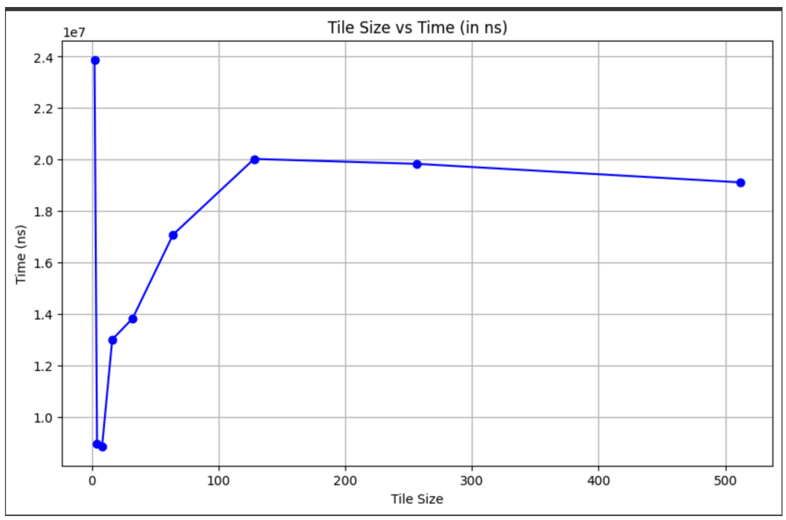
CS 484 - MP1

NetId - Harsha4
Date - 02/17/24

Link to Google Colab* - *CS484 Anaylsis.ipynb

1.1

1.

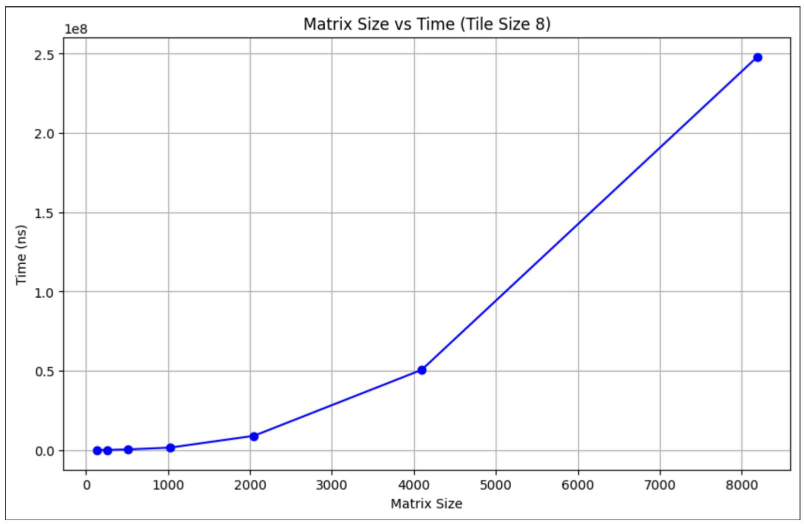


From the `cache_properties.txt` file, we see the following information

LEVEL1_ICACHE_LINESIZE	64
LEVEL1_ICACHE_ASSOC	8

This signifies that each cache line is of size 64 bytes. Since the array is of doubles, there can be 8 elements in one cache line. The set associativity of this cache is 8 which means that there are 8 cache lines in total. As a result, using tile size of 8 will provide maximum ability to exploit the spatial cache locality and hence take the least amount of time.

2.



The above graph shows the time taken by varying sizes of N to complete execution. Specifically, using a tile size of 8 fits well with the machine's specifications since the cache line is 64 bytes long and can fit 8 doubles. Having 8 way set associativity also means it can fit 8 cache lines in the L-1 cache. As a result, when the tile size is 8, the spatial locality is extensively used and hence helps in more cache hits making 8 an optimal choice.

Conclusion

Using a tile of size 8 improves performance by over 4x when the array is large enough.

1.2

```
Benchmark                                                                     Time             CPU
---------------------------------------------------------------------------------------------------
BM_Transpose_Naive/matrix_size:512/process_time/real_time                   888326 ns       888252 ns
BM_Transpose_Tiled/matrix_size:512/tile_size:8/process_time/real_time       214977 ns       214961 ns
BM_Multiply_Naive/matrix_size:512/process_time/real_time                 444028039 ns    443994178 ns
BM_Multiply_Tiled/matrix_size:512/tile_size:8/process_time/real_time     100699554 ns    100691922 ns
BM_Multiply_Transposed/matrix_size:512/process_time/real_time             87305868 ns     87296634 ns
```

From: `gbench_results.txt`

multiply_results.txt

mp1 > writeup > multiply_results.txt

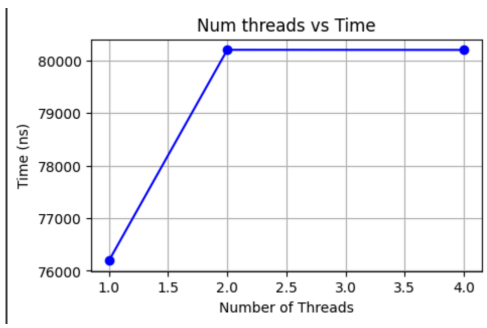
```
Average execution time of multiply_basic(): 1.007680
Average execution time of multiply_tiled(): 0.131155
Average execution time of multiply_transposed(): 0.088673
```

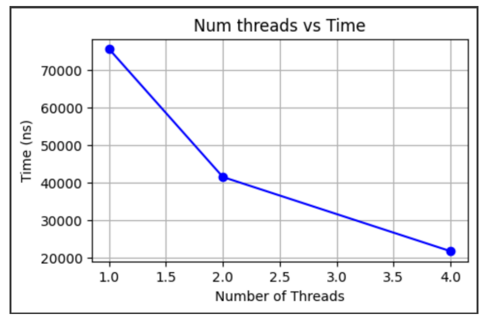
From: `multiply_results.txt`

Based on the tests and benchmark tools used above, it is evidently clear that Multiply_Transposed works better than Multiply_Tiled. This can be explained as follows - Transposing one of the matrices can enhance memory access patterns by improving spatial locality. When accessing elements from the transposed matrix, consecutive memory accesses are more likely to be adjacent in memory, which can reduce cache misses and improve performance.

1.3

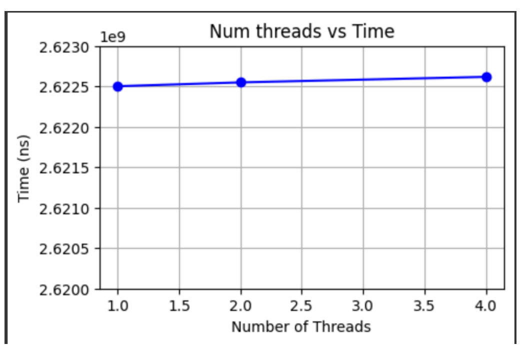
1.

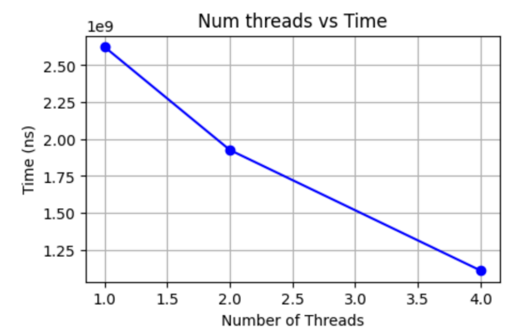




Function: `squares_parallel`

Without OpenMP With OpenMP





Function: `primes_parallel`

Without OpenMP With OpenMP

In both functions, Using 4 threads reduces the time drastically.

For the first function, the result of using 4 threads increased speedup by 4X and for the second function, the result of using 4 threads increased speedup by 2X. Hence #pragma omp parallel for increases speedup by 4X and 2X respectively.