CS4532 Concurrent Programming

Take Home Lab 3 and 4

Task:

- use a “parallel for” statement to iterate through a loop body in parallel
- improve the performance of the given program using a suitable optimization technique

Group Members:
Dinindu Nissanka 120434D
Menaka Lahiru Sirisena 120628C

Tasks:

1. Please refer to sequential_multiplication.cpp in Lab_3_4_120434D_120628C.zip

2. OpenMP library was used in this lab

3. Please refer to openMP_parallel_multiplication.cpp in Lab_3_4_120434D_120628C.zip

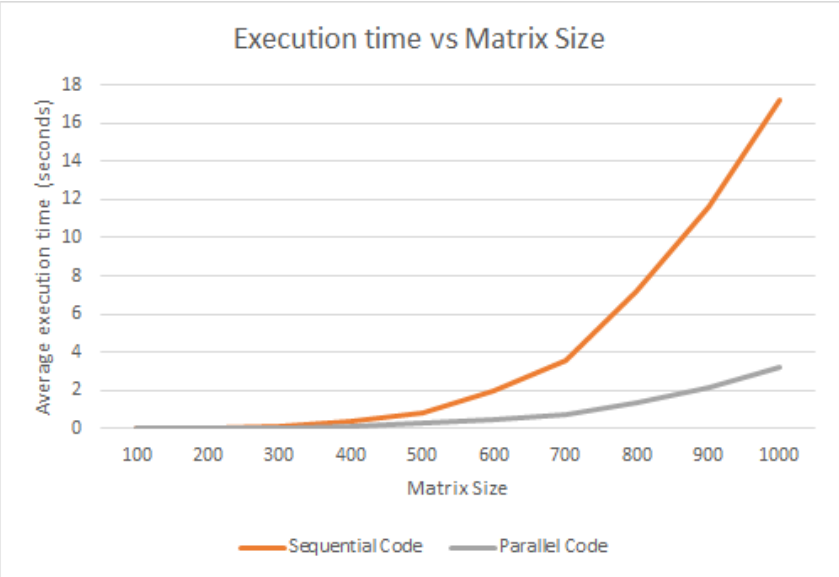
4. **Execution time (sequential and parallel) values and graphs**

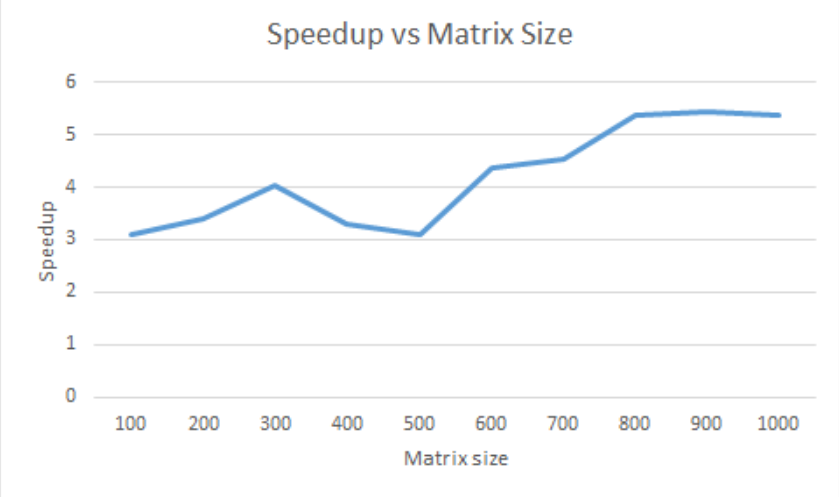
Number of samples = $n = \left(\frac{100\ zs}{r\underline{x}}\right)$

Note: All the sample sizes are calculated based on the initial sample size of 20.

Required number of program executions		
Matrix step size	**Sequential**	**Parallel**
100	59	78
200	6	13
300	2	4
400	1	2
500	1	1
600	1	1
700	4	2
800	4	2
900	1	1
1000	1	1

Matrix Size	Average execution times in seconds		Speedup
	Sequential Code	Parallel Code	
100	0.00507273	0.00164587	3.082096399
200	0.0491532	0.0144235	3.407855236
300	0.173061	0.042971	4.027390566
400	0.413824	0.124993	3.310777404
500	0.865356	0.280956	3.080041003
600	1.95462	0.446581	4.376854367
700	3.57461	0.785662	4.549806405
800	7.21654	1.3452	5.36465953
900	11.6528	2.14125	5.442054874
1000	17.258	3.21595	5.366376965





5. Observations:

- **CPU Specification:**

Cores	4
Hardware Level Threads	8
Name	Intel Core i7 3630QM
Code Name	Ivy Bridge
Package	Socket 988B rPGA
Technology	22nm
Specification	Intel Core i7-3630QM CPU @ 2.40GHz
Family	6
Extended Family	6
Model	A
Extended Model	3A
Stepping	9
Revision	E1/L1
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, Intel 64, NX, VMX,AES, AVX
Virtualization	Supported, Enabled
Hyperthreading	Supported, Enabled

Caches:

L1 Data Cache Size	4 x 32 KBytes
L1 Instructions Cache Size	4 x 32 KBytes
L2 Unified Cache Size	4 x 256 KBytes
L3 Unified Cache Size	6144 KBytes

- **Justifications of the gained speed up knowing the architecture of the CPU:**
  According to Amdahl's law, the maximum speedup which can be gained as defined as follows;

  Max speedup = 1 / (1 - p + p / n)

  For the CPU used in this lab, value for n is 8 i.e. hardware level thread count. Hence the theoretical max speedup would be 1/ (1/8) = 8 when p reaches 100%. The speedup values gained are justifiable since they all are below the theoretical maximum speedup values.

- **Observed behavior in graphs to the architecture of the CPU:**
  It can be seen that the execution time for the matrices which are less than 500 in size do not have a significant difference when the matrix multiplication is done in parallel manner when compared with sequential matrix multiplication. This can be due to the higher penalty caused during the thread spawning is much higher than the benefit gained from the “parallel for” implementation in matrix multiplication.

6. Two ways to optimize matrix-matrix multiplication

- **Matrix Transpose:**
  Matrix values are stored sequentially in the memory. When the action performed processor first go and see whether required rows and columns are in L1 cache. If the data is stored in L1 cache, it will be retrieved. If it is not then go and check L2 cache. If the data is not in L2 cache, processor will check L3 cache. If the data is still not there then the array will be retrieved from the storage. In the normal matrix multiplication first matrix's rows are multiplied with second matrix's columns. But cache load rows from the matrix. So to perform matrix multiplication cache needs to load each row from the second matrix. This leads to great cache misses. But when we stored the transpose of the second matrix, the initial columns are saved as rows. So matrix multiplication can be done in less number of cache loads. This will reduce the time of the matrix multiplication.[1]

- **SIMD approach:**
  SIMD means Single Instruction Multiple Data. SIMD approach is achieved by using processors defined SIMD instructions. These instructions are special instructions. Using SIMD instructions can increase performance greatly when the same operation are to be performed on multiple data objects. Matrix matrix multiplication is a good example where we can use SIMD instructions. Examples of SIMD instructions are SSE, SSE2, SSE3 and AVX. We use SSE3 to optimize matrix multiplication.[2] [3]

7. Please refer to improved_multiplication.cpp in Lab_3_4_120434D_120628C.zip

8. Execution time graphs of optimized parallel matrix multiplication

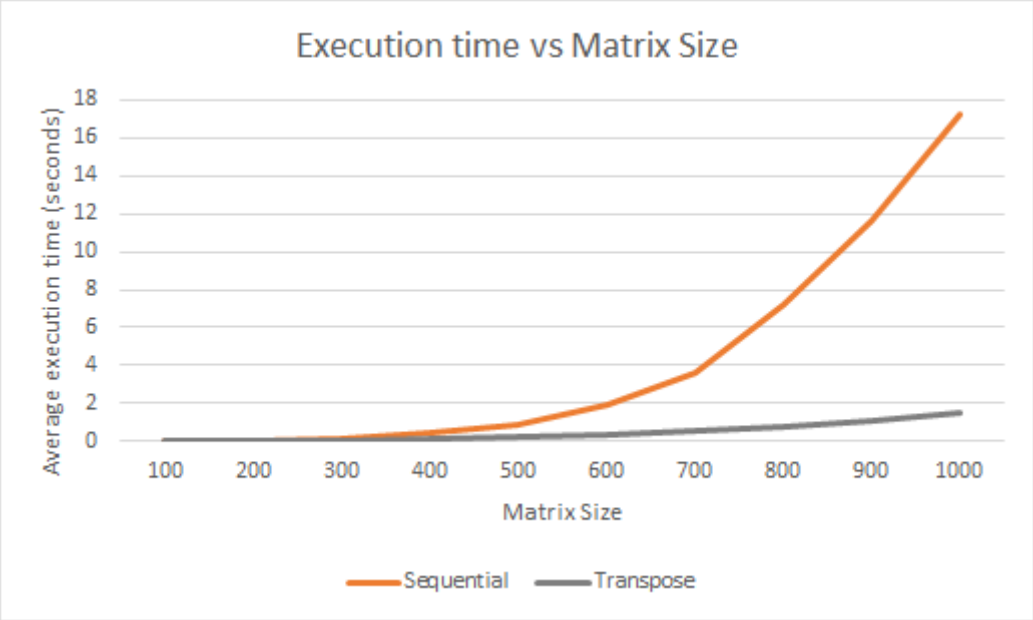
Number of samples = $n = \left(\frac{100\ zs}{r\underline{x}}\right)$

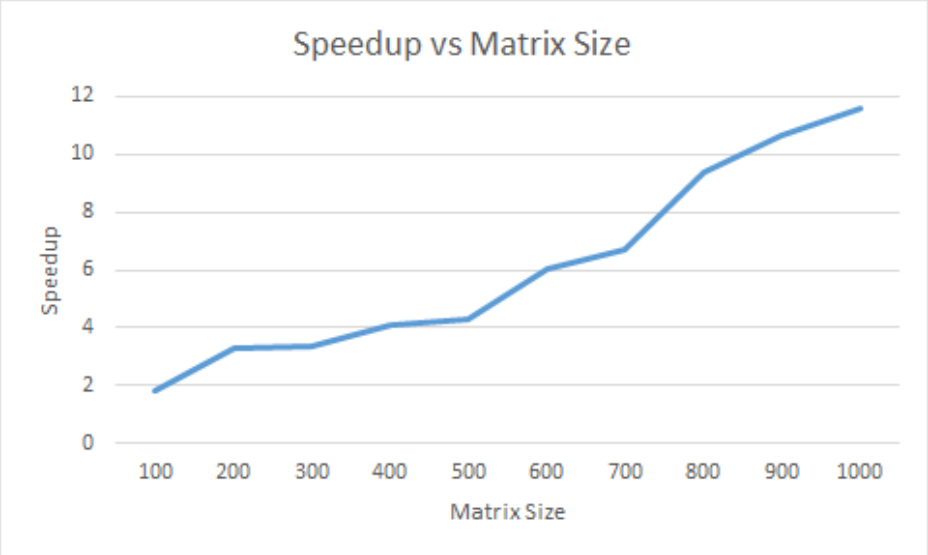
Note: All the sample sizes are calculated based on the initial sample size of 20.

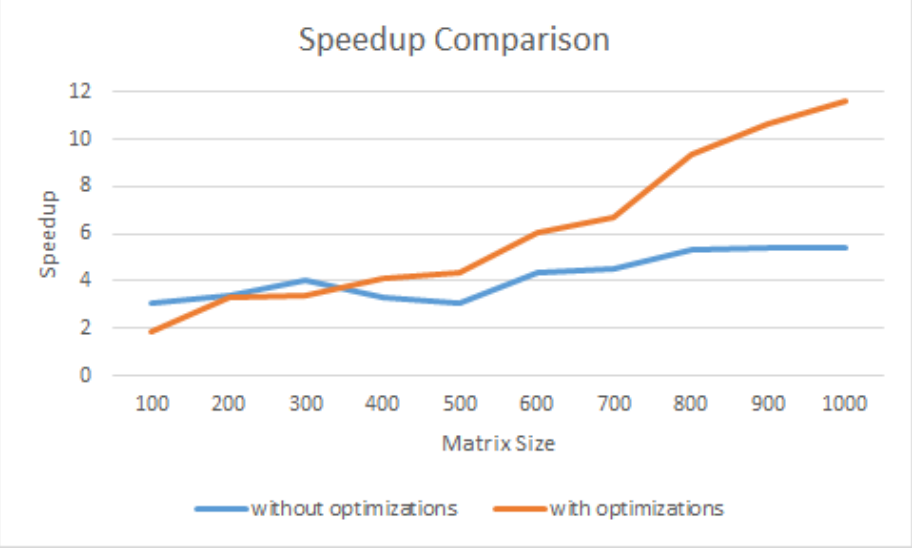
Matrix Transpose:

Required number of program executions	
Matrix step size	**Transpose**
100	13
200	5
300	2
400	1
500	1
600	1
700	2
800	1
900	1
1000	3

Matrix Size	**Average execution times in milliseconds**		**Speedup**
	Sequential	**Transpose**	
100	0.00507273	0.00277085	1.830748687
200	0.0491532	0.0150098	3.274740503
300	0.173061	0.051034	3.391092213
400	0.413824	0.101067	4.094551139
500	0.865356	0.200132	4.323926209
600	1.95462	0.325215	6.01023938
700	3.57461	0.531853	6.721048861
800	7.21654	0.771511	9.353774606
900	11.6528	1.09573	10.63473666
1000	17.258	1.48765	11.60084697


Execution time vs Matrix Size
Average execution time (seconds)
18
16
14
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size
Sequential
Transpose

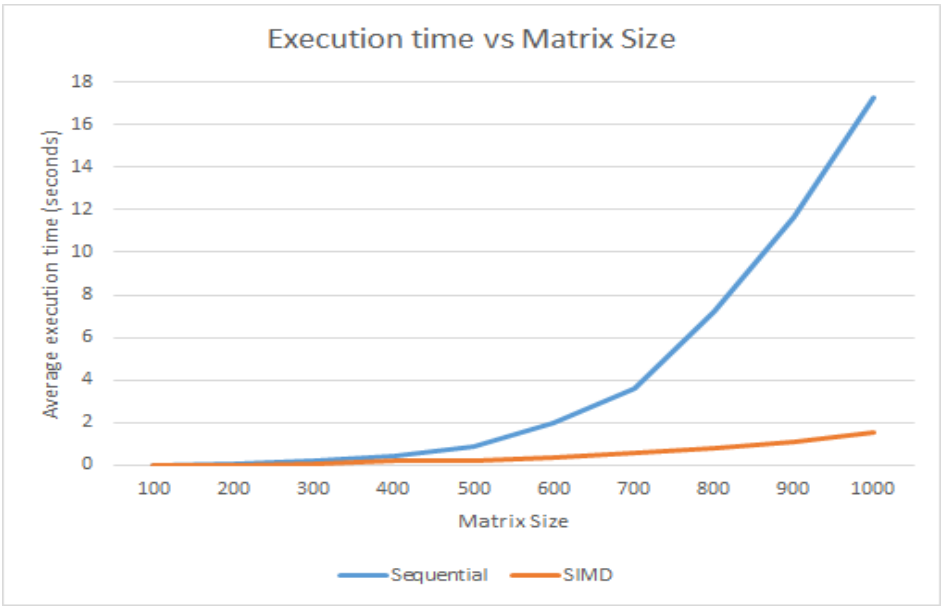

Speedup vs Matrix Size
Speedup
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size

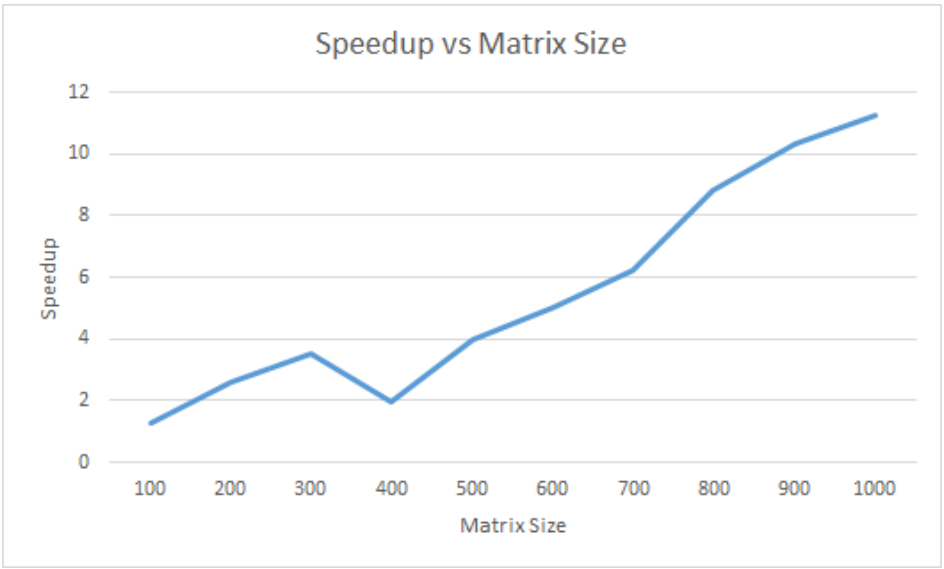

Speedup Comparison
Speedup
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size
without optimizations
with optimizations

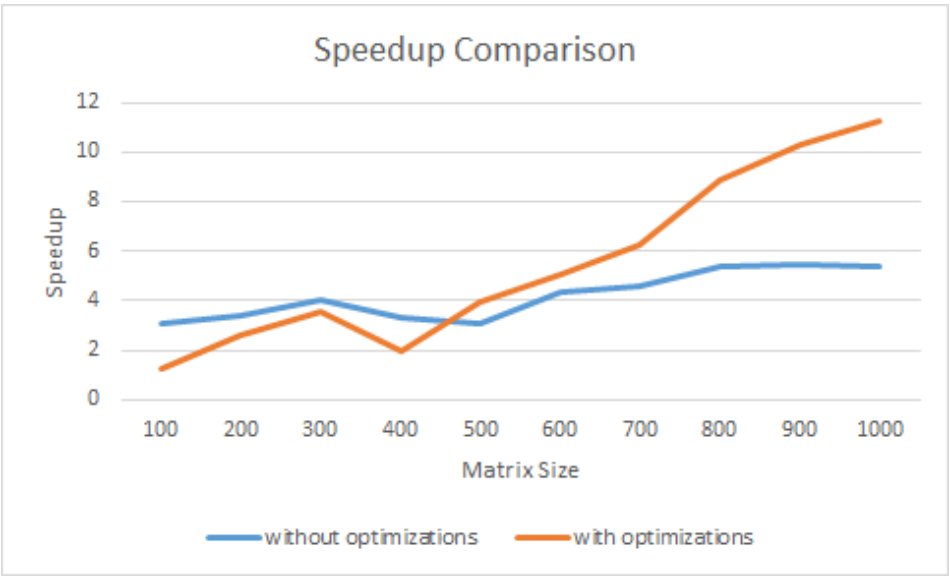
SIMD(SSE3) Approach:

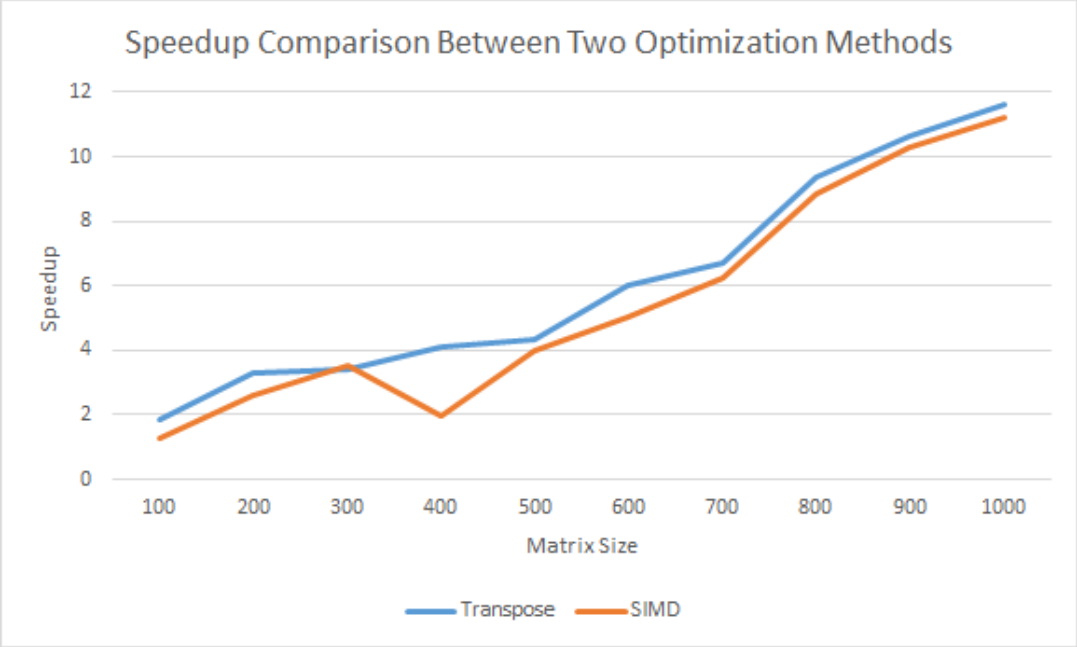
Required number of program executions	
Matrix step size	**Transpose**
100	21
200	7
300	3
400	1
500	1
600	1
700	1
800	1
900	1
1000	1

Matrix Size	**Average execution times in milliseconds**		**Speedup**
	Sequential	**SIMD**	
100	0.00507273	0.00396799	1.278413
200	0.0491532	0.0189921	2.588086625
300	0.173061	0.0492783	3.511910922
400	0.413824	0.2089	1.98096697
500	0.865356	0.217573	3.977313361
600	1.95462	0.38858	5.030161099
700	3.57461	0.572123	6.247974649
800	7.21654	0.81589	8.844991359
900	11.6528	1.13003	10.31193862
1000	17.258	1.5354	11.24006773


Execution time vs Matrix Size
Average execution time (seconds)
18
16
14
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size
Sequential
SIMD


Speedup vs Matrix Size
Speedup
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size


Speedup Comparison
Speedup
12
10
8
6
4
2
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size
without optimizations
with optimizations



9. Observations in relation to the optimization technique(s) you selected and CPU architecture

Matrix Transpose:

Transpose the matrix and then multiply gives a significant increase in speed up when compared to normal sequential multiplication. We can see that the speed up gets increased with the matrix size. For small matrices this approach does not increase the speed up in a significant manner. That is because matrix transpose approach reduce cache misses. But cache can load all the data of the matrix if it is small in size. Hence for small matrices this approach does not gives significant speed up because cache misses are relatively low when the matrix is small. But when the matrix gets bigger there will be huge number of cache misses. Those cache misses can be reduced by transposing the matrix. Hence we can see a significant speed up when the size of matrix gets increased.

SIMD(SSE3) Approach:

Matrix multiplication involves same instruction do over a long period of time on multiple data. Single Instruction Multiple Data is a very good alternative to improve speed up of the matrix multiplication. SSE3 is a SIMD instruction set developed by Intel.
In normal approach processor gets one data from matrix 1 and then gets another from matrix 2 and then multiply them. This action continues and the summation of all the multiplication is calculated. But in SIMD instruction we can get four values from matrix one and four values from matrix 2 and multiply them and gets the summation. This makes the speed up more increased. The SSE3 can manage 8 double numbers. So 4 from matrix 1 and another 4 from matrix 2 retrieved. Because of this parallelisation speed up of the SIMD approach is better than sequential matrix multiplication.

References:

[1]Timothy J. Rolfe. Program Optimization: Enforcement of Local Access and Array Access *via* Pointers [Online]. Available: http://penguin.ewu.edu/~trolfe/MatMult/MatOpt.html

[2]Intel Corporation. (1999). Streaming SIMD Extensions - Matrix Multiplication [Online]. Available: http://download.intel.com/design/PentiumIII/sml/24504501.pdf

[3]Streaming SIMD Extensions. (2017) Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions