

CS 434: Parallel and Distributed Computing

Percy Brown: 40382022

March 10, 2021

Lab Assignment 2

1 Multi-Threaded Algorithms for Computing Transpose of a Matrix

1.1 Diagonal-Thread - PThreads

In this algorithm, I generated for each diagonal element. For each diagonal position, the corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position.

First of all, I created a method that creates pthreads based on the number of threads specified and joins all threads created. The algorithm is as follows:

Algorithm 1 void createPthreads(int numThreads, void *(*next)(void *))
//Create a Pthreads object and joins all threads to aid in Diagonal-Threading algorithm

```
pthread_t threads[numThreads];
int tracker, thread;
for thread = 0; thread < numThreads; thread ++ do
    tracker = pthread_create(&threads[thread], NULL, next, (void *) thread);
    if tracker then
        print error message
        exit(-1);
    end if
end for
for thread = 0; thread < numThreads; thread ++ do
    pthread_join(threads[thread], NULL);
end for
```

I now have a method which extracts the start and end points for each thread, so that it is able to move diagonally in the matrix without having to accidentally reshuffle elements that have already been shuffled. Based on these points, it swaps the elements for each diagonal position such that the corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position. The algorithm is as follows:

Algorithm 2 void *diagonalPThreads(void *id)

//Diagonal-Thread Algorithm Using PThreads. For each diagonal position, the corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position

```
int start, end, row, column, threadID;
threadID = (int)id;
end = (threadID + 1) * (N/THREADS);
start = threadID * (N/THREADS);
for row = start; row < end; row ++ do
    for column = row + 1; column < N; column ++ do
        swap(row, column);
    end for
end for
pthread_exit(NULL);
In the main method, I run:
createPthreads(THREADS, diagonalPThreads);
```

1.2 Naive OpenMP Algorithm

A simple non-threaded approach, or serial algorithm, is to transpose the given matrix with a simple nested loop by swapping the elements $A[i,j]$ and $A[j,i]$. I inserted the `#PRAGMA OMP FOR NOWAIT` before for loops, to automatically parallelize the code and also not wait to threads to finish before execution continues if compiled as an OpenMP application. Below is the algorithm.

Algorithm 3 void naiveOMPMT(int **matrix, int size)

//Diagonal-Thread Algorithm Using PThreads. For each diagonal position, the corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position

```
int numThreads;
#pragma omp parallel
int threadID, row, column;
if threadID == 0 then
    numThreads = omp_get_num_threads();
end if
#pragma omp for nowait
for row = 0; row < size; row ++ do
    for column = row + 1; column < size; column ++ do
        swap(row, column);
    end for
end for
```

1.3 Diagonal Thread - OpenMP


Similar to the diagonal threading using PThreads, this time the function uses OpenMP instead. The number of threads is extracted using method *omp_get_num_threads()*. The threadID is extracted using the OpenMP method *omp_get_thread_num()* which is then used together with the number of threads in calculating the start and end points, in which the diagonal position is established with corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position. Below is the algorithm:

Algorithm 4 void diagonalOMPMT(int **matrix, int size)

//Diagonal-Thread Algorithm Using OpenMP. For each diagonal position, the corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position

```
#pragma omp parallel
int start, end, row, column, threadID, nThreads;
nThreads = omp_get_num_threads();
threadID = omp_get_thread_num();
start = threadID * (size/nThreads);
end = (threadID + 1) * (size/nThreads);
for row = start; row < end; row ++ do
    for column = row + 1; column < size; column ++ do
        swap(row, column);
    end for
end for
```

1.4 Comparative Table

 $N_0 = N_1$	Basic	<u>PThreads</u> Diagonal	Open MP	
			Naive	Diagonal
128	0.000306 s	0.001824 s	0.0017777 s	0.000141 s
1024	0.000088 s	0.000933 s	0.000852 s	0.010064 s
2048	0.000091 s	0.000781 s	0.000544 s	0.068831 s
4096	0.000080 s	0.000954 s	0.000421 s	0.274870 s

