

NOTE: In order to get an accurate sampling rate, I iterated infinitely over the array X (as I ran out of space copying it). The first iteration results are printed, and ther est are left out.

I broke up the FFT function into several smaller functions to better identify bottlenecks.   
  
The main function sets up the arrays and variabled needed for execution.

**reversal** - performs the reversal before calcfft

**calcfft** - wraps the fft\_nest3 function which contains the internal 2 for loops wrapped by the for loop which iterates over all elements.

**fft\_nest3** - performs the following calculations and calls the inner nested loop:



**fft\_nest2** - performs the inner most loop calculations



The inner most for loop is going to have the most iterations overall and therefore would be a good place for PL optimization.

As seen here:



The fft\_nest2 function has the most compution excluding all other function calls. (Column 1)

Whereas we see the inclusive (column 2) combines the function calls within a function. Thus why the "inclusive" (column 2) values for fft\_nest3 are so high.

Overall, **fft\_nest2**, would be a good place to look for optimizations.