COEN – 319 Parallel Computing

Assignment – 1

Dense Matrix Multiplication (OpenMP)

REPORT

Student Name: Piyush Kulkarni

Campus ID: W1629006

The purpose of this assignment is to become familiar with OpenMP by implementing dense matrix-matrix multiplication.

Program Steps:

I. **A single thread creates the matrices:**
Initially, a single thread (master thread) starts executing the `main()` function of the code, which has a function call to `create_mat()` which creates two random matrices, A and B.

These matrices are further passed as arguments to two functions, one for multiplying without tiling, and one with tiling. These functions are where I have implemented parallelization using OpenMP directives.

II. **Multiple threads work together to do the matrix multiplication**

Matrices A and B are further passed to two functions `mult_mat()` and `matrix_matrix_mult_by_tiling();`

a. Multiplication without tiling (`mult_mat()`)
As the main thread starts executing this function, and reaches `#pragma omp parallel` construct, it creates a new team of threads. The number of threads is controlled by the environment variable by using `export OMP_NUM_THREADS`.
The for construct splits the proceeding for loop so that each thread in the current team handles a different portion of the loop, however, in an arbitrary order.
So, each thread gets a different section of the loop, and they execute their own sections in parallel. Although, since different iterations in the loop may take different times to execute, I have implemented dynamic scheduling.
Collapse clause is used to distribute the threads amongst multiple nested for loops. The number 2 in `collapse(2)` specifies the number of nested loops that are subject to the work-sharing semantics of OpenMP for construct.
Thus, each thread will perform calculations for the resultant element, based upon which part of the loop it is executing, as assigned to it by dynamic scheduler in OpenMP runtime library.
At the end of the parallelized for loop, control is returned back to the main thread, and the resultant matrix is updated.

b. Multiplication with tiling (`matrix_matrix_mult_by_tiling()` and `matrix_matrix_mult_tile()`)

In this method, the input matrices are broken down into sub-matrices, called tiles.

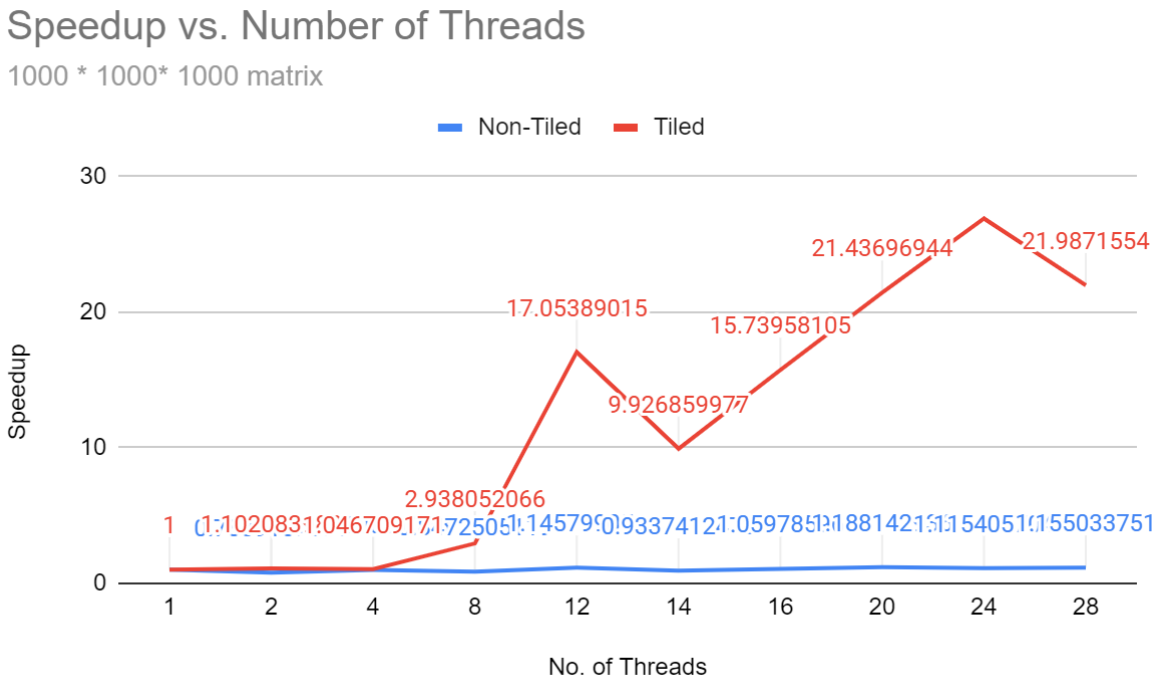
Tiling multiplication is done using two functions, one in which tiles are created, and the other in which these tiles are multiplied together to form a resultant tile, which is further stored in the resultant matrix, to be operated upon again in the next iteration of adjacent tile, to obtain a final result.

Here, I have implemented parallelization to one of these functions, where tiles are created `matrix_matrix_mult_by_tiling()`, and passed through tile multiplication function `matrix_matrix_mult_tile()`.

The master thread is divided into a team of threads, each thread getting assigned one tile to operate upon, using the `#pragma parallel for` construct.

Observations:

No. of Threads	Runtime – NonTiled (μs)	Runtime -Tiled (μs)
1	4534011.83	5351441.661
2	5783776.65	4855747.61
4	4597992.41	5112634.73
8	5351441.83	1821425.06
12	3957074.17	313795.95
14	4855747.61	539087.05
16	4278233.53	339998.99
20	3816049.14	249636.11
24	4064899.27	198836.42
28	3925436.66	243389.45



As depicted in the chart, it is observed that tiling increases the speedup of the operation.

However, only after the number of threads reaches 8 and above, is that the speedup shows a significant improvement. Until then, tiling shows very less to no improvement over the traditional matrix multiplication operation.

Multiplication without tiling shows no speedup improvement, even after increasing the number of threads.

As we increase the size of input matrices, the runtime of the program increases, thus showing direct proportionality between these two. This happens because the load on each thread (no. of operations) increases, thus each one takes more time to execute.

This can be tackled by increasing the number of threads the code executes on. However, if the size of the matrices increases along with the number of threads, there will not be a significant increase in speedup.