Comparison of some FFT libraries in C/C++

Philippe Gambron [*]

Science and Technology Facilities Council, Hartree Centre, Rutherford Appleton Laboratory, Harwell Campus, Harwell Oxford, OX11 0QZ, United Kingdom

Abstract

We compare the performance of several libraries computing FFTs called from C/C++ code and running on a single node.

1 Introduction

Some applications require the computation of the fast Fourier transform (FFT) of large datasets. In such a case, the efficiency of that step can become of critical importance. In this report, we compare the performance obtained, on a single node, with several libraries that can be called from C or C++.

2 Overview of the chosen libraries

We consider the following libraries computing FFTs: FFTW [1], MKL [2], GSL [3] and FFTPACK [4] (Table 1). They can perform complex transforms, real-to-half-complex ones (and conversely) as well as, in the case of FFTW, real-to-real transforms when the signal is odd or even. The half-complex output consists in half as many complex values as there were points in the signal, taking advantage of the hermiticity of the Fourier transform of a real function.

The GSL and FFTPACK libraries only work in one dimension[1]. On the contrary, FFTW and MKL can compute FFTs in several dimensions. They are also capable of working in a parallel way, using multithreading and MPI. The purpose of this work is to assess their performance on a single node. As a consequence, we will only study the effect of multihtreading.

[*] `philippe.gambron@stfc.ac.uk`

[1] FFTPACK can compute FFTs in several dimensions but it is written in FORTRAN and all the C or C++ wrappers we have found only allowed one-dimensional transforms.

	Type	Dim.	Radices	Parallelism	Licence
FFTW	R→H, C→C, H→R, R(odd/even)→R	Any	2, 3, 5, 7, 11, 13 + any with code generator	Multithreading, MPI	GPL v2
MKL	R→H, C→C, H→C	Any		Multithreading, MPI	Proprietary
GSL	R→H, C→C, H→R	1	2, 3, 5, 6, 7	-	GPL v3
FFTPACK (CASA wrapper)	R→H, C→C, H→R	1		-	GPL v2

Table 1: Overview of the FFT libraries considered. R stands for real, C, for complex, and HC, for half-complex.

3 Benchmark

The benchmark [6] consists in calculating the FFT of a series of volumes, in 1, 2 or 3 dimensions, of real or complex values. The purpose was to mimick a problem submitted to us by the CCP PET-MR collaboration [7], within the Software Outlook initiative [8], who needed to take the transform of series of square complex images. This example is depicted in Fig. 1. For our more general test, the 2-dimensional slices could be replaced by a line or a rectangular cuboid. We used 32 images, which was a typical value used by that collaboration, for the tests that were representative of their requirement. For the other runs, we always processed a single image.

We used a number of points varying from $\sim 10^3$ to $\sim 10^7$. The domains had sides of equal lengths (square or cubic) or were flattened (rectangle or cuboid). These dimensions could be powers of 2, products of powers of small integers (2, 3, 5 and 7) or prime numbers. We used a single thread in all cases except when we assessed the effect of multithreading. In that case, the number of threads was increased till 24.

The values appearing in the graphs are the execution time averaged over 10 runs. A certain number of bumps or slight unexpected features are apparent. However they were consistently repeated in all our measurements. This is also confirmed by the standard deviation of our results which was always quite small, of the order of a few percents of the average value. We chose not to display error bars on our graphs since they were so small that they were barely visible.

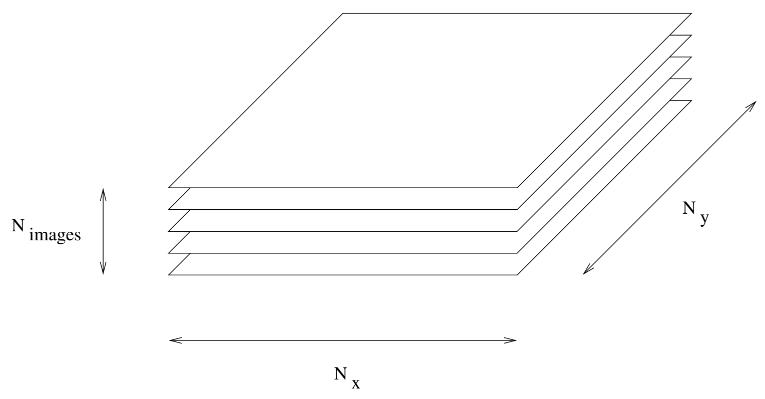


Figure 1: The benchmark consists in taking the FFT of several images. Each of them is made of real or complex values and can be a simple line, a rectangle or a cuboid.

4 Set up

Our measurements were carried out on Archer [9]. The modules `gcc/7.2.0` and `intel/17.0.3.191` were loaded. In order to take advantage of multi-threading, the environment variable `KMP_AFFINITY` must be set to `disabled`. We used our own version of Boost, FFTW and GSL because the version of Boost present on the system was lacking certain libraries and we wanted to use the most recent versions of FFTW and GSL.

Our benchmark was done in C++ using doubles. More precisely, we used version 3.3.8 of FFTW, version 17.0.3 of MKL and version 2.5 of GSL. The benchmark was compiled with the flags `-std=c++1z -O3 -fopenmp -lm -lfftw3 -lfftw3_threads -lgslcblas -lgsl -lboost_system -lboost_chrono -liomp5 -lmkl_core -lmkl_intel_thread -lmkl_intel_lp64 -lcasa_scimath -llapack`.

Since FFTPACK was written in FORTRAN, we have resorted to the C++ wrapper provided by CASA, the radioastronomy package [5]. This is straightforward on Debian-like systems, as the one used by CCP PET-MR in their virtual environment, where CASA can be installed using the package manager. However, including the headers and linking with the libraries is not possible neither supported with the version of CASA distributed by the National Radio Astronomy Observatory. As a consequence, this was quite difficult to set up on Archer and required to use the libraries provided by an old version of Debian, to match those available on the system.

5 Performance

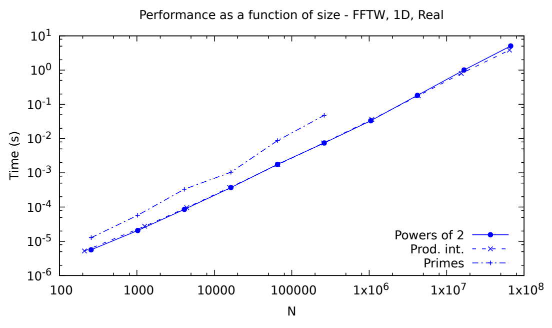
We first compare the performance of each library, in one dimension (Section 7), for numbers of points consisting of powers of 2, products of powers of small integers and prime numbers. We do so for real and complex values and with FFTW (Fig. 2a and 2b), MKL (Fig. 3a and 3b), GSL (Fig. 4a and 4b) and FFTPACK (Fig. 5a and 5b). We then compare the different libraries for powers of 2 (Fig. 6a and 6b) and in general (Fig. 7a and 7b).

Finally we repeat those measurements in more dimensions (Sections 5.2 and 5.3), with a domain that is a square or a cube, but only FFTW and MKL work in these situations (Fig. 8a - 15b).

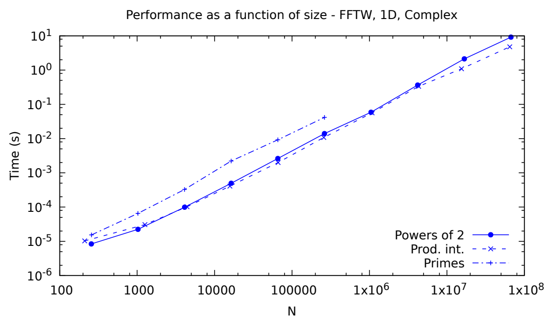
As expected, for all the libraries, the performance is worse with primes than with powers of 2 or products of powers of small integers. According to our measurements, the MKL library tends to be the fastest. We also notice that the performance with primes is significantly worse, with real numbers, in the cases of GSL and FFTPACK.

5.1 1 dimension

5.1.1 FFTW



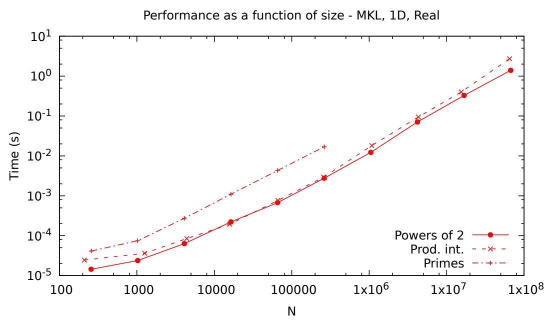
(a) Real



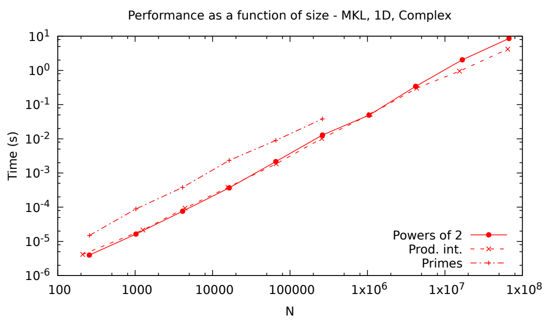
(b) Complex

Figure 2: Execution time as a function of the number of points (1 dimension, FFTW, 1 thread)

5.1.2 MKL



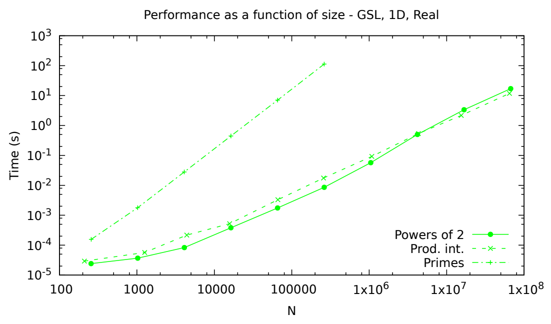
(a) Real



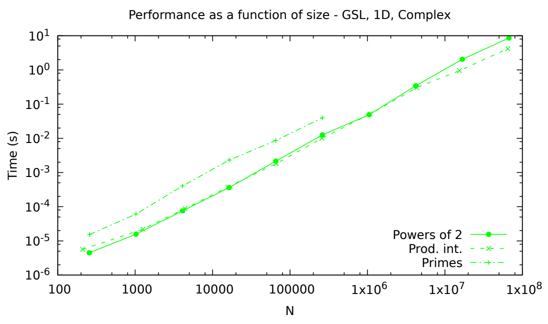
(b) Complex

Figure 3: Execution time as a function of the number of points (1 dimension, MKL, 1 thread)

5.1.3 GSL



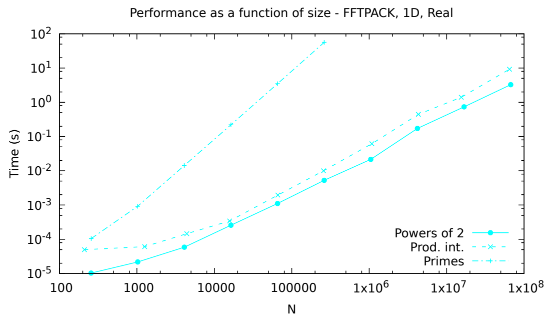
(a) Real



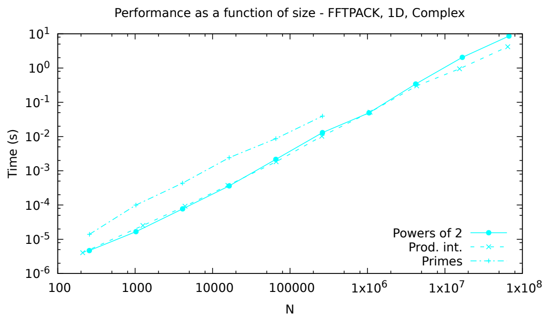
(b) Complex

Figure 4: Execution time as a function of the number of points (1 dimension, GSL, 1 thread)

5.1.4 FFTPACK



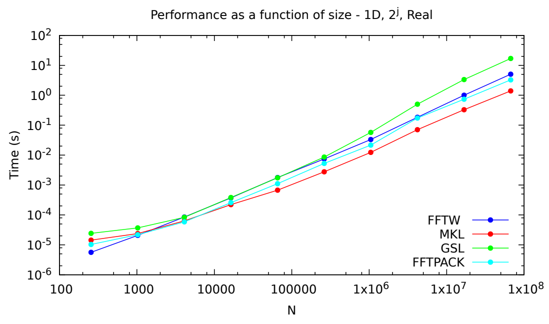
(a) Real



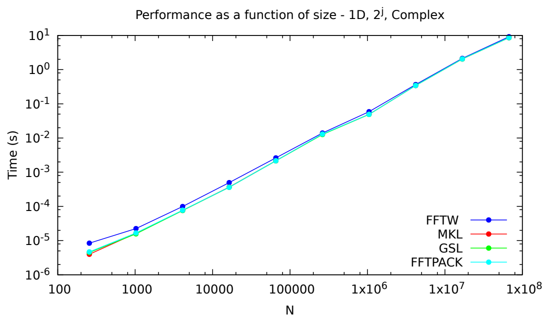
(b) Complex

Figure 5: Execution time as a function of the number of points (1 dimension, FFTPACK, 1 thread)

5.1.5 All the libraries - powers of 2



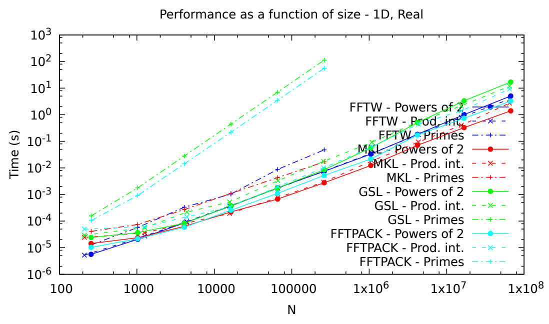
(a) Real



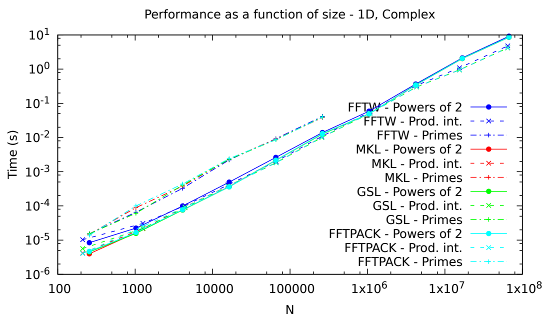
(b) Complex

Figure 6: Execution time as a function of the number of points (1 dimension, powers of 2, 1 thread)

5.1.6 All the libraries



(a) Real

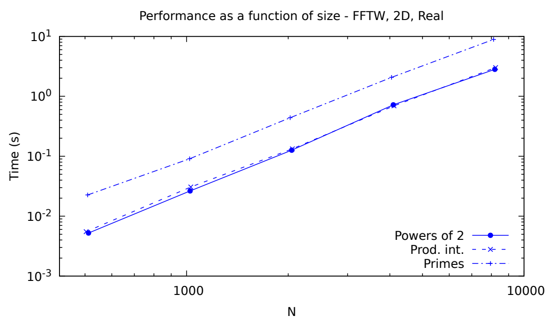


(b) Complex

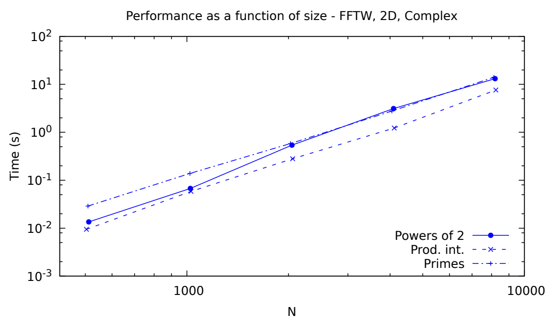
Figure 7: Execution time as a function of the number of points (1 dimension, 1 thread)

5.2 2 dimensions

5.2.1 FFTW



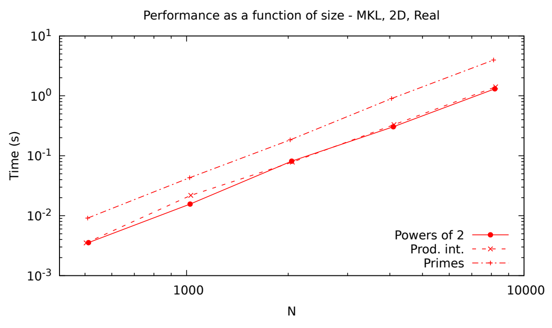
(a) Real



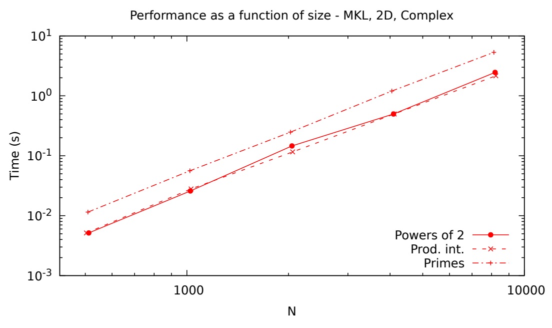
(b) Complex

Figure 8: Execution time as a function of square side
(2 dimensions, FFTW, 1 thread)

5.2.2 MKL



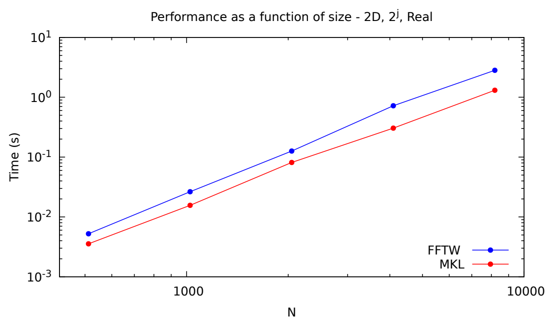
(a) Real



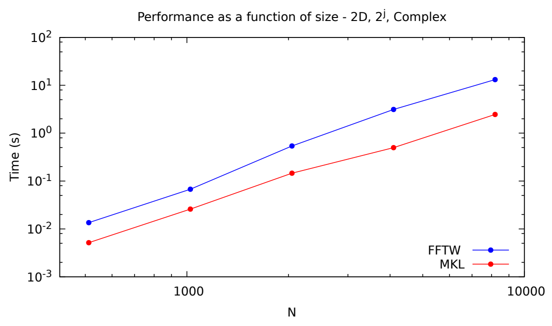
(b) Complex

Figure 9: Execution time as a function of square side
(2 dimensions, MKL, 1 thread)

5.2.3 All the libraries - powers of 2



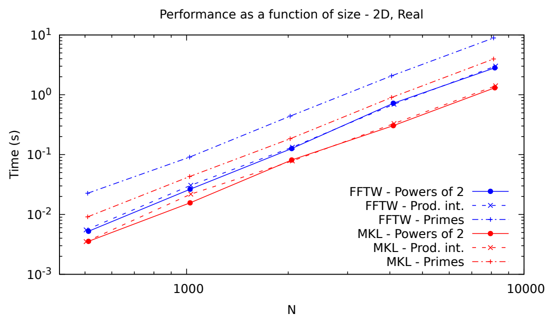
(a) Real



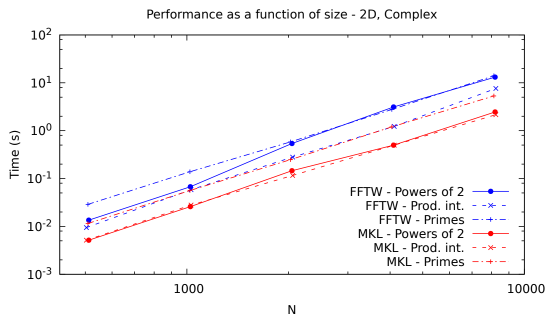
(b) Complex

Figure 10: Execution time as a function of square side
(2 dimensions, powers of 2, 1 thread)

5.2.4 All the libraries



(a) Real

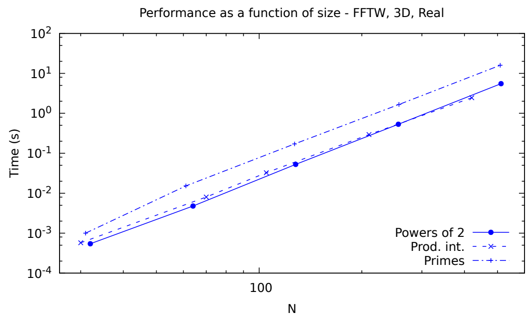


(b) Complex

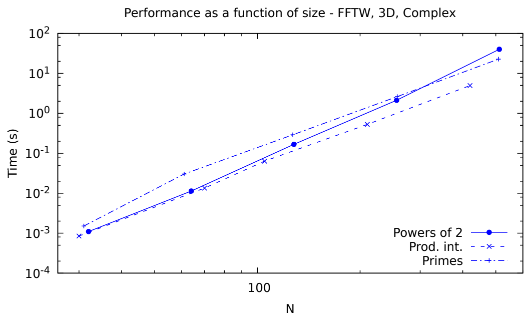
Figure 11: Execution time as a function of square side
(2 dimensions, 1 thread)

5.3 3 dimensions

5.3.1 FFTW



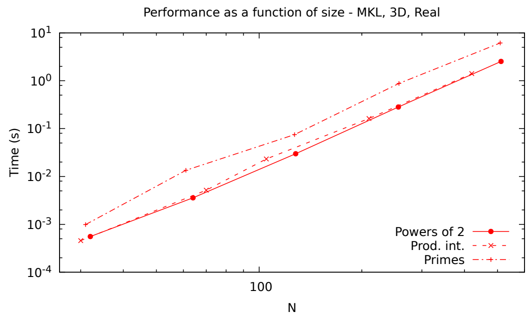
(a) Real



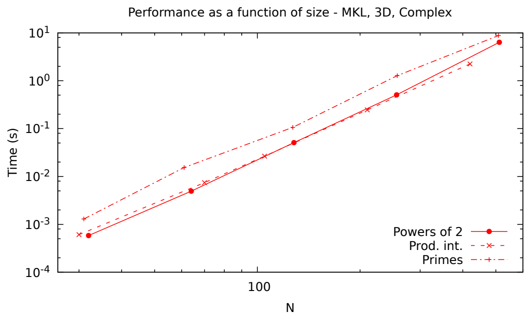
(b) Complex

Figure 12: Execution time as a function of cube side
(3 dimensions, FFTW, 1 thread)

5.3.2 MKL



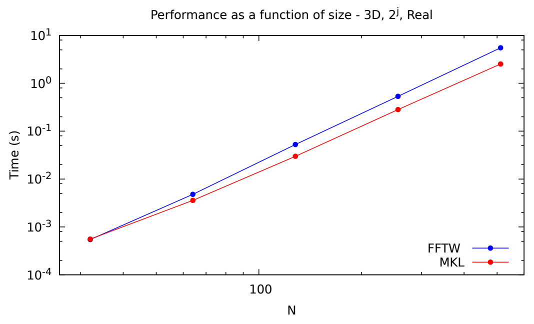
(a) Real



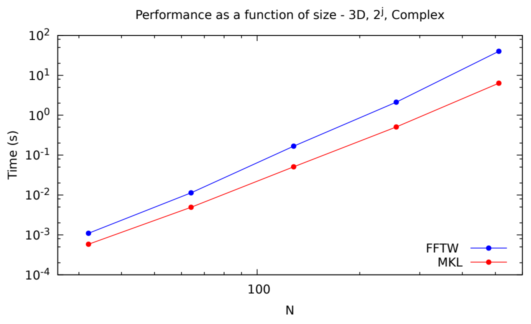
(b) Complex

Figure 13: Execution time as a function of cube side
(3 dimensions, MKL, 1 thread)

5.3.3 All the libraries - powers of 2



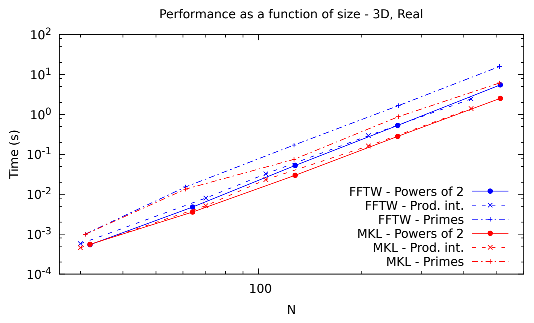
(a) Real



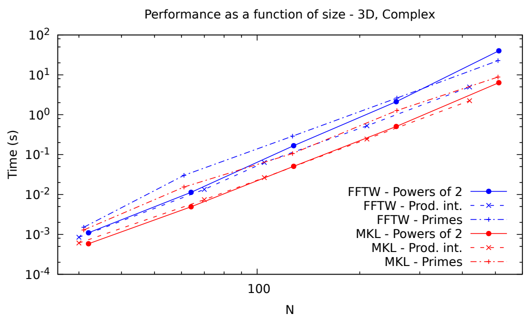
(b) Complex

Figure 14: Execution time as a function of cube side
(3 dimensions, powers of 2, 1 thread)

5.3.4 All the libraries



(a) Real

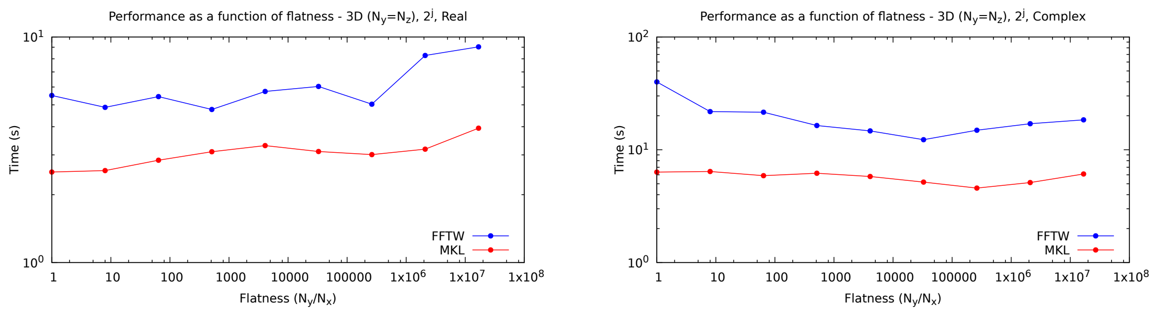


(b) Complex

Figure 15: Execution time as a function of cube side
(3 dimensions, 1 thread)

5.4 Flatness

These experiments were carried out with domains whose dimensions were the same in all directions, that is, squares or cubes. We have observed that there is no clear trend in the way the flatness of the domain affects the performance. Fig. 16 shows the performance obtained, in 3 dimensions, for a cuboid containing 512^3 points whose sides in the y and z directions have identical lengths. We hade defined the flatness as the ratio between the number of points in the y and x directions ($N_y/N_x,\ N_y = N_z$). We varied the proportions of the volume from $512 : 512 : 512$ till $2^{24} : 2 : 2$.



(a) Real (b) Complex

Figure 16: Execution time as a function of the flatness of the domain (N_y/N_x), for $N_y = N_z$ (3 dimensions, powers of 2, 1 thread)

6 Requirements from the CCP

We have also benchmarked the FFT libraries in a situation corresponding to a problem encountered by the CCP/PET-MR collaboration, by computing the transform of a series of 32 square complex images whose side consists of 256 points as well as the closest prime number (257) and product of powers of small integers ($2^2 3^2 7 = 252$). As in the other cases, the MKL library performs better than FFTW. None of the other libraries we have considered could be used in this situation. GSL works only in one dimension. The FORTRAN version of FFTPACK does not share that limitation. However all the C/C++ wrappers we have found offer only one-dimensional transforms. Finally FFTPACK is distributed with a licence that is less constraining than the GPL but, nevertheless, the C++ wrapper provided by CASA that we have used is distributed under the GPL v2.

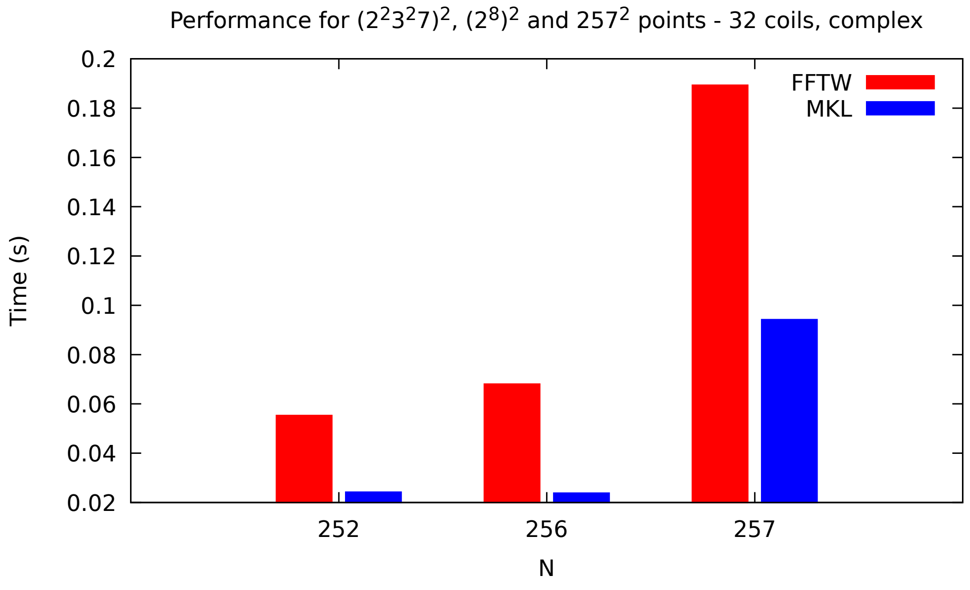


Figure 17: Execution time for the transform of 32 complex square images as a function of square side (2 dimensions, complex, 1 thread)

7 Acknowledgements

This work was funded by under grant .

References

[1] Frigo, Matteo and Johnson, Steven G., *The Design and Implementation of FFTW3*, Proceedings of the IEEE, 2005, 93, 2, 216–231, Special issue on “Program Generation, Optimization, and Platform Adaptation”

[2] https://software.intel.com/en-us/mkl

[3] M. Galassi et al, *GNU Scientific Library Reference Manual (3rd Ed.)*, ISBN 0954612078

[4] P.N. Swarztrauber, *Vectorizing the FFTs*, Parallel Computations (G. Rodrigue, ed.), Academic Press, 1982, pp. 51–83.

[5] McMullin, J. P., Waters, B., Schiebel, D., Young, W., Golap, K., *Astronomical Data Analysis Software and Systems XVI*, ASP Conf. Ser. 376, ed. R. A. Shaw, F. Hill, D. J. Bell (San Francisco, CA: ASP), 127

[6] git@github.com:SoftwareOutlook/FFTC.git

[7] https://www.ccppetmr.ac.uk/

[8] https://www.softwareoutlook.ac.uk/

[9] https://www.archer.ac.uk/