-0.7	-0.4	0.6	1.0	-0.6	-0.5	0.3	-0.8	0.9	-0.8
-0.7	-0.1	-0.6	0.1	0.9	0.1	-0.9	0.6	0.2	0.5
1.0	-0.4	0.5	0.7	-0.8	0.9	-0.8	0.8	-0.6	0.8
0.8	0.6	-0.5	0.2	0.1	-0.9	-1.0	0.2	0.3	0.5
0.0	0.4	0.8	-0.7	-0.7	0.9	-1.0	0.1	0.7	-0.9
-0.5	-0.3	0.7	-0.2	-0.7	0.0	0.0	-0.7	-0.1	0.9
-0.4	0.3	-0.8	-0.4	0.1	0.7	0.2	1.0	-0.9	0.0
0.5	0.9	0.3	0.1	0.5	-0.4	0.8	-0.8	0.9	1.0
0.5	0.2	-1.0	0.4	0.2	1.0	-1.0	0.9	1.0	-0.3
0.8	-0.4	0.2	-0.9	0.5	-0.7	0.1	0.2	0.7	-0.7

24-780 B—Engineering Computation

Assigned: Mon. Nov. 22, 2021
Due: Tues. Nov. 30, 2021, 11:59pm
(But will do most of the work in the lecture on Mon., Nov. 22)

Problem Set 10: Matrices Better

This assignment will not require any graphics or sound, so keep working hard on your team project. By the way, this is the last assignment.

Once again, we are creating an assignment that builds on previous work, so I do not want you to start the current assignment at a disadvantage. You have the following choice:

- Take your PS09 solution as your starting point
  (advantage is that you carry through with your assumptions and data models rather than trying to figure out mine)

OR

- Use my solution to PS09 on Piazza
  (advantage is that you "cut your losses")

Task 1 (Adjust old matrixMultiply function)

Our old matrixMultiply() function multiplied the two matrices completely (all rows and columns). In order to make use of multi-threading, we need to be able to perform matrix multiplication on a partition of the matrices. In particular, we need to provide start and end row on the left-hand matrix and start and end column on the right-hand matrix. Since we want to keep the old functionality, these additional parameters should be optional (i.e., should have default values):

```
// Multiplies the matrix with otherMatrix and stores the result in the
// resultMatrix. Note that otherMatrix is the right-hand operand (since matrix
// multiplication is NOT commutative). Use optional parameters for portioning.
void matrixMultiply(Matrix2D<T, NC, NR>& otherMatrix,
              Matrix2D <T, NR, NC>& resultMatrix,
              int startRow = 1, int endRow = NR,
              int startCol = 1, int endCol = NC);
```

Task 2 (Matrix multiply using multi-threading)

Matrix multiplication is rather slow. Implement the following function so that we can use the full power of 21st century computing. In addition to the following functions, you need to come up with a mechanism for controlling the threads and avoiding the race condition.

```
// similar to matrixMultiply, but uses multi-threading
void matrixMultiplyParallel(Matrix2D<T, NC, NR>& otherMatrix,
          Matrix2D <T, NR, NR>& resultMatrix);

// thread entry static function
public: static void threadEntry(Matrix2D<T, NR, NC>* thisPtr);
```

In addition to adding functionality to the class for Matrix2D, also edit the main() function (in a separate file) that is used to test the functions of the class.

Deliverables

All files you create, very appropriately named and zipped together which at a minimum include:

ps10_matrix_better_andrewID.h << contains declaration and bodies of the classes you develop

ps10_matrix_checker_andrewID.cpp << contains a series of simple tests for your class functions

Upload the zip file to the class Canvas page before the deadline (Tues, Nov. 30, 11:59pm).

Learning Objectives

Deep Copy

Multi-threading

Template classes.

Array manipulation.

Computational programming using matrices.

Using tester programs for class debugging.

Searching references (online or textbook) for C++ library functions.