**Riten Chhatrala Assignment-2**

[**rchhatrala@hawk.iit.edu**](mailto:rchhatrala@hawk.iit.edu)

**CWID:-A20385102**

I have uploaded all my code file in Jarvis under home/rchhatrala/hw2 directory.

**Dependency In Gaussian Elimination Serial Algorithm For loop is as below:**

* The values in the outer most row and column are used to update the inner sub matrix before going on to next row/column, meaning that outer most loop has data dependency. Therefore we can't use this loop because of data dependence to parallelize.
* Second loop does not have any dependency in any iteration so can parallelize this loop.
* Third inner most loop also does not have any dependency so can also parallelize this loop.
* **Serial Gaussian Elimination Implementation:-**

**File Name:**- **serialGaussianAlgorithm.c**

In serial algorithm has three loop. After implementing Gaussian Elimination with three loop we will get upper triangular matrix with all element under the diagonal of NxN matrix will be zero.

When we get upper triangular matrix we will pass that matrix into **backSubstitution** function to calculate the answer.

* **Code Explanation**:-
  + **struct Thread\_Data** :- Used to make data global for other function use.
  + **GenerateMatrix** :- This function will generate a NxN matrix with random values.
  + **PrintMarix**:- By calling this function we will prinf the matrix. But we are currently not using this function as matrix is too large to print if we pass order of matrix large value.
  + **BackSubstitution**:- This function use to calculate answer of X vector after matrix will transform into upper triangular form.
  + **SerialAlgorith**:- Using this function I am implementing gaussian Elimination on NxN matrix.This function has three loop and need to parallelize this code.
  + **Main()**:- Execution start from this function. Dynamically allocate memory and generate NxN matrix and Nx1 vector and than call above function and calculate time for running Gaussian Elimination,Back Substitution and Total time of algorithm.
* **Running instruction:-**

**1. vi serialGaussianAlgorithm.c**

* + - copy **serialGaussianAlgorithm.c** file from provide code and past it in terminal.
    - Save it by **:wq**

**2. gcc -o serialGaussianAlgorithm serialGaussianAlgorithm.c**

* compilethe code by above command. This command will generate object file **serialGaussianAlgorithm**

**3. ./serialGaussianAlgorithm**

* run program by above command.It will ask for Order of matrix and number of thread.

**Output:**

program will output running time for Gaussian Elimination ,Back substitution time and Total running time for algorithm. It will also display solution matrix.

* **Pthread** **Gaussian Elimination Implementation:-**

File Name:- **pThreadGaussianAlgorithm.c(Final version).**

Parallelize the serial algorithm we need to identify that which loops we can parallelize. So when we unroll the loops we can identify that the first loop(outer most loop) has data dependency as values calculated in above row and column will be used in below sub matrix. So can not make above loop parallel. Now for inner two loop there is no dependency so we can parallel any loop. But for better performance and best implementation I have parallelize middle loop out of three loop. As when we parallel middle loop less number of fork and join perform so communication and synchronization cost will be less.

* **Code Explanation:-**
  + **struct Thread\_Data**
  + **GenerateMatrix**
  + **PrintMarix**
  + **BackSubstitution**

Above functions are same as serial algorithm.

* + **Main():** This function is same as serial but for pthread we are creating threads and as it has data dependency in outer most loop we are joining threads to wait for every thread to finish that row operation. This function will calculate the running time for pthread Gaussian Elimination, back substitution and Total time for Pthread implementation.
  + **Pfunction() :-** This is parallel thread function. Each Thread will execute this function and perform the task for there data part in parallel.
* **Running instruction:-**

**1. vi pThreadGaussianAlgorithm.c**

* + - copy **pThreadGaussianAlgorithm.c** file from provide code and past it in terminal.
    - Save it by **:wq**

**2. gcc -o pThreadGaussianAlgorithm pThreadGaussianAlgorithm.c -lpthread**

* compilethe code by above command. This command will generate object file  **pThreadGaussianAlgorithm.** We need to link the pthread library in compiling.

**3. ./pThreadGaussianAlgorithm**

* run program by above command. It will ask for Order of matrix and number of thread.

**Output:**

program will output running time for Gaussian Elimination ,Back substitution time and Total running time for algorithm. It will also display solution matrix.

* **OpenMP Gaussian Elimination Implementation**:

**File Name:- openMPGaussianAlgorithm.c**

OpenMp is directive base parallel programming in which we need to identify which loop to parallelize and then define directive above the for loop in defined directive parallel region.

* **Code Explanation:-**
  + **struct Thread\_Data**
  + **GenerateMatrix**
  + **PrintMarix**
  + **BackSubstitution**

Above functions are same as serial algorithm.

* + **Main():** This function is same as serial but for openMP we are calling openMPAlgorith function for parallel OpenMP implementation. This function will also calculate the running time for OpenMP Gaussian Elimination, back substitution and Total time for OpenMP implementation.
  + **openMPAlgorith()** :- In this function we are defining parallel region by below line

#pragma omp parallel num\_threads(t\_info.numThreads) shared(n,A,j) private(i,c,k) and

and by defining below line we can parallel for loop how to parallel this for loop will be handled by compiler.

#pragma omp for schedule(static)

* **Running instruction:-**

**1. vi openMPGaussianAlgorithm.c**

* + - copy **openMPGaussianAlgorithm.c** file from provide code and past it in terminal.
    - Save it by **:wq**

**2. gcc -o openMPGaussianAlgorithm openMPGaussianAlgorithm.c -lgomp -fopenmp**

* compilethe code by above command. This command will generate object file  **openMPGaussianAlgorithm.** We need to link the openMp library **gomp** in compiling.

**3. ./openMPGaussianAlgorithm**

* run program by above command. It will ask for Order of matrix and number of thread.

**Output:**

program will output running time for Gaussian Elimination ,Back substitution time and Total running time for algorithm. It will also display solution matrix.

* **Other Algorithm Version Implementation:**

1. **masterAllAlgrith:**

**File Name:- masterAllAlgrith.c**

In this I had implement all the algorithm together and display the output for all algorithm time and solution vector for all algorithm.

Algorithm Implemented in this program:

1)Serial Algorithm

2)pthread with middel loop parallelism implement Main and final version of pthread.

3)OpenMP Algorithm

4)Pthread version 2 third loop parallelism(inner most loop) version 2 implementation worst performance as communication and synchronization cost is very high.

5)Pthread implementation with partial pivot this program will ensure that it will not go to infinity.

* + - **Running instruction:-**

**1. vi masterAllAlgrith.c**

* + - * + copy **masterAllAlgrith.c** file from provide code and past it in terminal.
        + Save it by **:wq**

**2. gcc -o masterAllAlgrith masterAllAlgrith.c -lpthread -lgomp -fopenmp**

* compilethe code by above command. This command will generate object file  **masterAllAlgrith.** We need to link the pthread(**-lpthread**) and openMp(**-lgomp**) library in compiling.

**3. ./masterAllAlgrith**

* run program by above command. It will ask for Order of matrix and number of thread.

**Output**:

program will output running time for Gaussian Elimination ,Back substitution time and Total running time for all the algorithms. It will also display solution vectors of all algorithms.

1. **pThreadPartialPivot:**

**File Name:-pThreadPartialPivot.c**

Pthread implementation with partial pivoting that will insure that it will not go to infinity.

* + - **Running instruction:-**

**1. vi pThreadPartialPivot.c**

* + - * + copy **pThreadPartialPivot.c** file from provide code and past it in terminal.
        + Save it by **:wq**

**2. gcc -o pThreadPartialPivot pThreadPartialPivot.c -lpthread**

* compilethe code by above command. This command will generate object file  **pThreadPartialPivot.** We need to link the pthread(**-lpthread**) and openMp(**-lgomp**) library in compiling.

**3. ./ pThreadPartialPivot**

* run program by above command. It will ask for Order of matrix and number of thread.

**Output**:

program will output running time for Gaussian Elimination ,Back substitution time and Total running time for algorithm. It will also display solution matrix.

3. **pThread Gaussian Implementation Version2**:

**File Name:- pThreadGaussianVersion2.c**

In this pthread version I parallelize the inner most loop that third loop. So in this version more number of thread will be created. For more thread communication cost and synchronization cost increases. So therefor this version will give worst performance. According to best practice for parallel algorithm implementation we always have to parallelize outer loop first. But for understanding purpose and to see the performance difference I implemented this version.

* + - **Running instruction:-**

1. **vi pThreadGaussianVersion2.c**

* + - * + copy pThreadGaussianVersion2.c file from provide code and past it in terminal.
        + Save it by :wq

2. **gcc -o pThreadGaussianVersion2 pThreadGaussianVersion2.c -lpthread**

* compile the code by above command. This command will generate object file pThreadGaussianVersion2. We need to link the pthread(-lpthread) and openMp(-lgomp) library in compiling.

3. **./ pThreadGaussianVersion2**

* run program by above command. It will ask for Order of matrix and number of thread.

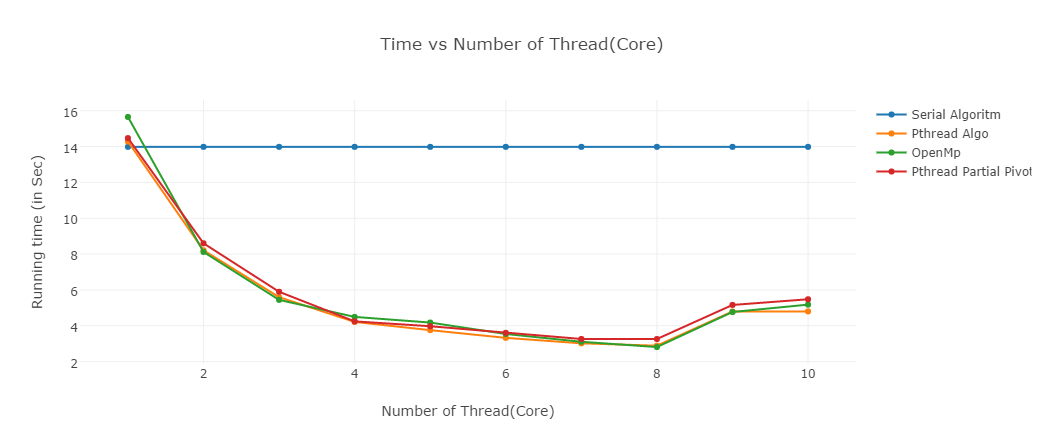
**Output:** program will output running time for Gaussian Elimination ,Back substitution time and Total running time for algorithm. It will also display solution matrix.

* **Running Time For different Algorithm with different number of threads:**

**Matix Size=2000x2000**

**On jarvis Machine:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of Threads(Core)** | **Serial Algorithm(Sec)** | **Pthread Algorithm(Sec)** | **OpenMP algorithm(Sec)** | **Pthread Version2(Sec)** | **Pthread Partial pivoting(Sec)** |
| 1 | 13.983 | 14.282 | 15.652 | 134.927 | 14.471 |
| 2 | 13.983 | 8.213 | 8.119 | 146.100 | 8.601 |
| 3 | 13.983 | 5.596 | 5.445 | 169.306 | 5.897 |
| 4 | 13.983 | 4.214 | 4.494 | NA | 4.245 |
| 5 | 13.983 | 3.755 | 4.179 | NA | 3.974 |
| 6 | 13.983 | 3.325 | 3.542 | NA | 3.604 |
| 7 | 13.983 | 3.020 | 3.106 | NA | 3.262 |
| 8 | 13.983 | 2.889 | 2.812 | NA | 3.260 |
| 9 | 13.983 | 4.796 | 4.768 | NA | 5.161 |
| 10 | 13.983 | 4.801 | 5.179 | NA | 5.478 |



**Analysis:**

From above table we can see that running time for Pthread, OpenMp and Pthread with partial pivoting is decreased with increase in number of threads.But from number of thread 9 and 10 observation we can see that running time increased because when number of thread > no of core in cup then threads will required contact switch in CPU. So cost of contact switch will have increased. So for number of thread 9 and 10 running time in pthread and OpneMp increases.

In Pthread Version2 algorithm I parallelize third inner most loop. So in this scenario it will create lot number of thread so communication cost, contact switch cost and synchronization cost will increases. So for that algorithm we are getting worse result that we can see from above table.