HPC LAB-3

V Sai Srikar SE20UARI164

B.Krishna Vamshi SE20UARI179

Aashish Joshua James SE20UARI002

Aditya Sake SE20UARI006

ADITYA SAI SE20UARI005

To Run the programs-:

Go to Lab3 folder->

and go Threshold Folder first run Matrix\_Gen.c file

Run the following command to Generate Matrix and store it in a File-:

1. gcc Matrix\_Gen.c -o matrixgen -fopenmp
2. ./matrixgen <number of rows> <number of columns> <file name.txt>

Run Threshold.c to get Binary image of the given generated matrix-:

1. gcc Threshold.c -o thresh -fopenmp
2. ./thresh <file name.txt> <Percentage(10)> <No of threads>

Parallel code for-:

**1.Threshold-:**

The count distribution of matrix is a technique that can be used to analyse the pixel values in an image. By analysing the count of pixel values in the image, we find the threshold value.

Once the threshold is set, we can traverse the image matrix and compare each pixel value to the threshold. Pixels with values above the threshold are set to 1 (white), while pixels with values below the threshold are set to 0 (black).

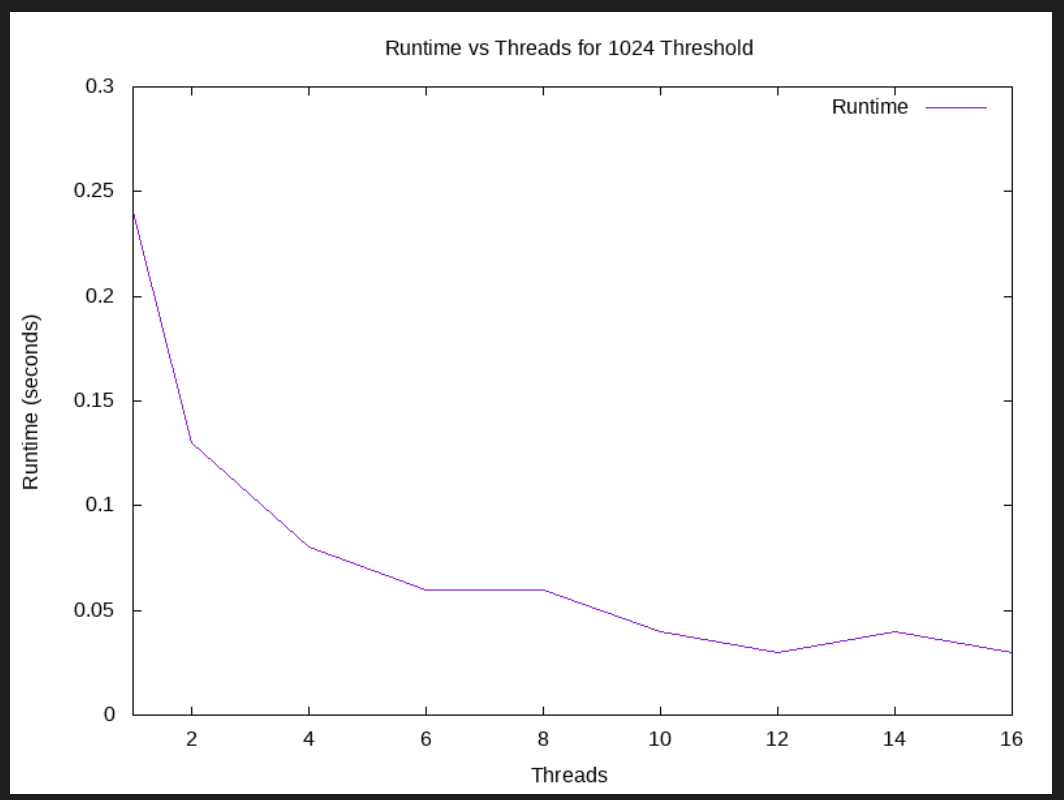
The given code has been parallelized using OpenMP to improve its performance on multi-core processors. The count\_distribution function, which performs a nested loop to count the pixel values in the image, has been parallelized using the "collapse" clause to parallelize the outermost loop over multiple threads.

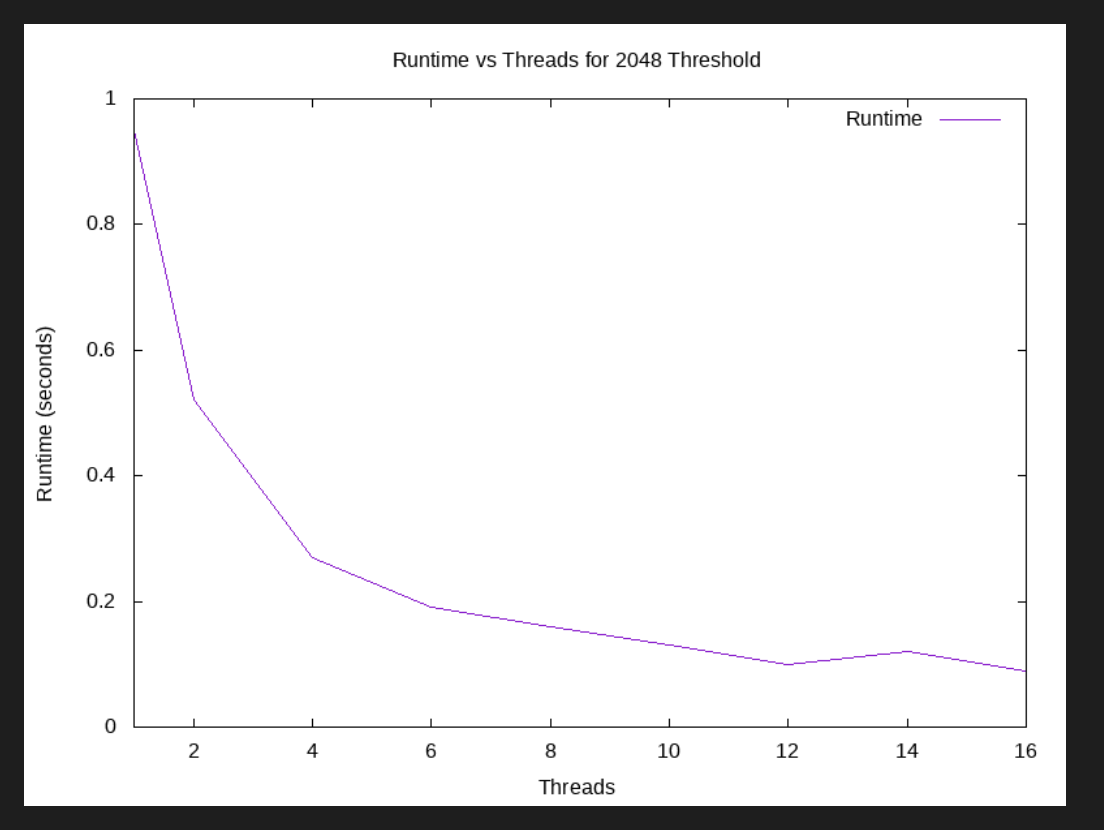
Similarly, the threshold function, which sets the binary threshold for the image, has also been parallelized using OpenMP by using the "collapse" clause to parallelize the outermost loop over multiple threads.

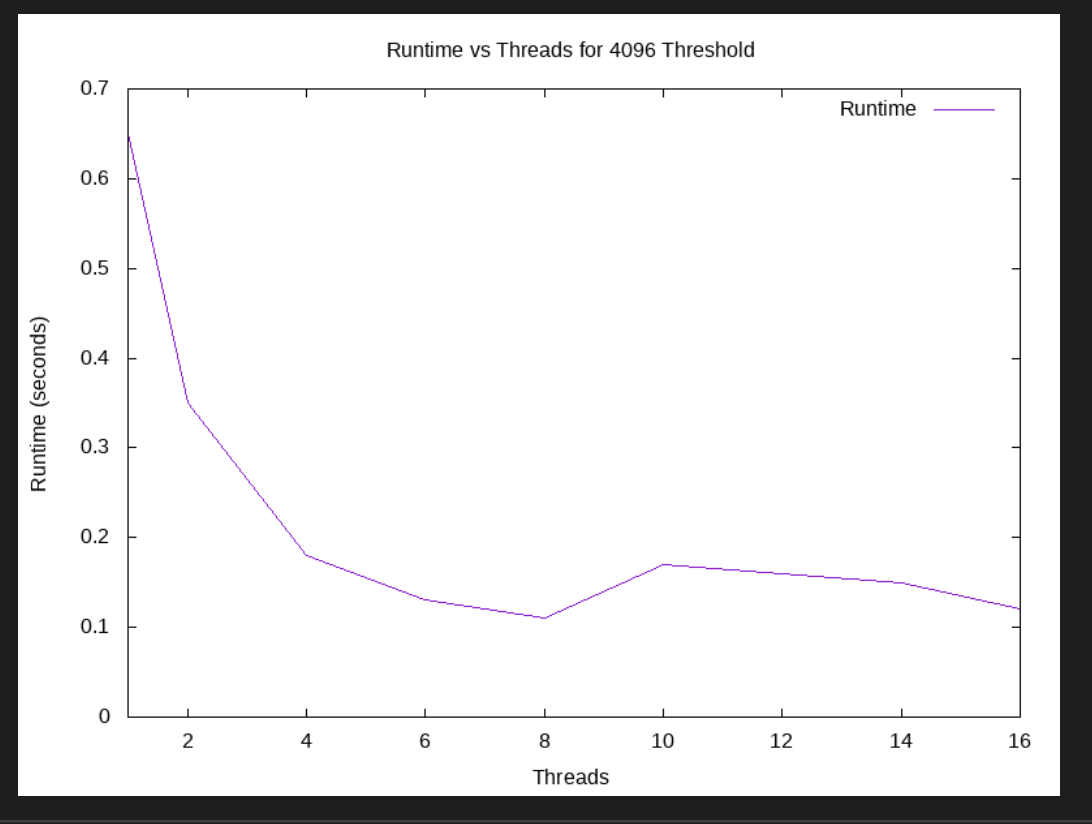
Moreover, the number of threads can be set using the omp\_set\_num\_threads function to control the number of threads that will be used to execute the parallel sections.

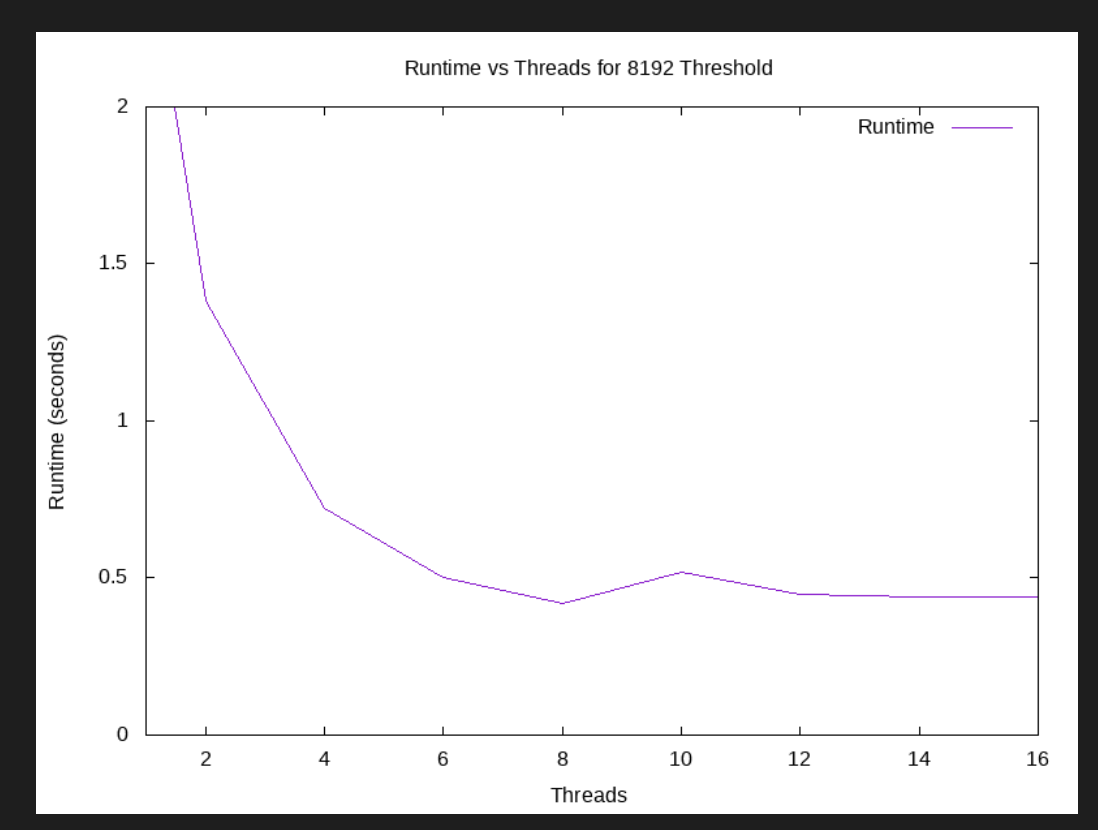
Below you can see the GNUPlots we made for various matrix sizes starting from 1024 to 8192 and you can see that as the number of threads increase the amount of parallelism increases thereby reducing the run time.

We think that the small spikes after we reach saturation is because as the number of threads increase even though we have reach the saturation for parallelism creation of threads take some time.









To Run the programs-:

Go to Lab3 folder->

and go ChainMatrixMultiplication Run [ChainMatrixMultiplication.c](https://github.com/Baka-14/HPC-Labs/blob/main/Lab-3/ChainMatrixMultiplication/ChainMatrixMultiplication.c)

To run-:

1.gcc ChainMatrixMultiplication.c -o cmm -fopenmp

2. ./cmm <number of threads>

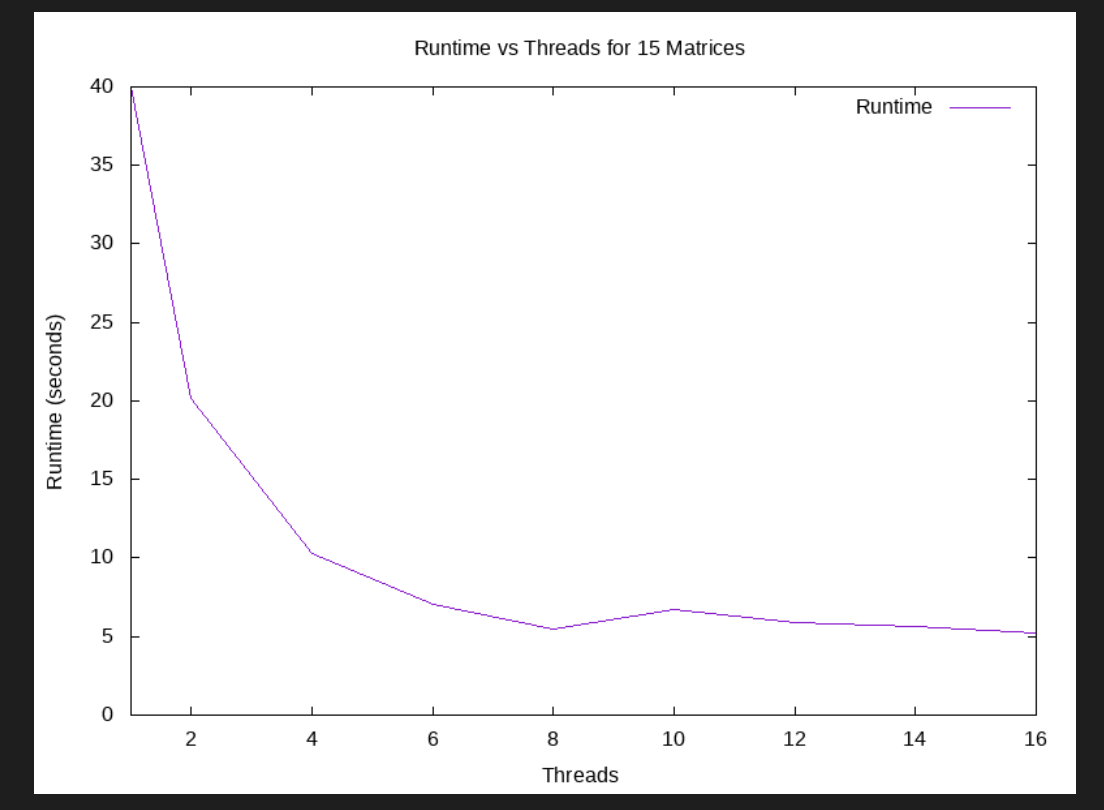
**2.Chain Matrix Multiplication-:**

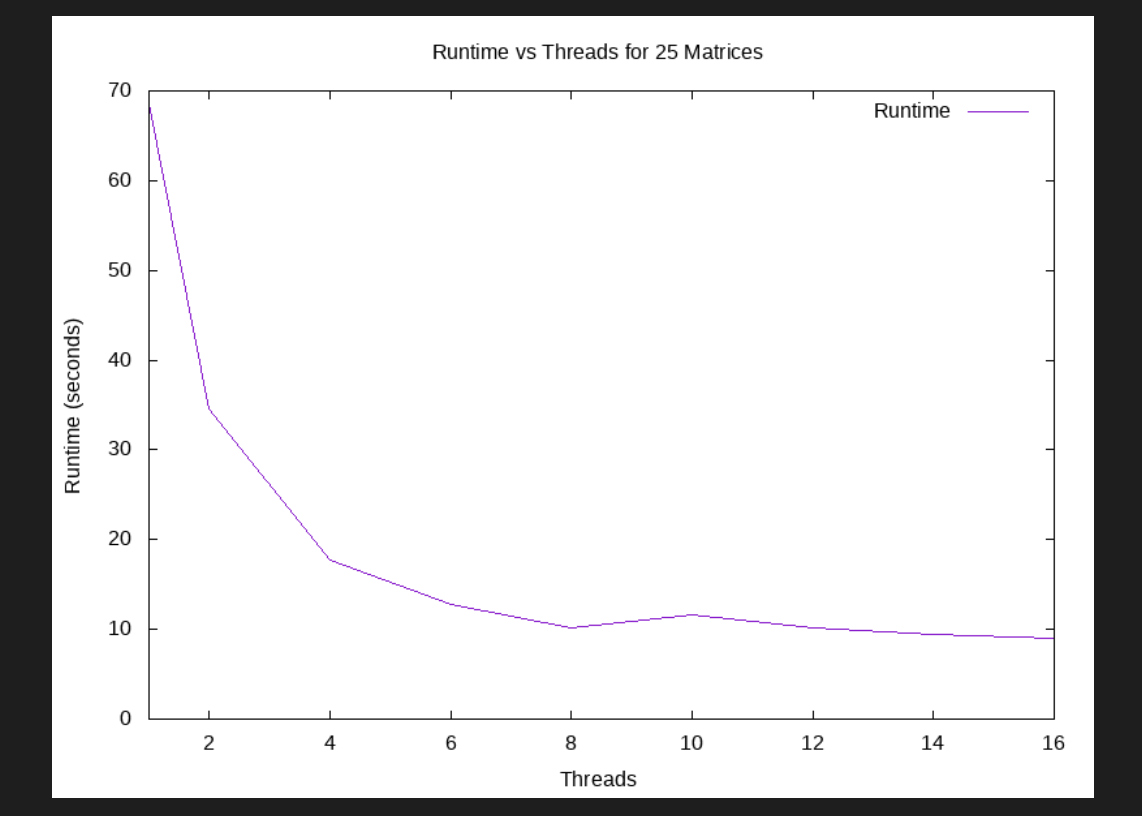
Chain matrix multiplication is the process of multiplying several matrices together in a specific order.

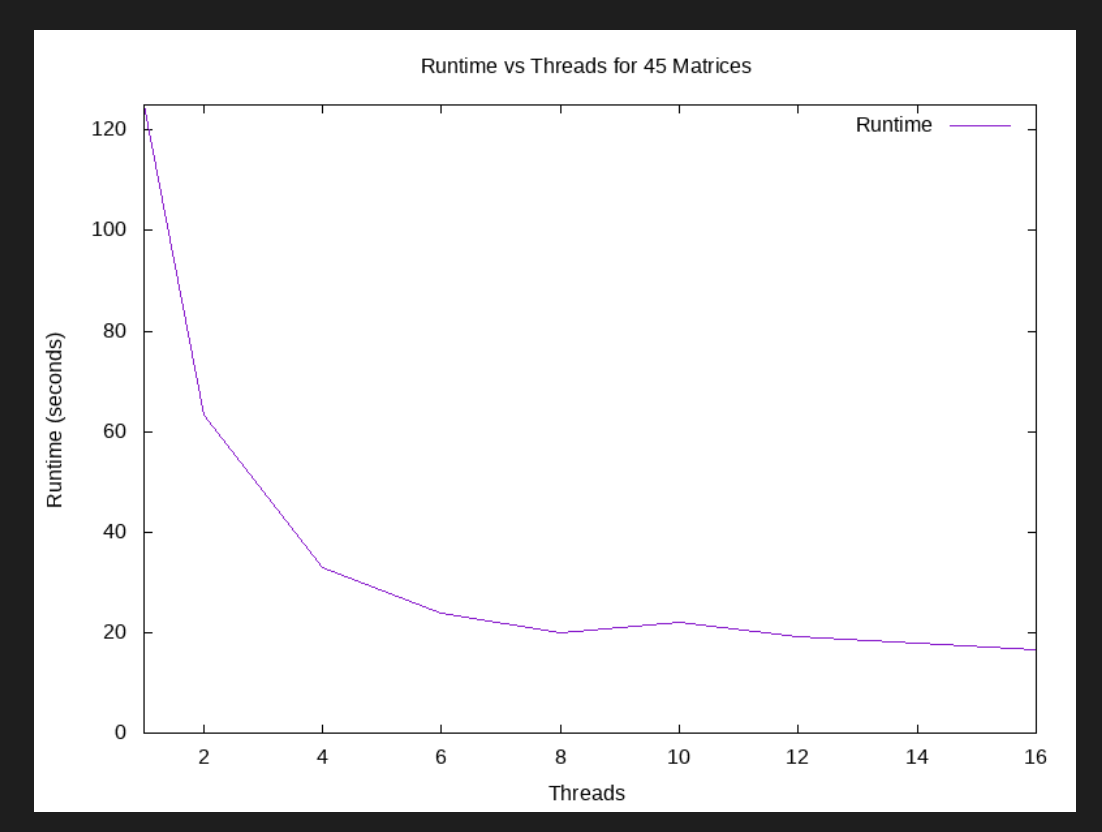
The code performs chain matrix multiplication using OpenMP, a parallel programming library for shared-memory architectures. The program takes the number of threads as an argument and creates an array of matrices. It then multiplies the matrices in a chain using the multiply() function, which implements parallelism with the OpenMP pragma directive.

The multiplication is done by iterating over the rows and columns of the result matrix, with a nested loop to calculate the dot product of each element.

Below we can see the GNUplots for different values of n=15,25,45 and we observe that as n increases the time increases and as the number of threads increase the amount parallelism increases thereby reducing the runtime.







Go to Lab3 folder->

and go to Game of Life folder-:

First Run the following command to Generate seed Matrix and store it in File-:

1. gcc SeedGenerationc -o seed -fopenmp
2. ./seed <number of rows> <number of columns> <file name.txt>

Run GameOflife.c to get board after 10 generations-:

1. gcc GameOflife.c -o gof -fopenmp
2. ./gof <file name.txt> <No of threads>

**3.Game of Life-:**

The Game of Life is a cellular automaton game devised by British mathematician John Horton Conway in 1970. It consists of a two-dimensional grid of cells, where each cell can either be alive or dead, and evolves according to a set of simple rules based on the state of its eight neighbours.

The given code runs the Game of Life simulation for a specified number of generations. It reads in an initial grid configuration from a file, updates the grid for the specified number of generations, and prints out the updated grid after each generation.

To parallelize the simulation using OpenMP, the code uses the "#pragma omp parallel for" directive to distribute the work of initializing and updating the grid across multiple threads. Specifically, the "seed" and "copy\_arr" functions use this directive to parallelize the initialization and copying of the grid, respectively, while the "update" function uses the "collapse(2)" clause to parallelize the nested loop that updates each cell's state

Below you can see the GNU Plot for a board of size of 10,00 X 10,00 and you can see that as the number of threads increase the runtime decreases

