COSC 561
copt - optimizations for a C program

The copt starter code contains unoptimized implementations of four common operations in C code: 1) matrix initialization, 2) array initialization, 3) factorial computation, and 4) matrix multiplication. Your assignment is to write optimized versions of each operation using the optimizations discussed in lecture. You will also measure and report the speedup obtained by your optimized implementation. The starter directory in `copt.tar.gz` contains the following files:

`Makefile`	- copt Makefile
`copt.c`	- main copt program, defines all optimized and unoptimized routines
`test.sh`	- script for running sample inputs for copt

We have also included two reference executable files `copt_O0_ref` and `copt_O3_ref` that contain our working implementation compiled with either -O0 (i.e., with no optimizations) or -O3 (i.e., with a standard set of optimizations supplied by gcc). The executables were built on and will work on all of the hydra machines at UTK.

For your assignment, you only need to modify the copt.c program and test using test.sh. You should focus your testing on the -O0 version of the executable to eliminate potential interactions between the optimizations you implement and the optimization passes applied by the compiler with -O3. After you have completed optimizing and testing with -O0, compile your copt version with -O3, and collect the results with test.sh. Note that test.sh requires a single argument, which is the name of the copt executable file. In this way, you can easily test different copt versions with the same script.

It is important to note that copt reports performance as raw execution time. Due to variations in events happening on the machine (especially on a shared machine like hydra), execution time can vary significantly from run to run. We recommend you run your tests several times to ensure you are getting consistent results. Typically, professionals would run such tests in a loop more than ten times and collect the average and variation of each execution time result. However, for this assignment, you are only required to test enough so that you are convinced the results are consistent and reproducible.

You will also submit a project report with the following information:

1. Descriptions of the optimizations you implemented for each operation, as well as any difficulties or challenges you had in implementing your approach.
2. Comparisons and analysis of the results with your copt program and the provided reference executables. You should include the output of test.sh for both the -O0 and -O3 versions of your copt executable (or, if you prefer, the average output over several runs of test.sh).
3. In your comparisons and analysis, please answer the following questions:
   (a) Which optimizations are most effective for each operation?
   (b) Some optimizations are less effective when -O3 is enabled. Why do you think this is the case?
   (c) Are any optimizations just as effective (or even more effective) when -O3 is enabled? If so, which optimization(s)? Why do you think these optimization(s) are still effective when -O3 is enabled?

Extra Credit Opportunity: If you can demonstrate that your implementation of any given operation consistently outperforms the reference executable by more than 10% with the -O0 version, we will give up to 10 points of extra credit on this assignment for each such operation. Thus, you could potentially receive 40 points of extra credit on this assignment and receive a maximum possible score of 140 / 100.

For your submission, you should upload a gzipped tar file (created with `tar cvzf ...`) with your source files and a pdf of your report to the Canvas course website before midnight on the assignment due date. Partial credits will be given for incomplete efforts. However, a program that does not compile or run will get 0 points. Point breakdown is below:

• optimizations for each operation	(60)
• project report	(40)

Sample test.sh output on hydra:

```
mrjantz@hydra2$ ./test.sh copt_O0_ref
Running MATRIX_INIT with n = 3000 loop = 200

UNOPTIMIZED(ms):       33133.0
OPTIMIZED(ms):         12200.0
SPEEDUP:                  2.72

Running ARRAY_INIT with n = 300000 loop = 20000

UNOPTIMIZED(ms):       26250.0
OPTIMIZED(ms):         13400.0
SPEEDUP:                  1.96

Running FACTORIAL with n = 20 loop = 200000000

UNOPTIMIZED(ms):       23150.0
OPTIMIZED(ms):         15500.0
SPEEDUP:                  1.49

Running MATRIX_MULTIPLY with n = 1600 loop = 1

UNOPTIMIZED(ms):       76967.0
OPTIMIZED(ms):         50950.0
SPEEDUP:                  1.51
```