Next fast FFT size

Peter L. Søndergaard

Abstract

This note explains the rationale behind the `nextfastfft` function in LTFAT, and backs this by some timings done on a standard desktop PC.

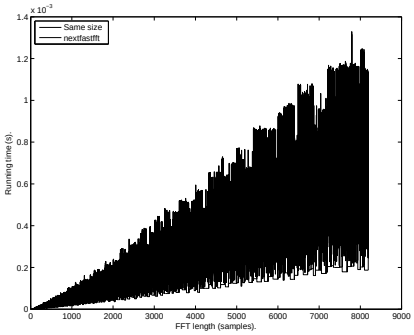
1 Rationale

The basic form of the FFT algorithm works by splitting the problem into subproblems, computing the FFT of each subproblem, and then combining the result. This is the classical divide-and-conqueror approach [1], which gives the FFT an $\mathcal{O}(N \log N)$ running time. The success of this scheme depends crucially on the size of the input: it should be possible to split the input into pieces of equal size. The more possible splittings, the fewer flops the algorithms use. If the input size is a large prime number, the traditional approach will not work, but there do exists fast algorithms even for this case [3]. For this reason, sizes of FFTs are often choosen to be powers of 2, because this leads to the most efficient algorithm. In many applications, the actual FFT size is not important, as long as it is longer than the size of the input. Therefore Matlab and Octave offers the `nextpow2` function for finding the next larger problem size that is a power of 2.

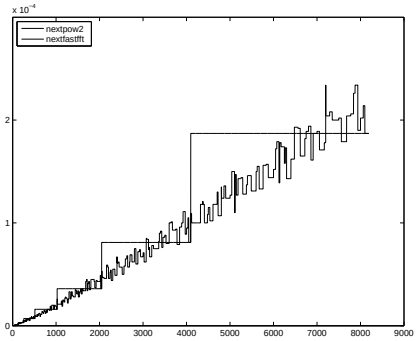
Matlab and Octave use both use the "Fastest Fourier Transform in the West" (FFTW, see [2]), which is an open source efficiant FFT implementation. The running time of FFTW for a range of problem sizes is shown on Figure 1a. The large variation of the running time between different problem size is apparent.

A modern FFT implementation like the one offered by FFTW contains special code to handle splittings not only of size 2, but also for sizes 3, 5 and 7. This makes the `nextpow2` method seem overly pessimistic, as it may potentially double the problem size, where a faster and smaller problem could be solved instead. For this reason, LTFAT [4] includes the `nextfastfft` function, which will return the next higher number that is solely comprised of prime factors 2, 3, 5 and 7. Figure 1b shows the running time of FFTW if the `nextpow2` method or the `nextfastfft` method is used. Some important conclusions can immidiatly be seen:

1. Using either method gives a guaranteed low running time, which is much lower than the running time for the "bad" cases.



(a) The running time for all FFT-sizes from 1 to 10.000. The results are averages of 1000 runs.



(b) The running time for all FFT-sizes from 1 to 10.000.

Figure 1: Timings of FFTW done on a an Intel Core2 Duo 1.6 GHz cpu.

2. The `nextfastfft` method is usually the fastest if the `nextpow2` method would yield a large increase in problem size. This reverses if the problem size is slightly below a factor of 2.

3. The running time of the problem sizes predicted by `nextfastfft` is not increasing monotonically, sometimes a larger problem size executes faster. However, the difference is small compared to the variability among all problem sizes.

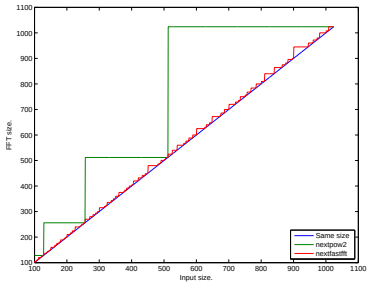
2 Implementation

The current implementation uses a table lookup to find the exact solution for all problem sizes smaller than $2^{20} = 1048576$. For problem sizes larger than this number, the problem size is first reduced by factors of 2, until it is smaller than 2^{20}. The table is computed only the first time the function is run. For problem sizes smaller than 2^{20} the table contains 1286 entries, and it is searched by a simple bisection method.

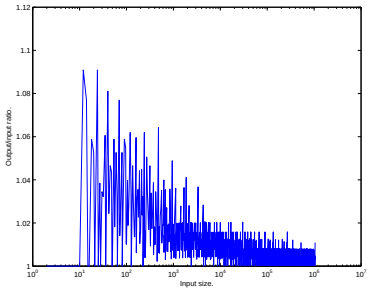
Figure 2a shows the sizes returned by `nextfastfft` compared to using `nextpow2`. As can be seen, the `nextfastfft` approach gives FFT sizes that are much closer to the input size. Figure 2b shows the highest output/input ratio for varying input sizes. As can be seen, the efficiency is better for larger input values, where the output size is at most a few percent larger than the input size.

Acknowledgment

The author would like to thank Johan Sebastian Rosenkilde Nielsen and Tom Høholdt from the DTU department of Mathematics for a good discussion on



(a) Problem sizes predicted by `nextpow2` compared to `nextfastfft`.



(b) The highest output/input ratio for varying input sizes of `nextfastfft`.

Figure 2: Predicted sizes of nextfastfft (this is the output from the `demo_nextfastfft`).

possible algorithms for `nextfastfft`.

References

[1] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*, 19(90):297–301, 1965.

[2] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[3] C. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.

[4] P. L. Søndergaard, B. Torrésani, and P. Balazs. The Linear Time Frequency Analysis Toolbox. *International Journal of Wavelets, Multiresolution Analysis and Information Processing*, submitted, 2011.