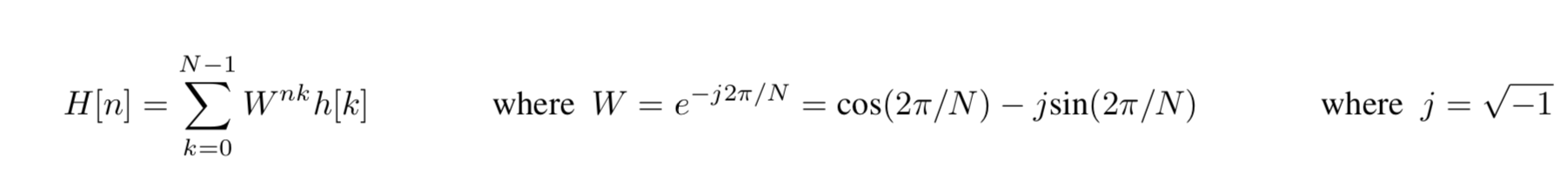
**P2 – Threaded Two-Dimensional Discrete Fourier Transform**

Given a one-dimensional array of complex or real input values of length N, the Discrete Fourier Transform (DFT) consists of an array of size *N* computed as follows:



For all equations in this document, we use the following notational conventions. *H* is the discrete-time sampled signal array. *H* is the Fourier transform array of *h*. *N* is the length of the sample array, and is always assumed to be a power of 2. *N* is an index into the *h* and *H* arrays, and always in the range of 0… (N-1). *k* is also an index into *h* and *H*, and is the summation variable when needed. *j* is the square root of negative one.

The above equation clearly requires *N*2 computations, and as *N* gets large the computation time can become excessive. There are a number of well-known approaches that reduce the computation time considerably and we may look at one in a future assignment, but for this assignment you can just use the simple double summation shown above.

Given a two-dimensional matrix of complex input values, the two-dimensional Fourier Transform can be computed with two “simple” steps. First, the one-dimensional transform is computed for each row in the matrix individually. Then a *second* one-dimensional transform is done on each *column* of the matrix individually. **Important**: The transformed values from the first step are the inputs to the second step.

With a multicore CPU and armed with the threading knowledge you now possess, it should be relatively clear how some of these steps can be done simultaneously. For example, if we are computing a 2D DFT of a 16 by 16 matrix, we could launch 16 threads where each thread would compute the DFT of a given row. Thread 0 would compute the one-dimensional DFT for row 0, Thread 1 would compute the one-dimensional DFT for row 1 etc. In a perfect world, this should run 16 times faster than if we used only one thread.

For the second step, computing the *column* values things become slightly trickier. When thread 0 computes the one-dimensional DFT for row 0, it would presumably be ready to compute the 1D DFT for column 0. However, the computed results may not be completed by the other threads. Therefore, after computing its row, the thread must wait, until all of the other threads have completed prior to starting the column DFT.

**Implementations**

NOTE: If you are using Windows, you MAY use Visual Studio 2017 for this assignment since it may make it easier to use things like threads. However, you are kind of on your own on figuring out how Visual Studio works. There should be plenty of information online. CMAKE allows for the creation of Visual Studio projects from CMAKE projects. Google is your friend.

**IF YOU ARE USING VISUAL STUDIO IT MUST BE ONE OF THE 2017 VERSIONS AND YOU SHOULD SUBMIT YOUR ENTIRE SOLUTION FOLDER. FUTHERMORE THE FINAL ZIP SHOULD BE NAMED P2\_WIN.zip**

For Mac/Linux the submission should be P2.zip or P2.tar.gz.

What you must do:

1. Implement a 2D DFT using the double summation approach in the equations above. In the starter code there exits two files on which to test: Tower.txt and Tower-Large.txt. after1d.txt shows how Tower.txt’s data should look after the row calculations have completed. after2d.txt shows how the data should look after completion of the column calculations. Save your output to a file named MyAfter2D.txt.
2. Implement a reverse DFT Algorithm and use it to restore the transformed image to the original real values for each pixel. Use SaveImageDataReal method in InputImage to save the image to a file named MyAfterInverse.txt.

3 You can use any of the approaches we have talked about in class for your threads.