Name: _____

Lab for ITSC 3181, Introduction to Computer Architecture, Spring 2023

Lab #13 and #14: Memory and Cache Access, Stride Access and Loop Ordering with Matrix Multiplication
Due by 04/26 (NO extension) and count for 2% of the final accumulated percentage grade.

	Task 1 (75%)					Task 2 (10%)	Task 3 (15%)			Total
	ikj (multMat2)	jik (multMat3)	jki (multMat4)	kij (multMat5)	kji (multMat6)		Question 1	Question 2	Question 3	
Percentage	15%	15%	15%	15%	15%	10%%	5%	5%	5%	
Your grade										

In this lab, given 6 different implementation of matrix multiplication C[N][N] = A[N][N] * B[N][N], you 1) inspect the memory access pattern of each matrix (stride), 2) find the number of misses, number of accesses and miss rate of each matrix access using a simple 3-block 32-byte-block cache, and 3) also execute the program in real to compare the performance of the 6 implementations and explain the impact to performance by memory/cache access.

Background:

Matrices are 2-dimensional array data structure wherein each data element is accessed via two indices. And a 2-dimensional array is stored in memory in row-major. Stride of two consecutive memory accesses is the number of elements in memory between the addresses of the accesses. It is used to describe the temporary and spatial locality of memory accesses. If the stride of two access is 0, memory accesses have temporary locality since the second access is accessing the same item as the first one. If the stride is 1 or -1, we consider memory accesses have spatial locality since the second access is touching the memory of immediately next the first one. If the stride is a large number, there is no spatial or temporary locality. Give the two-dimensional array A[n][n] in row-major above, A[i][j] and A[i][j+1] / A[i][j-1] has stride 1 (same row access, spatial locality). A[i][j] and A[i+1][j]/A[i-1][j] has stride n.

To multiply two matrices, we can simply use 3 nested loops, assuming that matrices A, B, and C are all n-by-n and stored in one-dimensional row-major arrays, and n is 64. This implementation is called **ijk (multMat1)** and there are other five implementations.

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i+j*n] += A[i+k*n] * B[k+j*n]; //C[i][j] += A[i][k]*B[k][j]
```

Fact: Matrix multiplication operations are at the heart of many linear algebra algorithms, and efficient matrix multiplication is critical for many applications within the applied sciences.

In the above code, note that the loops are ordered i, j, k. If we examine the innermost loop (the one that increments k), we see that it...

- moves through B with stride 1
- moves through A with stride n
- moves through C with stride 0

Below table shows more detailed simulation of the array element access for each matrix for ijk (multMat1) for the inner most loop for k from 0 to 63. **We also assume we have a simple 3-block cache and each block is 32 bytes (can store 8 elements of an array). One array (A, B, C) use one block, block-A, block-B, and block-C,**

assuming all blocks are empty at the beginning. The table also shows cache miss/hit of each memory access, the total number of accesses and misses and the calculated miss rate.

ijk (multMat1)																				
	k=0		k=1		k=2		...	k=7		k=8		...	k=62		k=63		Summary			
Array	Element	Cache H/M	Element	Cache H/M	Element	Cache H/M	...	Element	Cache H/M	Element	Cache H/M	...	Element	Cache H/M	Element	Cache H/M	Stride	# Accesses	# Misses	Miss Rate	Notes
Load for A	A[0]	M	A[n]	M	A[2n]	M	...	A[7n]	M	A[8n]	M	...	A[62n]	M	A[63n]	M	n	64	64	100%	Every access is Miss
Load for B	B[0]	M	B[1]	H	B[2]	H	...	B[7]	H	B[8]	M	...	B[62]	H	B[63]	H	1	64	8	0.13	1 out of 8 accesses is miss. Each memo
Load for C	C[0]	M	C[0]	H	C[0]	H	...	C[0]	H	C[0]	H	...	C[0]	H	C[0]	H	0	64	1	0.02	For all the 64 accesses, the first one is a
Store for C	C[0]	H	C[0]	H	C[0]	H	...	C[0]	H	C[0]	H	...	C[0]	H	C[0]	H	0	64	0	0	All 64 accesses are hit
																	Total	256	73	0.29	Miss Rate = # Misses / # Accesses

To compute the matrix multiplication correctly, **the loop order doesn't matter**. **BUT**, the order in which we choose to access the elements of the matrices can have a **large impact on performance**. Caches perform better (more cache hits, fewer cache misses) when memory accesses take advantage of spatial and temporal locality, utilizing blocks already contained within our cache. Optimizing a program's memory access patterns is essential to obtaining good performance from the memory hierarchy.

Task 1 (Total 75%, 15% per implementation for the remaining 5 implementations):

For the other five implementations of matrix multiplication, fill in the excel sheet table for memory access pattern of each matrix (stride), the number of misses, number of accesses and miss rate of each matrix access using the simple 3-block 32-byte-block cache. The table is only for showing the 64 iterations of the inner most loop. You only need to fill in the empty yellow cell of the sheet, and the green cells and the "Miss Rate" plot will be automatically populated after the yellow cells are filled in.

Download the matrixMultiply.c file (https://passlab.github.io/ITSC3181/notes/Lab_13/matrixMultiply.c) and the Lab_13_MemoryCacheAccess_MM.xlsx (https://passlab.github.io/ITSC3181/notes/Lab_13/Lab_13_MemoryCacheAccess_MM.xlsx).

Task 2 (10%):

Go to https://repl.it/languages/c and cut-past the content of the matrixMultiply.c file into the editor window. Then save the file by clicking the save button. On the terminal window, type the following two commands to compile and execute the program:

```
gcc -O3 main.c -o mm
./mm
```

This will run some matrix multiplications according to the six different implementations in the file, and it will tell you the performance (*speed)* at which each implementation executed the operation. The unit "Gflops/s" reads, "Giga-floating-point-operations per second." **THE BIGGER THE NUMBER THE FASTER IT IS RUNNING!** Input the performance number in the yellow cells in the Excel sheet's "Task 2" table. The numbers (all are 1) in the sheet are just dummy number and you need to replace with what you get. The performance figure will be automatically plotted.

Task 3 (15%):

Comparing the two figures ("Miss Rate" and "Performance" Figures) and relating them to the stride information of the six implementations, write a one-page summary to answer the following three questions. Make sure your writing uses the terms we discussed during the class (spatial and temporal locality, cache, miss/hit ratio, etc).

1. Which ordering(s) perform best for these 1000-by-1000 matrices? **Why?**
2. Which ordering(s) perform the worst? **Why?**
3. How does the way we stride through the matrices with respect to the innermost loop affect performance?

Submission:

Use the provided template file for the submission. Lab_13_MemoryCacheAccessLoopOrderStride_SubmissionTemplate.docx (https://passlab.github.io/ITSC3181/notes/Lab_13/Lab_13_MemoryCacheAccessLoopOrderStride_SubmissionTemplate.docx). For Task 1 and 2, copy and paste all the Excel tables and figures to the template file. For task 3, write your summary in the file.