Math 320 Programming Project III - Fall 2024

Fast Fourier Transform – Non-Recursive (Cooley-Tukey) Version with Bit Reversal

Please submit all project parts on the Moodle page for MAT320. You should include all necessary files to recompile, and a working executable, all in a zipped folder (one file for upload). Time-stamp determines the submit time, due by midnight on the due-date. A user interface is not required, however since more students are adopting JUCE for audio projects, it is reasonable to do this sequence of projects with JUCE also. The basic requirements remain the same: text input and output.

1. Integer bit reversal algorithm
   - input: command line arg positive integer $N = 2^k$
   - output: list of integer and binary output, each line containing: i, b_1, i_r, b_2, where:
   - i is integer value from 0 to $N - 1$
   - b_1 is binary form of i
   - i_r is integer value of bit reversal of b_1
   - b_2 is bit reversal of b_1
2. Non-Recursive Fast Fourier Transform (FFT) with bit reversal
   - input: command line arg $N = 2^k$, text file of N complex numbers
   - output: DFT of input as list of N complex numbers
3. Timing comparisons:
   - Time your implementations of the DFT, FFT recursive form, and FFT non-recursive form.
   - Use $N = 1024$ and a somewhat arbitrary set of complex numbers as input.
   - Report your timing comparisons in a text file called timing.txt (Use enough decimal places to notice the difference.) Also submit the code used to do the timing.

Notes:

1. Please follow these naming guidelines: there should be two separate programs, one called bitrev.cpp, one called fft2.cpp, and one text file called timing.txt. The code should compile on the command line with g++ and should not require any other headers or linked files.

2. The bit reversal algorithm should use the following binary operators: $>>=$ (shift right equals), & (bitwise AND), $\&=$ (bitwise AND equals), $|=$ (bitwise OR equals), $\sim$ (bitwise inversion, or one's complement).

3. The bit reversal is accomplished by building the integers from 0 to $N-1$, each one from the previous case, but always backwards in binary. This means that we simply need to know how to add one to a backwards binary number to get the next backwards binary number. To add one to a regular binary number, we can check the rightmost bit. If it is a zero, we simply change it to a one and we are finished. If it is a one, we change it to a zero and proceed to the left. We continue changing any ones to zeros until we encounter a zero and then change to a one, which performs the carry. For backwards binary we do the same thing starting on the left and proceeding right. To check each digit we use a MASK which has values $MASK >>= 1$, which starts at $N >>= 1$. This has the correct number of bits to represent the integers from 0 to $N-1$ (forwards or backwards). Now suppose the incoming backwards binary number is B. We check digits using $B\&(MASK >>= 1)$. While this value is one, we change a one to a zero using using $\&= \sim MASK$. When the condition fails, we detect a zero, and simply change it to a one with $|= MASK$.

4. For the FFT we follow the same input as the previous FFT. This time we do not use a recursive function. Rather, we set up the array so that we can add adjacent entries and sum them up in a double loop which is the Cooley-Tukey FFT with bit reversal. This is the algorithm described in section 7 of chapter 8, on pages 164-167 of the text book. The bit reversals apply to indices of the array, so that the array elements are reorganized according to the reversed indices. The code is described below figure 7.2, where the merge step follows equation 6.9. (Also, see the extra notes on FFT with bit reversal on the web site.)