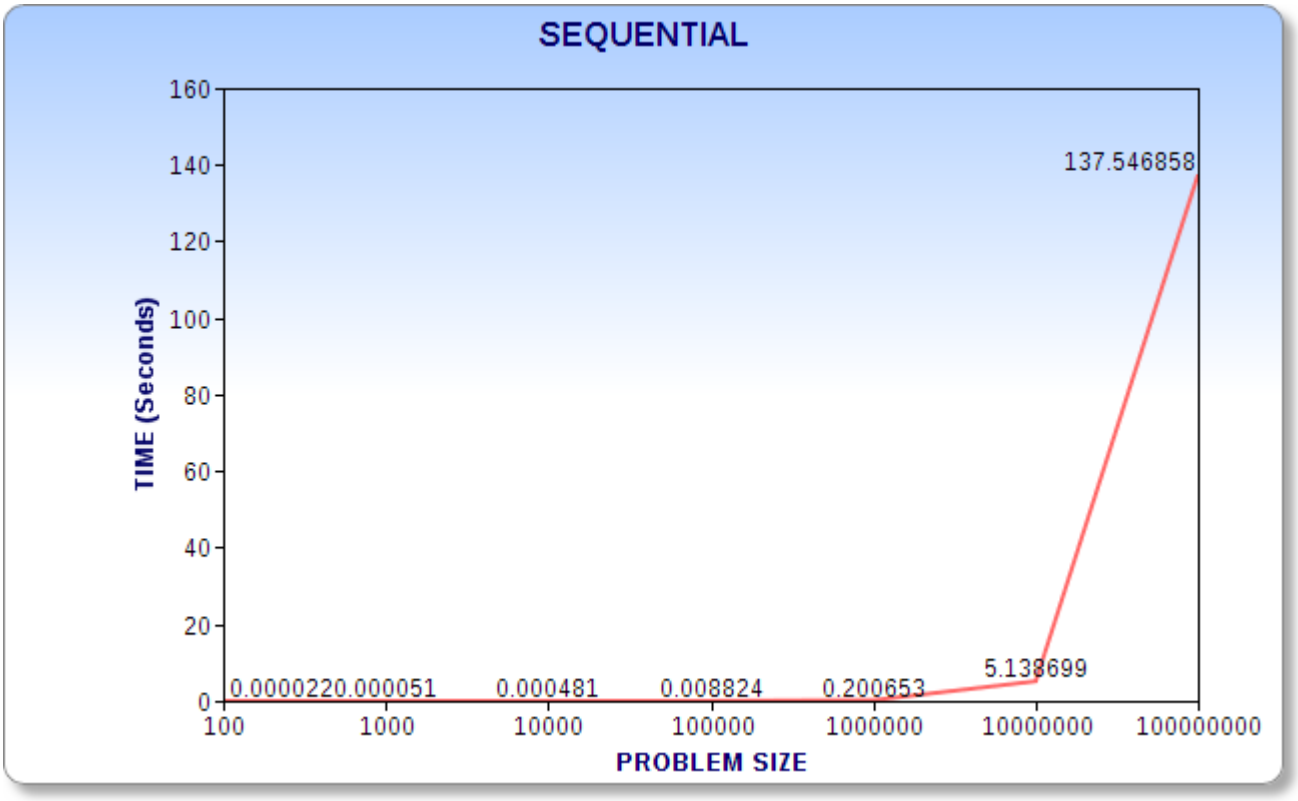
Rakesh Baingolkar
50097576

ASSIGNMENT 1 REPORT

SEQUENTIAL

Filename: PDP1Sequential.c
Makefile: make_seq
Command to run the makefile: make -f make_seq
Command to run the object file: ./sequential [PROBLEM_SIZE]
Sample: ./sequential 1000
SLURM FILE: SLURM_SEQUENTIAL.sh

Sequential file is basic program in which the for loop runs two times and finds the prime numbers upto the given Problem_Size. The graph when we increase the Problem_Size is shown below.

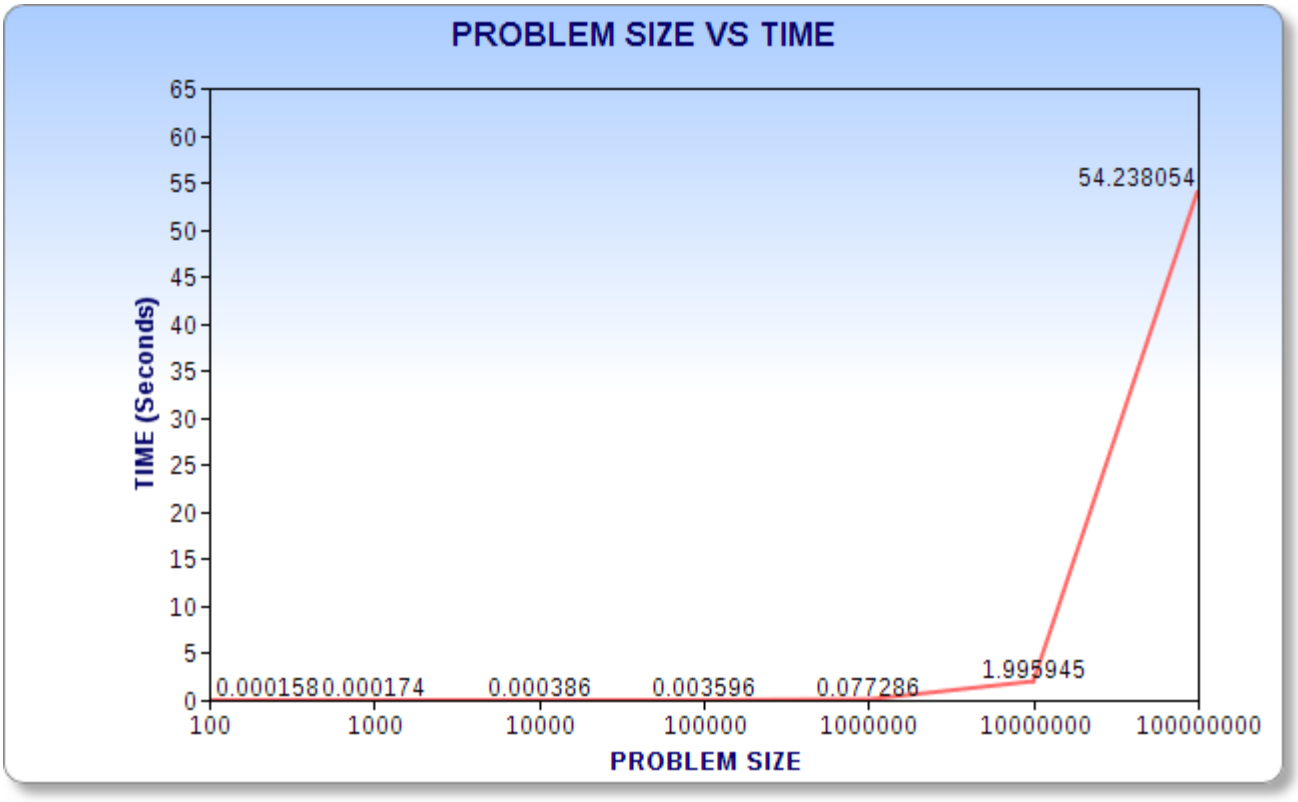


As we see from the graph the Time increases with increase in the Problem Size which is logical.

OpenMP

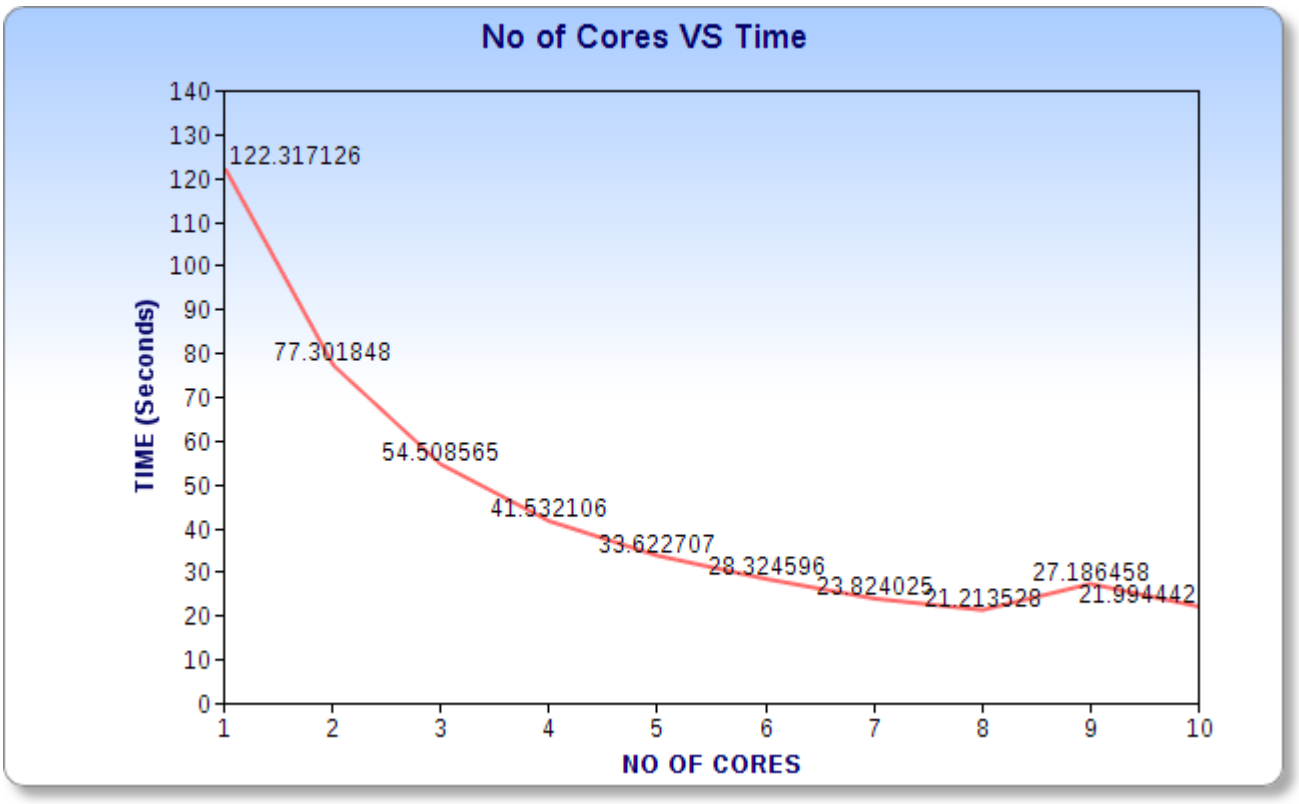
Filename: PDP1OPENMP.c
Makefile: make_openmp
Command to run the makefile: make -f make_openmp
Command to run the object file: ./openmp [NO_OF_CORES(Redundant)] [PROBLEM_SIZE]
Sample:./openmp 2 10000
SLURM FILE: SLURM_OpenMP.sh (NO_OF_CORES edit in SLURM FILE)

In this file I used OpenMP to run multiple threads in the loop section of the problem to carry out the task faster as compared to sequential.
The shared variable is the problem_size and the count is passed as a parameter to the reduction function and the largest is kept track of by putting it in the omp critical.
In the following graph we see the increase in the problem size vs the time.



Above graph has no_of_cores constant i.e. 3 and we see that the time taken to complete the task increases as we increase the Problem size.

The graph below is the graph where we increase the nodes to compute the operation and the Problem size is kept constant.

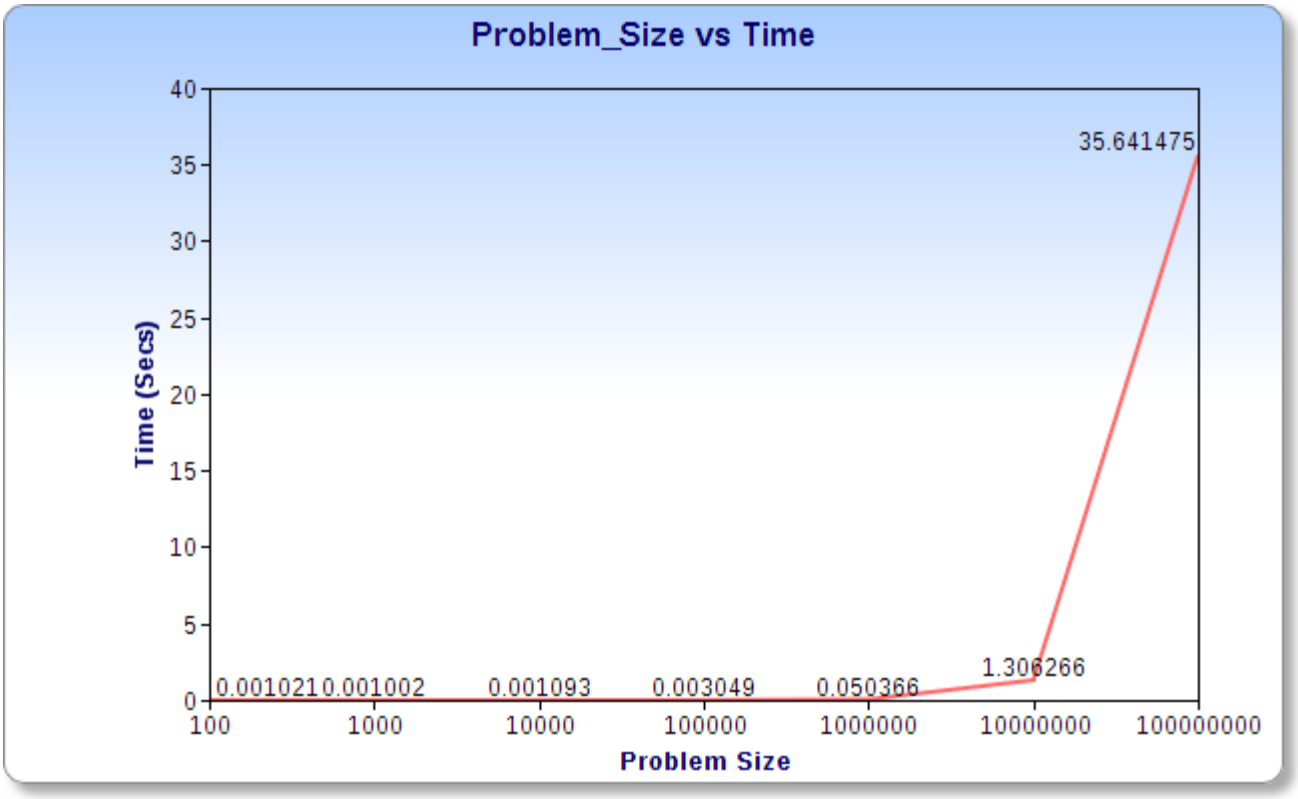


From the graph we can see that the time decreases as we increase the no of threads from 1 to 8 and from 8 it goes on incrasing. The decrease from 1 to 8 can be explained by the traditional way of task being divided into parts and the parts running parallely saving time. The increase in the time for 9 and 10 suggests the context switching overhead due to large number of threads in one node which takes considerable amount of time.

MPI

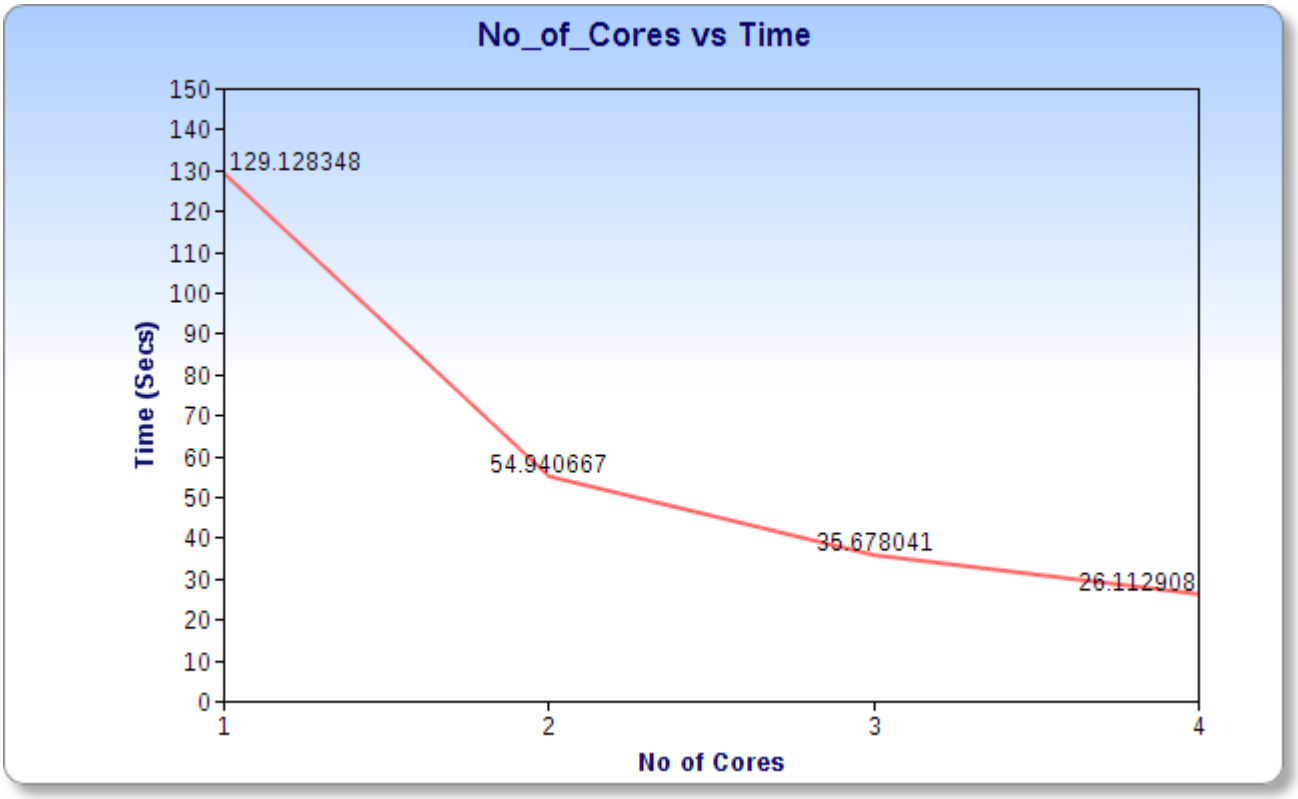
Filename: PDP1MP1.c
Makefile: make_mp1
Command to run the makefile: make -f make_mp1
Command to run the object file: ./mpI [PROBLEM_SIZE]
Sample: ./mpI 1000
SLURM FILE: SLURM_MPI_.sh (NO_OF_NODES edit in SLURM FILE)

In MPI we divide the task into nodes using the send and receive messages between the nodes thus dividing the task making it faster.



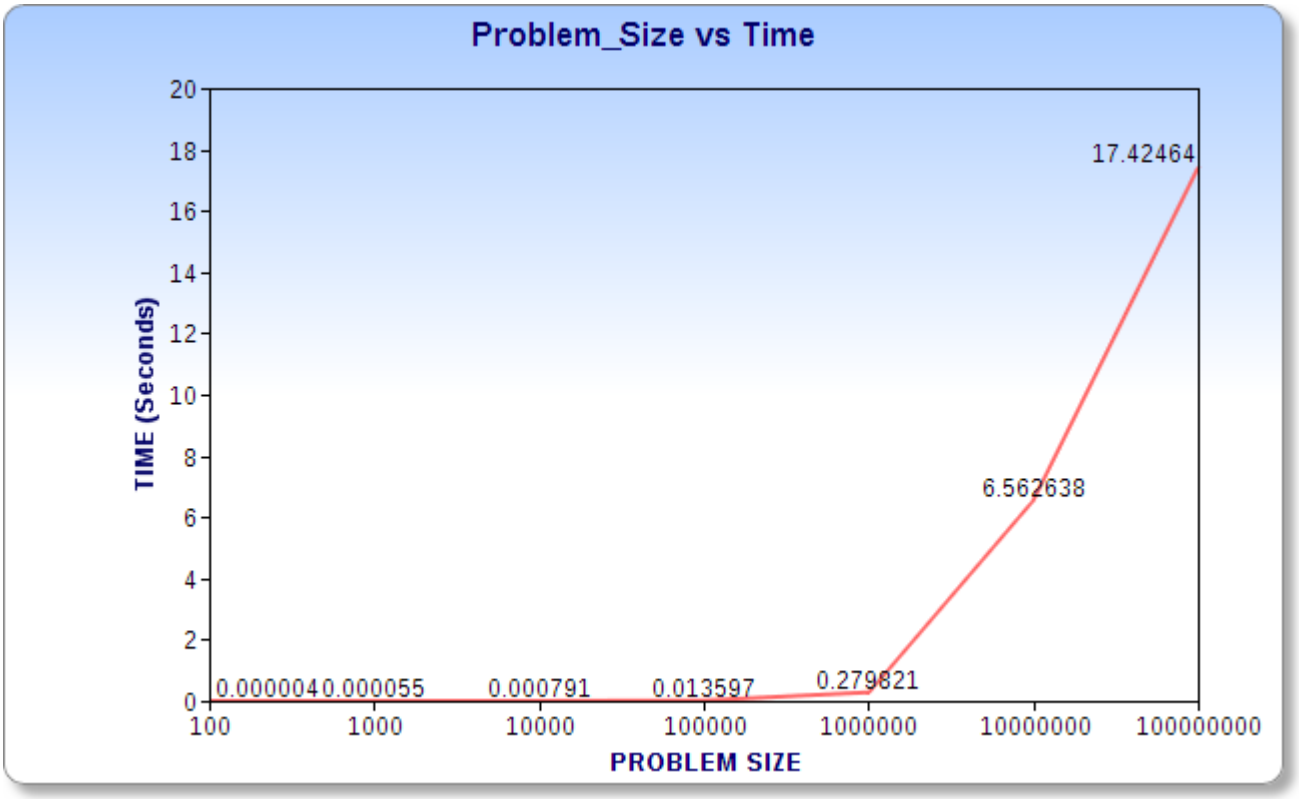
In the above graph we can see that the time increases as we increase the Problem Size.

The graph below shows that as we increase the number of cores the tasks are divided and the time taken to complete the task is less than that of using a single node.



SSE

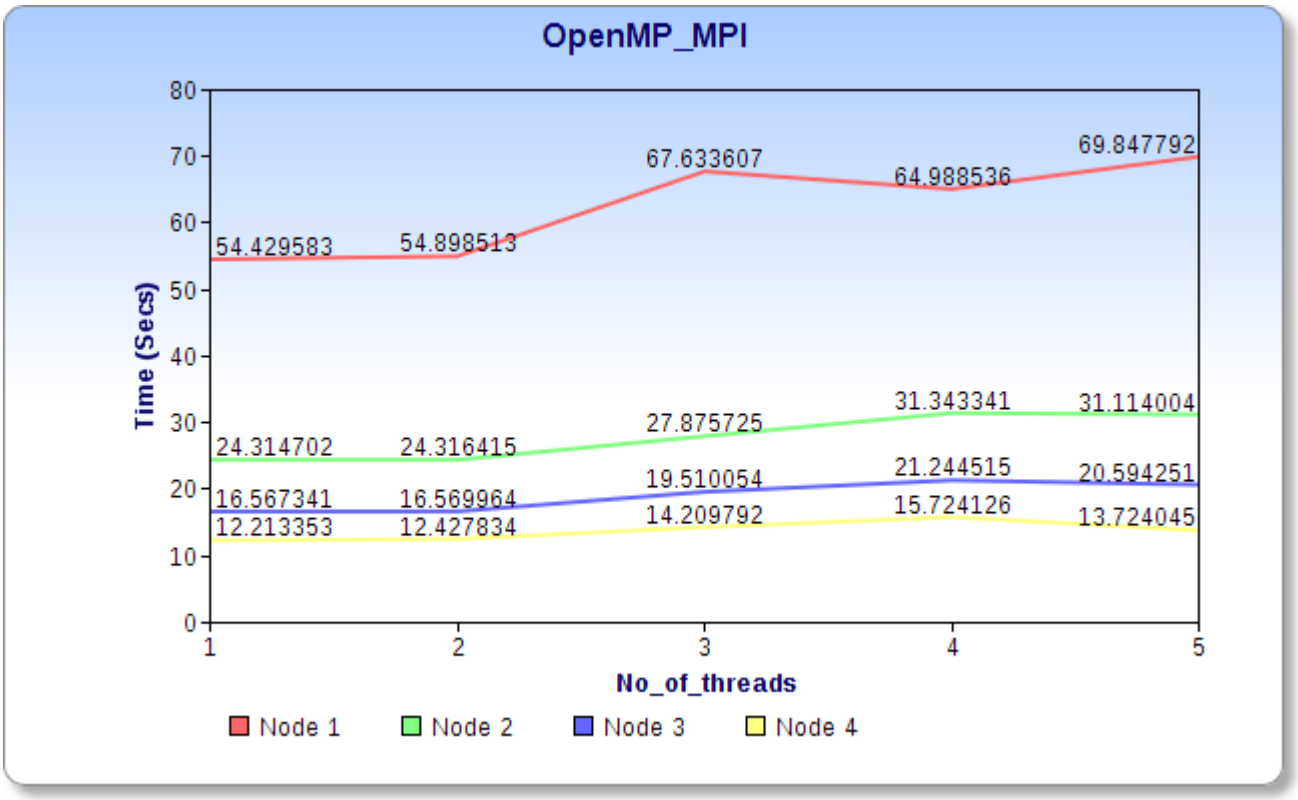
Filename: PDP1SSE.c
Makefile: make_sse
Command to run the makefile: make -f make_sse
Command to run the object file: ./sse [PROBLEM_SIZE]
Sample: ./sse 1000
SLURM FILE: SLURM_SSE.sh



In SSE we use three 128 bit registers X,Y Z to calculate the value. The logic used is to put a number in the X register 4 times and the for this number we put the corresponding values in Y to divide with. Now when carry out division,flooring,multiplication and subtraction operation and thus we have the remainder. Checking the remainder we know if the number is prime.

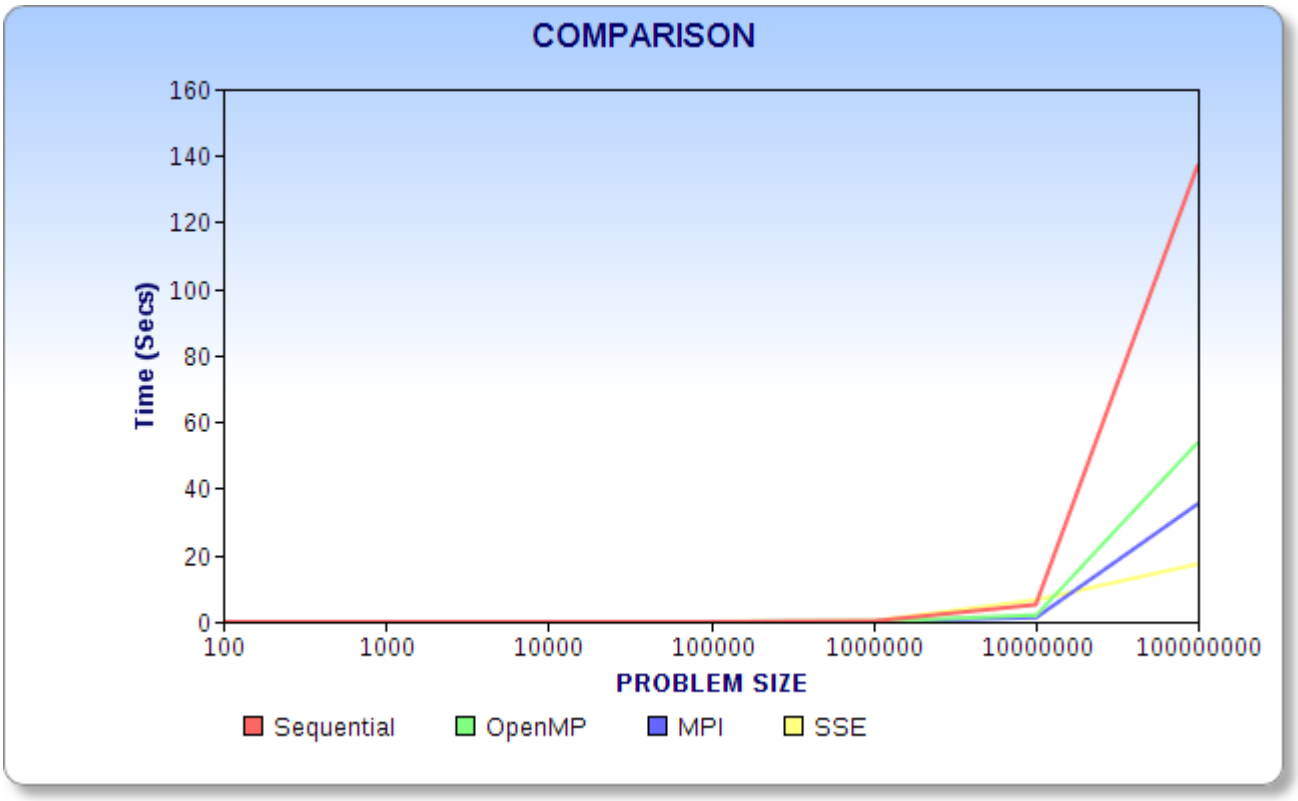
OpenMP-MPI

Filename: PDP_OPENMP+MPI.c
Makefile: make_openmp+mp1
Command to run the makefile: make -f make_openmp+mp1
Command to run the object file: ./plus [NO_OF_THREADS] [PROBLEM_SIZE]
Sample: ./plus 3 1000
SLURM FILE: SLURM_SSE.sh (Edit NO_OF_NODES in this file)



In openmp-mpi we take the part which is executed by each node and divide that part into various threads. The following graph shows the comparison between various nodes and the threads with respect to time.

COMPARISON BETWEEN SEQUENTIAL, OPEN MP, MPI and SSE



From the graph above we can say that for the higher powers of 10 we see a considerable decrease in time than the amount of time taken in a sequential program thereby saving time.