# Experiences with real-valued FFT algorithms

László Tóth

Research Group on Artificial Intelligence
Hungarian Academy of Sciences,
Aradi vértanuk tere 1, H-6720 Szeged, Hungary
Phone: (36) +(62) 454139, Fax: (36) +(62) 312508
e-mail: tothl@inf.u-szeged.hu,

**Abstract.** The classic FFT algorithm of Cooley and Tukey works on complex data. However, in real-life applications the input array is in most cases real-valued. This allows a reduction in the number of arithmetic operations, by at least an order of two. In every year several articles are published saying that "oops, I've found two unnecessary operations in my routine, so now it's again faster than yours". Runtimes, of course, also depend on criteria like what processor the algorithm is run on, must it be in-place or not, etc. So we decided to implement and test some of these algorithms to find which is the fastest for our speech processing aims. We found the split-radix algorithm of Sorensen to be the fastest. The Bruun-FFT was also very close to it, but since it cannot be implemented in-place, we chose the former to be used in our applications.

## 1 Introduction

It is well known that when dealing with real input, the computation complexity of the FFT algorithm can be approximately halved by exploiting the symmetry properties of the Fourier-transform. Several transform algorithms were proposed for this case; these, of course, all use $O(n * logn)$ arithmetic operations, and the differences come only from constant multiplicating factors and linear additive factors (e.g. $\frac{1}{2}n * logn - 2n$). However, in applications even these slight differences are of importance. Since on today's computers a data fetch instruction, an integer operation and even a floating-point operation require comparable processor time, we didn't see much sense in counting the number of multiplications and additions as is usual in numerical mathematics. Rather we simply measured the run times of the algorithms (which method is also questionable - see later) on a PC with a Pentium 100Mhz processor. The algorithms compared were the so-called "packing" method, the Bruun-FFT and the Radix-2 and split-radix versions of Sorensen's algorithm.

## 2 Aspects of Comparison

To make the algorithms comparable we first have to clearly specify what we expect from them. Since we use them for speech processing, it was quite clear that we were interested only in 1-dimensional FFT's, and it is enough if they work on arrays with lengths of a power of two.

A less obvious choice was to allow the algorithms to use lookup-tables or not. We chose the latter case, only because the algorithms look 'clearer' this way (some measurements showed that a sine call tooks about twice the time than reading it form a lookup table - on a Pentium). This way the routines had to be optimized so that they have the less possible sine and cosine calculations (and also scrambling index calculations in the case of the Bruun-FFT). Obviously, versions allowing lookup-tables might have resulted in different run-times.

Another important issue is whether the algorithm can be implemented in-place. The complex FFT gives the result in a so-called bit-reversed order; but luckily this is a very special permutation which can be "unscrambled" in-place. Similar problems arise in the case of the real-valued transforms: the packing and Sorensen algorithms use the same unscrambler as the complex FFT. Albeit they give their results in different orders, these orders are quite simple and does not cause problems for the applications. The Bruun-FFT, however, returns the values in a rather tricky order, and the unscrambling cannot be done in-place.

## 3 Short Overview of the Algorithms

### 3.1 The Packing Algorithm

The most classic method to calculate the DFT of a real-valued array of length N is to "pack" every second value into the imaginary part of the preceeding value. This way we get a complex array of length N/2. After transforming it by some complex FFT algorithm[1], the transform of the original series can be calculated by some additional O(N) operations[2]. Although the transform of N (in this case real) values would be N complex values, because of the Hermitian symmetry of the result only N/2+1 values need to be computed. Since the first and last of these are always purely real, the result fits into the input array, and so the FFT can be done in-place. The usual data order is:

Input: re(0),re(1),...,re(N-1)
Output: re(0),re(N/2),re(1),im(1),re(2),im(2),...,re(N/2-1),im(N/2-1)

### 3.2 The Bruun Algorithm

Georg Bruun noticed[5] that the FFT corresponds to filtering the data with a rather special filterbank. If we factor these filters into second-order ones, as is quite usual in DSP, then these small filters need only one real multiplication (if the input is real) and two additions. Also, with appropriate grouping of these filters only $O(n * logn)$ of them needs to be computed. This leads to a very efficient implementation of the DFT. Unfortunately, the output comes out in a rather awkward order, and the unscrambling cannot be done in-place (at least, we couldn't solve it and none of the references[5][3] deals with the question). However, in applications where unscrambling is not necessary (e.g. filtering) the

Bruun-FFT seems to be a good alternative (supposed that a good inverse FFT version can be made for it, which we didn't try).

### 3.3 Sorensen's Algorithm

Sorensen's approach[4] was to modify the complex FFT by omitting the operations that are unnecessary due to the symmetry properties in the case of real data. Both the radix-2 and the split-radix versions of the algorithm in their article were implemented; the output order of these algorithms is the following:

Input: re(0),re(1),...,re(N-1)

Output: re(0),re(1),...,re(N/2),im(N/2-1),...,im(1)

Although this order is not the same as in the case of the packing algorithm, it can quite easily be used in the applications, so it cannot be considered as a "scrambled" order (as in the case of the Bruun-FFT).

## 4 Run-time Measurements

The algorithms (see Appendix) were compiled with Microsoft Visual C++ 4.0 under Windows 95. The run-time measurements were made on a PC with a Pentium 100Mhz processor. To get more reliable results all of the algorithms were run in a loop (500 or 100 runs); since the algorithms divide the results by the array length, the array were initialized with very big (random) values to avoid underflow. The following table contains the measured run-times (normalized to one run) in 1000 milliseconds:

| array size | Packing | Sorensen Radix2 | Bruun* | Sorensen Split |
|---|---|---|---|---|
| 64 | 692 | 440 | 274/330 | 248 |
| 128 | 1534 | 1072 | 560/686 | 566 |
| 256 | 3624 | 2692 | 1402/1812 | 1344 |
| 512 | 8014 | 6208 | 2938/3572 | 2856 |
| 1024 | 17270 | 13460 | 5760/7140 | 5500 |
| 4096 | 82470 | 68810 | 27050/32130 | 26640 |
| 16384 | 391960 | 335100 | 127030/148740 | 124970 |

*without and with unscrambling

## 5 Further Possibilities of Speed-up

The above measurements were repeated several times, and the results appeared to be quite consistent, only a few percentage differences occured. However, only slight and seemingly irrevelant changes in the source caused in many cases run-time increase or decrease on the order of 10 percentages. This means that measurements of this kind can be considered only an approximate evaluation of the algorithms. Quite sure, for example, that implementing the algorithms optimized for Pentium (maybe in assembler) could result in even more serious speed increase.

## 6 Conclusion

The conclusion of our measurements is, quite in accordance with[6], that the commonly used "packing" method is very slow and should be avoided. The radix-2 version of Sorensen's algorithm is only slightly better, while the split-radix version is much faster. The Bruun-algorithm is practically as fast as the split-radix, but it has the big drawback of returning the result in a scrambled order, which cannot be unscrambled in-place. It seems that the generally neglected Bruun-algorithm is worth giving a try. Maybe a split-radix style version could be made of it, which could be even more faster (see the case of the Sorensen algorithm!).

## References


1. L. R. Rabiner - B. Gold: Theory and Application of Digital Signal Processing, Prentice Hall, 1975
2. E. O. Brigham: The Fast Fourier Transform, Prentice Hall, ???
3. H. J. Nussbaumer: Fast Fourier Transform and Convolution Algorithms, Springer-Verlag, 1982
4. H. V. Sorensen et al: Real-Valued Fast Fourier Transform Algorithms, IEEE Trans. ASSP, ASSP-35, No.6, June 1987
5. G. Bruun: z-Transform DFT Filters and FFT's, IEEE Trans. ASSP, ASSP-26, No.1, February 1978
6. P. R. Uniyal: Transforming Real-Valued Sequences: Fast Fourier versus Fast Hartley Transform Algorithms IEEE Trans. SP, Vol. 42, No. 11, November 1994