ASSIGNMENT-2 REPORT

U.S.M.M Teja
CS21BTECH11059

Graph-1:

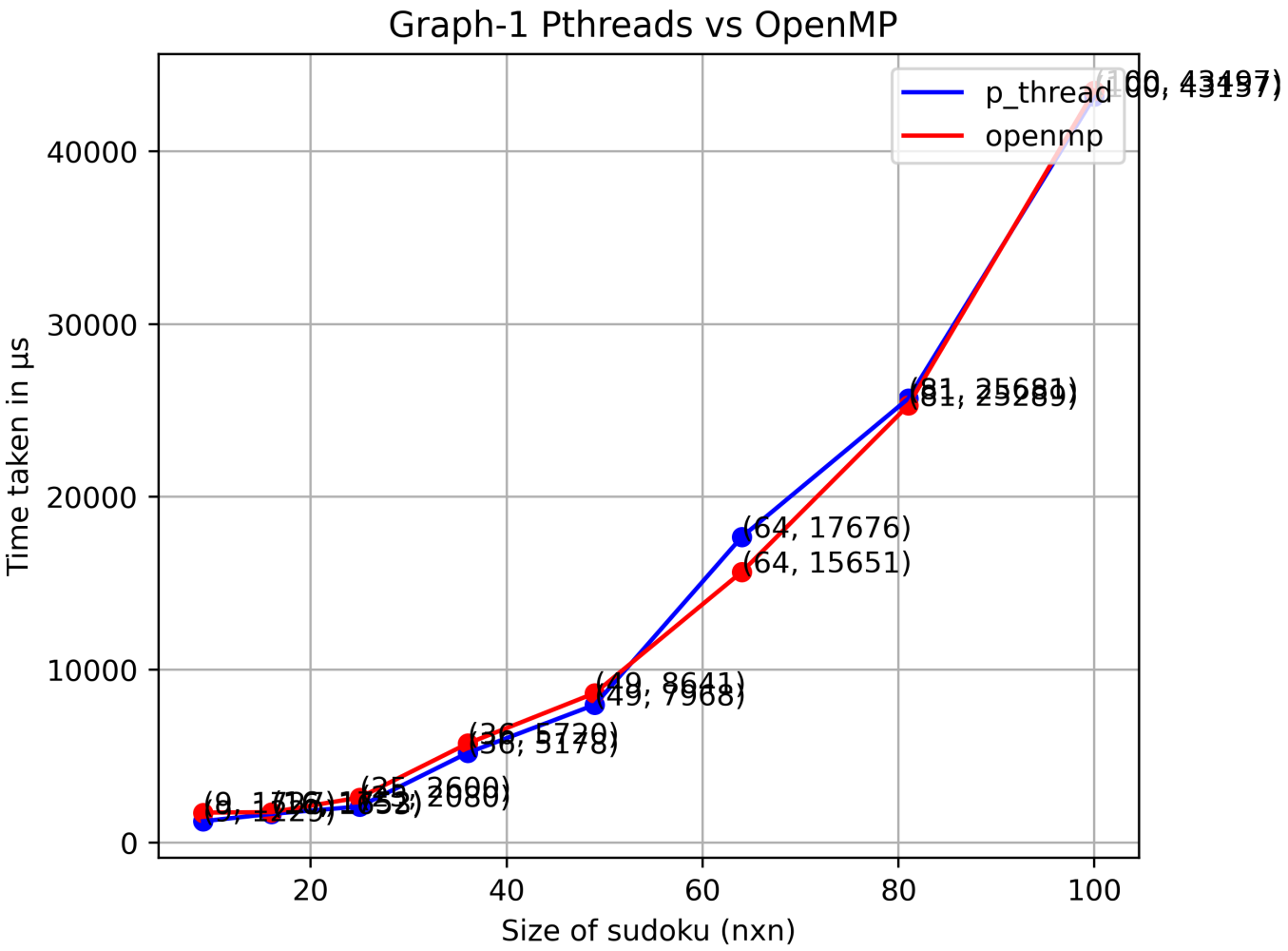

Figure 1: graph-1

Graph-2:

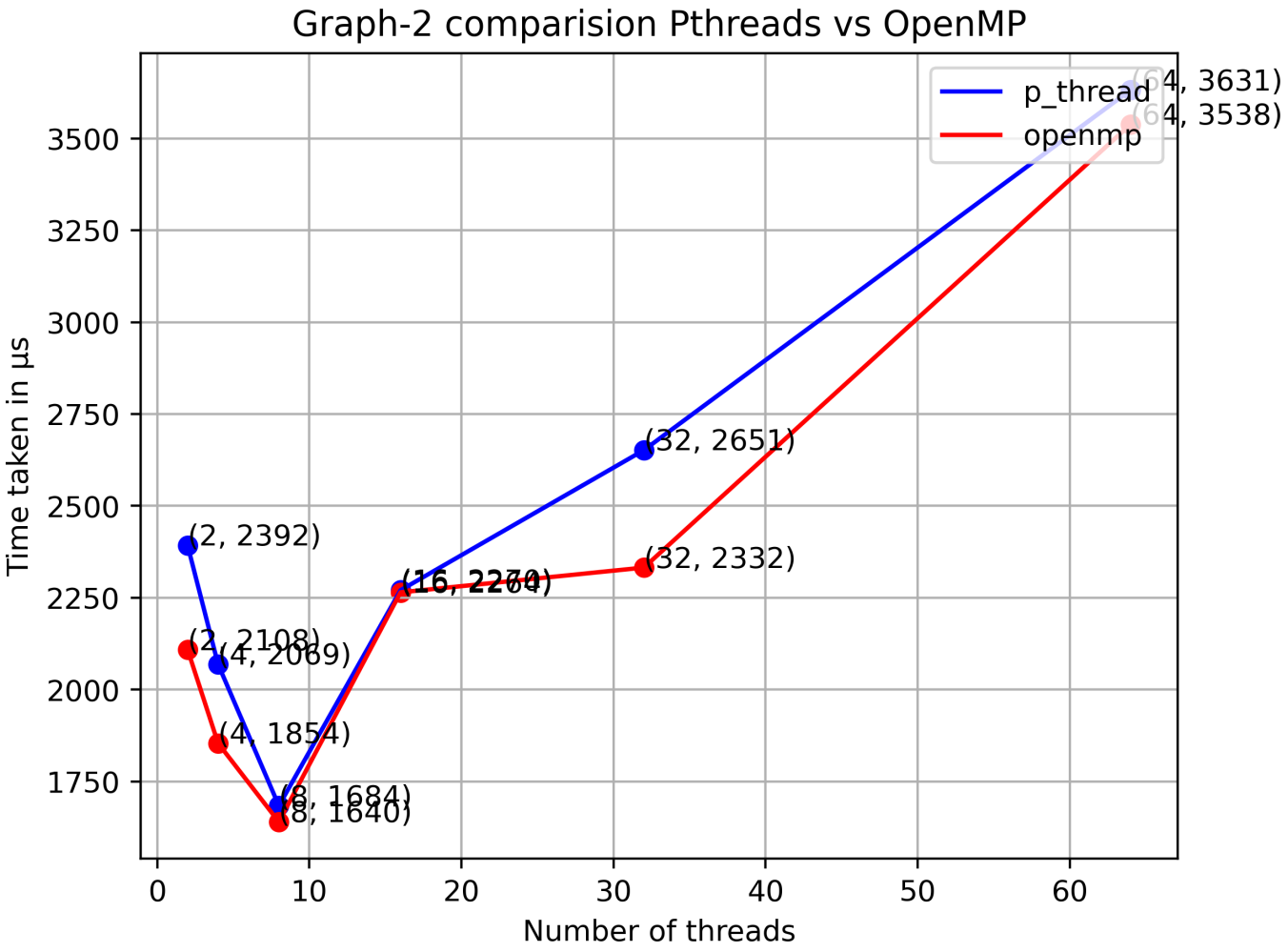

Figure 2: graph-2

Design:

In the source code Assign2SrcpthreadCS21BTECH11059.c, I have included several libraries: <stdlib> library: is used to allocate memory for int *Row, int *Colum, int *Grid, struct Node *node, int **arr(for matrix).

<math.h> library is used for calculating square root of n($n \times n$ matrix)

<pthread> library is used for assigning threads .

<sys/time.h> library is used for calculating time of execution from threads assigning to the task to threads getting joined.

<vector> is used in GridIterator, ColumnIterator, RowIterator functions for storing the numbers in their respective grids, columns, rows. After that they get sorted using sort function.

<iostream> should definitely get included.

In the source code I have declared struct Node, n, k, arr, Row, Colum, Grid globally so that every thread can access its contents. RowIterator function evaluates every row of the given sudoku grid by assigning every row elements into a vector. After that we sort the vector and we check wheather consecutive elements are equal or not. If they are equal we assign the Row[Rowindex] as 0 i.e it doesnot follow sudoku condition.Else it assigns 1. same goes with ColumnIterator, it evaluates every column of the given sudoku grid by assigning every column elements into a vector. After that we sort the

vector and we check wheather consecutive elements are equal or not. If they are equal we assign the Colum[columnindex] as 0 i.e it doesnot follow sudoku condition.Else it assigns 1. Same goes with grid iterator i.e it evaluates every subgrid of the given sudoku grid by assigning every subgrid elements into a vector. After that we sort the vector and we check wheather consecutive elements are equal or not. If they are equal we assign the Colum[columnindex] as 0 i.e it doesnot follow sudoku condition.Else it assigns 1.

void *task function is thread function where it assigns the threads to the functions i.e it takes the parameter var from the main function for loop (i = 0 or 1 or 2 or 3) and it calculates start value and end value as i * ((3 * n) / k) and (i + 1) * ((3 * n) / k) respectively. After that we run a for loop from start to end and if the iterator value is less than the n value it goes to RowIterator function and it evaluates the row. And if the iterator value is greater than the n value and less than the 2 × it goes to ColumnIterator function and it evaluates the Column corrosponding to that index.Else it goes to the GridIterator function and evaluates the grid. So technically we are distributing the 3 × n tasks into k threads almost equally. Specially for the GridIterator function we pass the struct to pass the coordinates and the corrosponding grid index value.

Finally coming to the main function we open the input.txt file and we scan the file to get k and n and then the sudoku grid. We assign the sudoku grid into an array. Then we assign the memory for each of the parameters such as arr, node etc. Then we create threads as per required input and after the threads run we join them. The gettimeofday function calculates the time of execution before the threads creation until the thread join. Finally we write the output into the file Outmain.txt file regarding thread checking the corrosponding row or column or the grid.

Same as for the sourse code Assign2SrcOpemMpCS21BTECH11059.cpp except that we have a pragma function in which we pass the k parameter i.e number of threads and we remove the void * and put it as void.Even we remove the * in the function arguement.Rest all the functions are same as in Assign2SrcpthredCS21BTECH11059.cpp.

Analysis of the source codes:
Time complexity of the algorithm is $O(\frac{3\times n^2\times logn}{k})$. This is because nlogn for sorting n numbers and scheduling 3× n group of numbers into k threads which run in parallel.

Anamolies:
In the graph-2 and graph-1 the graph for both pthread and OpenMp is always increasing .That is obvious because as the size of the grid increases keeping the thread number constant we get increased time computation. But as we observe for the higher number of threads OpenMp works better than pthreads.

The performance of OpenMp and pthread libraries depends on various factors such as input size, number of threads, system architecture etc This is because if the workload involves tasks with very short execution times, pthreads might perform better due to lower overhead from creating and synchronizing threads. OpenMP works faster for larger inputs as it has better load balancing, efficient use of shared memory, its optimized libraries. For larger inputs OpenMp performs better than pthread as OpenMp is designed for shared memory parallel programming.It also provides a higher level of abstraction for programming parallel applications than pthreads. It is also easier to use in diving the work to the threads. As it has high level of optimization for load balancing and thread coordination. It give better results than pthreads. So definitely for the larger inputs OpenMP works better.

But in case of pthreads it provides low level of abstraction and it more control over threads.This allows for more fine-grained control of thread behavior and can give better performance for small inputs where overhead from thread coordination is a larger fraction of the total execution time. But in case of larger inputs, the overhead from thread coordination can become a smaller fraction of the total execution

time.

And as we observe the graph-2 closely We might guess that the graph first gets decreased and then increasingly, but here are the reasons why it does gets always correct. At first graph may be increasing because of several issues like the overhead associated with creating and managing threads. But after that as we more threads we get better performance due to better performance and better utilization of threads. And as we significantly increase the number of threads, the exection time gets increased because of increased overhead and more time for context switching. Only initially we may see the problem i.e graph gets increased but the reason is clear as above.