CS546 Parallel and distributed Processing Programming Assignment
Design

In this assignment, I implemented two parallel algorithms: Threads and MPI for solving a dense linear equations of the form A*x=b, where A is an n*n matrix and b is a vector. I've used Gaussian elimination without pivoting. I've parallelized both the algorithms and optimized the code which runs as fast as possible.

Basically, the serial algorithms has two parts:

1.Gaussian Elimination: The original system of equation is reduced to an upper triangular form Ux=y, where U is a matrix of size n*n in which all elements below the diagonal are zeros which are the modified values of the A matrix, and the diagonal elements have the value 1. The vector y is the modified version of the b vector when you do the updates on the matrix and the vector in the Gaussian elimination stage.

2. Back Substitution : The new system of equations is solved to obtain the values of x. The Gaussian elimination stage of the algorithm comprises (n¬-1) steps. In the algorithm, the ith step eliminates nonzero sub diagonal elements in column i by subtracting the ith row from row j in the range [i+1,n], in each case scaling the ith row by the factor Aji/ Aii.so as to make the element Aji zero. Hence the algorithm sweeps down the matrix from the top left corner to the bottom right corner, leaving sub diagonal elements behind it.

Now, We will discuss about two parallel Algorithms:

PThreads:

Initially, we should accept 3 command line parameters.

1. Read Matrix size,

2. Random Seed

3. Output file.

And initialize inputs of A,B and X ,and Thread variables .and create the number of workers(threads) which includes main thread. Now, call gauss() function to parallelize the algorithm using PThreads. Immediately after creation of threads, all workers start waiting on the sync condition variable. This allows them to wakeup if there is work available.

now, we have to do gauss elimination using worker barrier since work gets executed from here by using two global variables and sending row reference to start work for dowork(). we have sync lock which is used for barrier and row lock used for scheduling. And worker barrier take care of synchronization .worker is going to Start processing only when all the children and main thread are ready. They will work in parallel using dynamic scheduling. And we are taking chunck size 4 to perform do work on Matrix size Workers performs do work() till all the rows are to be done. And we join and synchronize workers to do work. Work gets done on one instance where worker barrier synchronize with other threads finally, we will do back substitution to get the results. At last, we will join all the workers together before they terminated.

In Pthreads, I have tried different approaches to write the code. Firstly, I did the gauss elimination based on row lock without dividing the workload for particular thread. But, There are many computation performed by each thread, resulting more elapsed time. Then, I used chunk to dedicated few number of rows (based on chunk size) for each thread in each iteration. Here, I varied chunk sizes from 1,,2,3,4. Finally, I used chunk size 4, results good Performance.

MPI:

Initially, we should accept 3 command line parameters.

1. Read Matrix size,

2. Random Seed

3. Output file.

And number of processors.

Allocate memory for A,X and B. initialize inputs of A,B and X ,and Mpi initialization. And gauss elimination and back substitution is to be done in Gauss() function.

In Gauss elimination, we are going to do A*X=B for every iteration. Before that we need to broadcast the row for normalization in the iteration of A and also values in B. and divide the workload among all the processes. Each process will do operations on few rows based on workload without dependency. Here, we are assigning number of rows and row size for each processor. we have to scatter the data between the processes so that processes knows how many rows to work. Each process will start performing sequential algorithm on workload assigned to particular process. Process 0 collects the data using Recv, while the other processes send the data using send. The same procedure follows for each iteration. we will do back substitution to get the results, and store the results in output file.

In Mpi,I have been trying to optimize the performance. But, it is taking more time to send and receive data among the processes(Distribution).