FFTW_ESTIMATE vs FFTW_MEASURE

N.B. The following all assumes the standard interface to be used, rather than the GURU interface.

FFTW computes FFTs in three steps:

- planning
- execution
- cleaning up

Execution is simply FFTW_PLAN_3D(planpointer) and has no further options. Cleaning up is simply releasing the plan by FFTW_DESTROY_PLAN(planpointer), which is also straightforward.

However, the planning step is more involved. As FFTW contains numerous methods to compute an FFT, it is advantageous to pick the method that is the fastest upon execution. The chosen method (over which no direct control is possible) is stored in a pointer to an INTEGER*8 number containing all the options. After a plan has been decided on, this plan can be stored so that it can be reused when the same FFT is taken again to save on the planning time.

Planning time may vary strongly between different FFTs due to the size of the matrix and the planning method chosen. There are four different planning strategies, ordered by speed, labeled FFTW_ESTIMATE, _MEASURE, _PATIENT and _EXHAUSTIVE.

FFTW_ESTIMATE only looks at the size of the matrix and uses predetermined semantics to plan the FFT. Even though the planning stage is performed instantaneous, the resulting plan is suboptimal since no memory location, ordering, fragmentation and stride (a combination of vector length and their location in RAM memory with respect to cache sizes, details are hard to come by) is taken into account.

FFTW_MEASURE and _PATIENT do take those factors in account, and simply run a number of methods included in FFTW to determine which method yields the fastest Fourier transform. As this is determined by a one-shot experiment, it implies that on different runs, different methods can be selected in the planning stage. Therefore, in consecutive runs, different results of taking an FFT can be obtained. This difference is usually the machine precision and in the order of 10^{-15}. This claim is also written in the FFTW manual. The difference between _PATIENT and _MEASURE is simply the number of methods tested.

FFTW_EXHAUSTIVE basically does the same trick, although it also includes methods that are, according to the manual, “not likely to yield a faster transformation”.

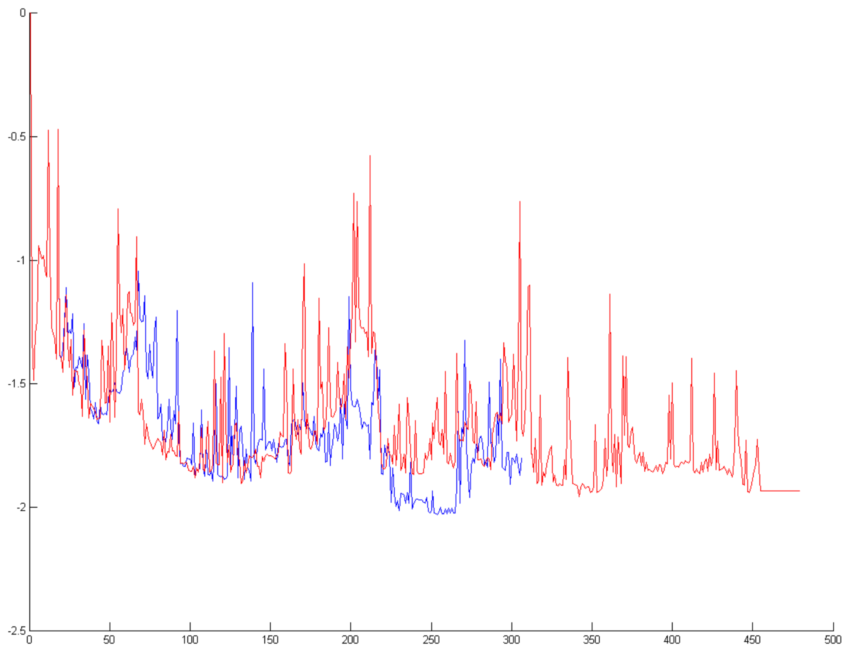
As an example, in the table below the plan- and computationtime of an FFT of a matrix of 408x532x64 elements are shown for the four planning strategies.

Planning strategy		**Wallclock time (s)**
FFTW_ESTIMATE	Planning stage	0
	Actual FFT	**4.52**
FFTW_MEASURE	Planning stage	22.0
	Actual FFT	3.82
FFTW_PATIENT	Planning stage	139.6
	Actual FFT	**3.57**
FFTW_EXHAUSTIVE	Planning stage	456.1
	Actual FFT	3.57

Note that indeed _EXHAUSTIVE did not yield a faster FFT, and that the planning stage can be quite time-consuming. However, almost a second can be gained in the actual computation. So, from a certain number of transformations, it becomes advantageous to reuse the plan.

However, this is not the complete story. In order to reuse an FFTW-plan, care has to be taken that the memory location of the matrix being transformed is *exactly* the same in the next transformation. This requires an "FFT-matrix" that is used by *all* functions and subroutines performing an FFT which is on no occasion deallocated. Besides the more complicated programming, this requires the matrix to be present in memory at all times, which increases the memory footprint of the software.

Furthermore, as was already previously mentioned, different FFT methods yield slightly different results. As shown in the figure below, this leads to (in the case of an iterative forward scattering problem) different convergence behaviour. In blue the remaining residual norm after N iterations is shown when FFTW_ESTIMATE is used for the FFT's, in red the same norm for the same problem though this time using FFTW_MEASURE. Both simulations ran simultaneously for the same amount of time, and the speedup of _MEASURE is clear. Upto 18 iterations the results are identical.



However, after more iterations the red line (quicker simulations) in general yields higher residual norms, and appears to flatten after about 450 iterations for this particular problem. Although a single iteration is considerably faster, more iterations are required to yield the same error, and in the end more computation time.

The red line changes a bit if the program is resubmitted, indicating that FFTW_MEASURE selected a different FFT method. The general conclusion, slower convergence, in most cases remained the same, however. Furthermore, since no direct control over the FFT method is possible, there is no way of predicting whether the used method is faster or slower in convergence – it may be different in the next run.

Summing up, FFTs can be sped up by using a different planning strategy, but the results are in the end slower and above all the convergence behaviour cannot be predicted. Hence I recommend using FFTW_ESTIMATE and just wait a bit longer per iteration.