Lab 2

Brendon Swanepoel - 601949,
Anita de Mello Koch - 1371116,
Nicholas Kastanos - 1393410

1 Comparisons between openmp and Pthreads

$N_0 = N_1$	Basic (s)	Pthread - Diagonal (s)	Pthreads - Blocked (s)	OpenMP - Naive (s)	OpenMP - Diagonal (s)	OpenMP - Blocked (s)
128	5.5828×10^{-5}	0.0057640	0.089816	0.0054073	0.0017463	0.00067019
1024	0.0037808	0.051978	5.2157	0.0028294	0.0040702	0.0022198
2048	0.021078	0.18114	21.217	0.0090414	0.0098804	0.0046060
4096	0.088854	0.63340	85.449	0.033679	0.039016	0.021988
16384	1.7036	9.5218	-	1.1346	0.96734	0.43899

2 Parallel Threading Algorithm

In order to parallelise the transposition algorithm, the transposition is completed one row at a time in a separate thread. To avoid swapping elements which have already been swapped, the algorithm makes use of a nested `for` loop with a depth of two. The outer loop row counter `i` runs through every row, and the inner loop counter for the columns begins at `j=i+1`.

2.1 PThreads

Using Pthreads, each column is given to a single thread. This thread swaps that row and column, from $j = i+1$ until the length of the matrix.

2.2 OpenMP

2.2.1 Naive method

Using the naive method of parallelization, both the inner and outer loops are parallelized . This gives each thread an single swap to complete. This is done by collapsing the for loops.

2.2.2 Diagonal method

Using the diagonal method, only the outer for loop is parallelized . As a result, each thread swaps a single column from the diagonal to the rightmost element.

3 Block Transposition Algorithm

The Block Transposition algorithm completes the process by swapping two blocks and then each block is transposed. This is shown to result in transposition as shown below:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^T = \begin{bmatrix} \begin{bmatrix} A \\ C \end{bmatrix}^T \\ \begin{bmatrix} B \\ D \end{bmatrix}^T \end{bmatrix} = \begin{bmatrix} A^T & C^T \\ B^T & D^T \end{bmatrix}$$

3.1 PThreads

Creating this operation using PThreads, each block B_{ij} of 2 x 2 matrix elements are assigned to a thread and is swapped with the corresponding block B_{ji}. Once this operation has been completed, two more child threads are spawned to transpose B_{ij} and B_{ji}.

3.2 OpenMP

The parallelisation using OpenMP involves parallelising the block operations. Each thread uses additional parallelisation to swap B_{ij} and B_{ji} and subsequently each transposition is parallelised.

4   Discussion

From the results, for smaller matrix sizes, the non-threaded implementations outperform the threaded implementations. For these smaller matrix sizes, the loss in time due to overhead outweighs the benefit gained from parallelizing the programme. As the matrix size increases, it is clear the parallelized implementations outperform the non-threaded, with exception of the Pthread codes.

The Naive approach out performs the diagonal approach for smaller matrix sizes. However, as the matrix size increases, the diagonal method out performs the naive. This is due to the extra communication and overhead that takes place during the naive swapping. For large matrix sizes, this lose outweighs the speed from using individual threads for a single swap. The block transposition method elliminates more overhead than the diagonal method, resulting in its outperformance when using OpenMP.

OpenMP outperforms Pthreads for each implementation. OpenMP is better optimized than Pthreads for the C++ language, resulting in this difference.

5   Pseudocode

input : Pointer to a 2D square matrix
output: In-place transposition of matrix
for *Each row of the matrix* **do**
   for *Each column element after the current row value* **do**
    | Transpose current row and column elements;
   end
end

<div align="center">Algorithm 1: Basic Transposition Algorithm</div>

input : Pointer to a 2D square matrix
output: In-place transposition of matrix
Create array of Pthreads of the same size as the matrix dimension;
Create data array of structs for the threads to work on;
Initialise the Pthread attribute to joinable;
Populate data with pointers to the matrix and row numbers;
for *Each row of the matrix* **do**
   Create thread for each row of matrix;
   Call transposistion function;
end
for *Each row of the matrix* **do**
   | Join the created threads;
end

Transposition function
input : Pointer to thread argument
output: Transposed diagonal
Extract data from thread argument pointer;
Extract current row from thread argument pointer;
for *Each column element after the current row value* **do**
   | Transpose along current diagonal;
end

<div align="center">Algorithm 2: Diagonal Pthread Transposition Algorithm</div>

input : Pointer to a 2D square matrix, matrix size, block size
output: In-place transposition of matrix
Create array of Pthreads of the of size ((matrixSize/blockSize)*(matrixSize/blockSize)/2)
+((matrixSize/blockSize)/2);
Create data array of structs for the threads to work on; Initialise the Pthread attribute to joinable;
 Populate data with pointers to the matrix and block size;
Initialise independent iterator to 0;
for *Each row of the matrix in steps of the block size* **do**
    for *Each column of the matrix in steps of the block size* **do**
        Populate data aray with row and column indicies;
        Create thread for each row of matrix;
        Call transposistion function;
        Increase independent iterator by 1;
    end
end
for *Each row of the matrix* **do**
    Join the created threads;
end

Transposition function
input : Pointer to thread argument
output: Transposed diagonal
Extract data from thread argument pointer;
Extract current row from thread argument pointer;
for *The size of the block* **do**
    for *The size of the block* **do**
        Transpose the block elements;
    end
end

Algorithm 3: Block Pthread Transposition Algorithm

input : Pointer to a 2D square matrix
output: In-place transposition of matrix
OMP Parallel For Loop;
for *Each row of the matrix* **do**
    OMP Parallel For Loop;
    for *Each column of the matrix* **do**
        Transpose matrix emelents;
    end
end

Algorithm 4: Naive OpenMP Transposition Algorithm

input : Pointer to a 2D square matrix
output: In-place transposition of matrix
OMP Parallel For Loop;
for *Each row of the matrix* **do**
    for *Each column of the matrix* **do**
        Transpose matrix emelents;
    end
end

Algorithm 5: Diagonal OpenMP Transposition Algorithm

input : Pointer to a 2D square matrix
output: In-place transposition of matrix
OMP Parallel For Loop;
for *Each row of the matrix in steps of the block size* **do**
    for *Each column of the matrix in steps of the block size* **do**
        OMP Parallel For Loop;
        for *The block size* **do**
            for *The block size* **do**
                Transpose block emelents;
            end
        end
    end
end

Algorithm 6: Block OpenMP Transposition Algorithm