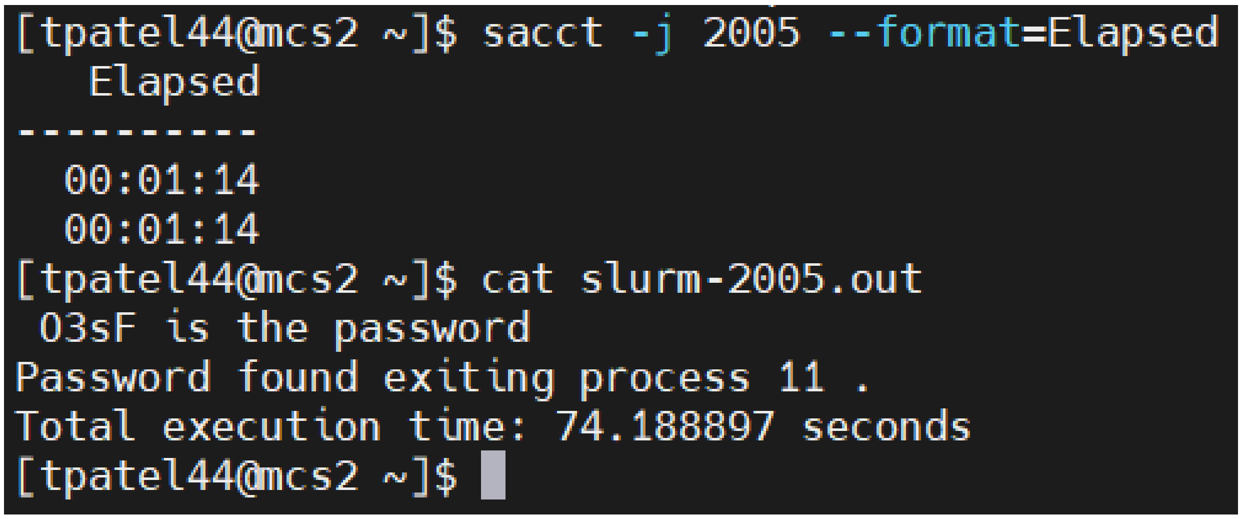
Job Number: 2005

Directory: /home/tpatel44

Job Runtime: 1 minute 14 seconds

Password obtained: O3sF

Screenshots:



```
[tpatel44@mcs2 ~]$ sacct -j 2005 --format=Elapsed
   Elapsed
----------
  00:01:14
  00:01:14
[tpatel44@mcs2 ~]$ cat slurm-2005.out
 O3sF is the password
Password found exiting process 11 .
Total execution time: 74.188897 seconds
[tpatel44@mcs2 ~]$
```

Question 2:

In this question we are dividing a large matrix and distributing it over different processes. Given that matrix size is divisible by the number of processes.

Let the matrix size = N

Let the no. of process = P

To send the matrix block by block to all processes and to keep memory usage efficient I am only reading the part of the matrix which I am sending to all processes. Once the part of the matrix is sent, I read another part of the matrix.

The lines of the matrix to read at a time = N/P = step_size

So, the size of the first Row_block to read = step_size * N

Size of the block sent to other processes = step_size * step_size = block_size

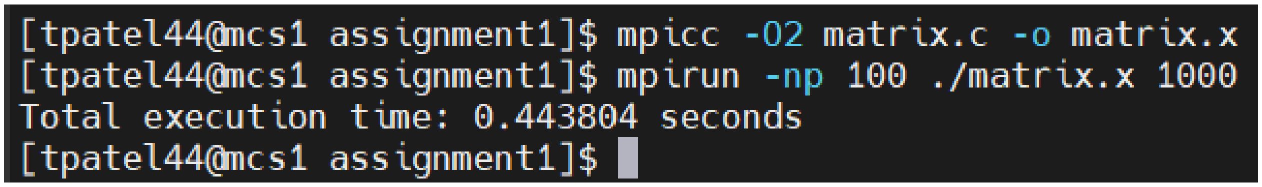
Now the number of steps of size 'steps_size' will be equal to P because it's a square matrix. So, perform the above logic in a loop that runs P times. Inside this loop create a buffer to read the matrix step by step with 'step_size' lines a time. Divide that buffer into blocks of size 'block_size' and send those to corresponding procceses. Now invert the above process for write function as its just a reverses process of reading and distributing the matrix. So, the logic is almost the same.

Time taken for different processes for matrix size 1000.

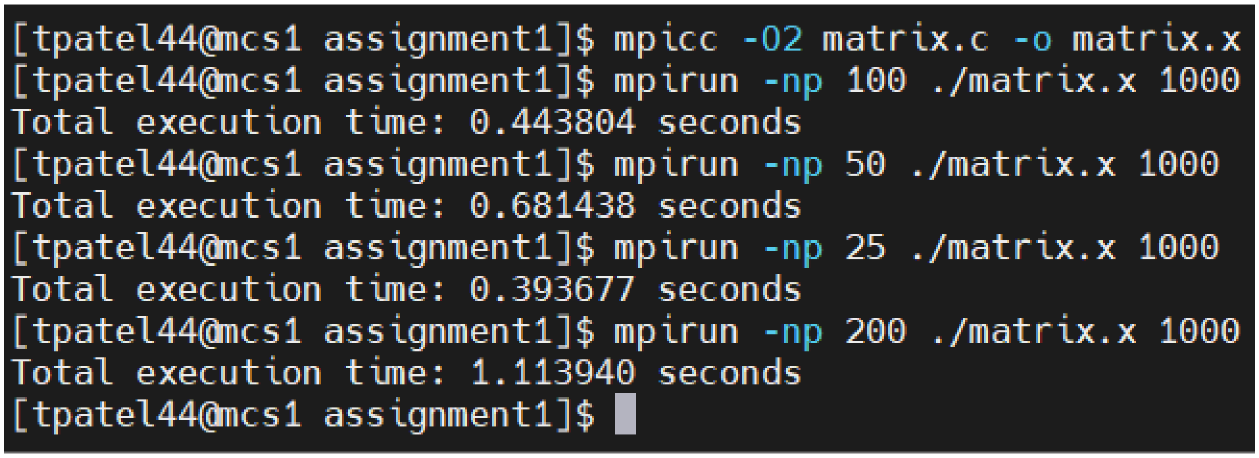
No. of processes	**Execution Time (seconds)**
200	**1.1**

100	**0.44**
50	**0.68**
25	**0.39**

Screenshots:



```
[tpatel44@mcs1 assignment1]$ mpicc -O2 matrix.c -o matrix.x
[tpatel44@mcs1 assignment1]$ mpirun -np 100 ./matrix.x 1000
Total execution time: 0.443804 seconds
[tpatel44@mcs1 assignment1]$
```



```
[tpatel44@mcs1 assignment1]$ mpicc -O2 matrix.c -o matrix.x
[tpatel44@mcs1 assignment1]$ mpirun -np 100 ./matrix.x 1000
Total execution time: 0.443804 seconds
[tpatel44@mcs1 assignment1]$ mpirun -np 50 ./matrix.x 1000
Total execution time: 0.681438 seconds
[tpatel44@mcs1 assignment1]$ mpirun -np 25 ./matrix.x 1000
Total execution time: 0.393677 seconds
[tpatel44@mcs1 assignment1]$ mpirun -np 200 ./matrix.x 1000
Total execution time: 1.113940 seconds
[tpatel44@mcs1 assignment1]$
```