Problem Statement

In this exercise, you’re supposed to use CUDA to implement the Gauss-Jordan Elimination method to solve a system of linear equations. The algorithm starts by building the augmented matrix for a system of linear equations as shown in the example below:

\[
\begin{align*}
2y + z &= 4 \\
x + y + 2z &= 6 \\
2x + y + z &= 7
\end{align*}
\]

System of Equations

\[
\begin{bmatrix}
0 & 2 & 1 & 4 \\
1 & 1 & 2 & 6 \\
2 & 1 & 1 & 7
\end{bmatrix}
\]

Augmented Matrix

The procedure then repeatedly applies elementary row operations on the augmented matrix, \([A|b]\), in order to transform \(A\) into a diagonal matrix. An elementary row operation on a matrix scales a row, swaps two rows, or subtracts a scaled version of one row from another. The set of elementary row operations for the example above are as follows:

\[
\begin{bmatrix}
0 & 2 & 1 & 4 \\
1 & 1 & 2 & 6 \\
2 & 1 & 1 & 7
\end{bmatrix}
\]

\(\rightarrow\)

\[
\begin{bmatrix}
1 & 1 & 2 & 6 \\
0 & 2 & 1 & 4 \\
2 & 1 & 1 & 7
\end{bmatrix}
\]

\(\rightarrow\)

\[
\begin{bmatrix}
1 & 1 & 2 & 6 \\
0 & 2 & 1 & 4 \\
2 & 1 & 1 & 7
\end{bmatrix}
\]

\(\rightarrow\)

\[
\begin{bmatrix}
1 & 1 & 2 & 6 \\
0 & 2 & 1 & 4 \\
-2 \times r_1 + r_3
\end{bmatrix}
\]
The last step of the algorithm divides the diagonal element and the right-hand-side element in each row by the diagonal element in that row making each diagonal element equal to one:

\[
\begin{bmatrix}
1 & 0 & \frac{4}{5} \\
0 & 2 & \frac{4}{5} \\
0 & 0 & -\frac{3}{2}
\end{bmatrix}
\begin{bmatrix}
1 \\
2 \\
0
\end{bmatrix}
\begin{bmatrix}
\frac{4}{5} \\
\frac{4}{5} \\
-\frac{3}{2}
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
1 \\
2 \\
0
\end{bmatrix}
\begin{bmatrix}
\frac{4}{5} \\
\frac{4}{5} \\
-\frac{3}{2}
\end{bmatrix}
\begin{bmatrix}
\frac{11}{5} \\
\frac{7}{5} \\
\frac{6}{5}
\end{bmatrix}
\]

Hence,

\[
x = \frac{11}{5}, \quad y = \frac{7}{5}, \quad \text{and} \quad z = \frac{6}{5}.
\]

[The description of the algorithm and the example is taken from: http://ceee.rice.edu/Books/CS/chapter2/linear44.html]

The input to your program is a randomly-generated square matrix of floating point numbers. For simplicity you might assume that the size of the matrix is a power of two (e.g., 32, 4096, etc.). The output will be the solution for the system of linear equations (if there exists any).

In order to test the correctness of your program, you must implement a sequential function which takes a copy of the input matrix as the input parameter and produces the solution for the system of linear equations. Then you can compare the output of the two functions to make sure there is no discrepancy between them. However, you may allow small differences between the results of the two correct CPU and GPU programs due to different implementations of the floating point operations in the two architectures.

You also need to evaluate the performance of your GPU program by reporting its parallel speedup, that is, the ratio of the execution time with the sequential algorithm divided by the execution time of the parallel one. This evaluation must be done for different matrix sizes (up to 8192x8192). Also, you are supposed to attempt to find the optimal number of thread blocks and number of threads in each block by trying different kernel execution configurations for your
CUDA kernel program. You can use the time measurement mechanisms that were discussed in the class to measure parallel speed up.

**Deliverables**

1. The source code, the `makefile`, and the compilation instructions. Your program should be compiled on Linux. Also, it’s recommended that you use the NVIDIA_CUDA_SDK project template as a basis for organizing your code.

2. A brief performance evaluation report of your program showing parallel speedup for different execution configurations, and different input sizes (512x512, 1024x1024, 2048x2048, 4096x4096, and 8192x8192). While the exact format of this report is up to you, one should be able to conclude from this report what would be the optimal execution configuration for each input size.