Labolatory Exercise no: 2

Sizwe Lekoba (793314)*, Phiwe Mbongwa (606352)†, Ndabezinhle Ndlovu (545276)‡, Tshembani Sibuyi (838214)
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2050, South Africa
ELEN4020: Data Intensive Computing in Data Science

I. SOLUTION

A. First procedure: computing matrix transpose in OpenMP

The transpose algorithm is entirely done in parallel using OpenMP. The transpose is achieved by simultaneously interchanging each element in row with element in the column, using the diagonal element as a reference point. The performance of the algorithm is determined by calculating the time it takes the algorithm to transpose the matrix, excluding the time it takes to populate the elemnts of the matrix.

Algorithm 1 Transpose matrix algorithm in OpenMP

```
procedure (Transpose(array, squaresize))
    chunk ← squaresize
    $starterRow$ ← set to 0
    $starterCol$ ← set to 0
    Initialize i,j,k,nthreads,tid
    $Time$ ← omp get time finction
    Begining of omp parallel region
    Begining of omp parallel for loop
    bounds ← temp1
for(i = starterRow to squaresize+1)
if(i is greater than 0)
    $starterCol \leftarrow starterCol + 1$
     for(j = starteCol to squaresize)
    temp ← array[i][j]
    array[i][j] ← array[j][i]
    array[j][i] ← temp
end for
end for
    $Final$ ← omp get time function - Time
    print(final)
end
```

B. second procedure: computing matrix transpose on PThread

This function is implemented in a similar fashion to openmp. This is to ensure comparison of algorithm performance is valid. Implementation of pthreads requires a pointer to a function and a single pointer as a parameter. To achieve this a struct is used to store relevant values for the algorithm. The pthreads are created in a for loop with the arguments being an element from an array of struct pointers.This is to avoid conflicting manipulation of variables.

The transposing algorithm swaps the first row and column element by element. Parallelism is achieved by each thread swapping an element and then skipping the number of threads and swapping that element. After the whole row and column swap is complete, the function is called recuresively and passed a parameter pointing to the array one diagonal down.

Algorithm 2 Transpose matrix algorithm in PThread

```
procedure ( ThreadFunct(param))
    array ← param[0]
    $dimension$ ← param[1]
    $originaldim$ ← param[2]
    n ← set to 0
    temp ← points to n
    $rowptr$ ← array
    colptr ← array
    $rowptr \leftarrow rowptr + 1$
    colptr ← $colptr + *originaldim$
for(i = 1 to *dimension, increment i by nthreads)
    *temp ← *rowptr
    *rowptr ← *colptr
    *colptr ← *temp
    $rowptr \leftarrow rowptr + nthreads$
    colptr ← $colptr + nthreads * originaldim$
    end for
    array ← $array + 1$
    array ← $array + originaldim$
    $dimensions \leftarrow dimensions - 1$
    $newparam[3]$ ← (array, dimensions, originaldim)
if(*dimensions != 0)
    call thread(newparam)
```

C. Comparative table of perfomance

The perfomance for the different array sizes, the performance of the transposing algorithm is calculated when using different threads to compute the transpose. Table 1 shows the comparison of the performance of the algorithm when doing the transpose serially, usign OpenMP and PThread. These results were obtained using intel celeron cpu, with 2 cores, using linux operating system.

D. Results

For the implementation of pthreads, results were not obtained for matrix of size 8192x8192. According to analysis on the memory tracking tool, Valgrind, memory leaks occured when creating the threads, stating ”invalid read of size 4”. This occurs after allocating memory for the array. A reason

TABLE I
COMPARATIVE TABLE OF THE PERFORMANCE OF PTHREAD ALGORITHM AND OPENMP ALGORITHM

		Time (s)	
Matrix Size	No. of Threads	OpenMP	PThreads
128	4	0.000271	0.001170
	8	0.003100	0.001327
	16	0.000711	0.000945
	64	0.001698	0.00367
	128	0.001978	0.007295
1024	4	0.005607	0.000294
	8	0.000603	0.000578
	16	0.002108	0.000899
	64	0.007499	0.119327
	128	0.132659	0.037610
8192	4	0.167759	
	8	0.148541	
	16	0.146330	
	64	0.146349	
	128	0.151699	

for this leak may be due to the array pointers pointing to invalid memory making the elements inaccessable.Further theories include limited access to heap memory by threads versus stack memory. Future improvements include improved memory handling techniques. The maximum performance is achieved when the number of threads is approximately equal to the number of cores. As the number of threads increase, the performance is observed to decrease. A reason for the decrease in performance could be that the threads are trying to utilise the same CPU resource simultaneously.

E. Main Program: Dynamically Generating the arrays and allocating memory

Dynamic allocation of memory for arrays is achieved by using the malloc() function which takes in the arguments of the size of the data type of the array for openMP.

For pthreads, arrays were created statically to avoid memory leaks as previously discussed.