Software for Science, power monitoring The algorithm

Maurizio Giannattasio, Stefan Schokker, Bram de Wit

January 12, 2020

ASTRON
Netherlands Institute for Radio Astronomy

Abstract

In this paper we will explore what algorithm we should pick to test on our platform. The algorithm should be relevant to radio astronomy and provide an interesting study case.

Contents

1 Algorithms in radio astronomy

At first we wanted to know what algorithms were relevant to our project, since we are doing this project for Astron[1], the algorithm we are going to use should have some relevance in radio astronomy. After some searching we came up with a few algorithms that are used in that field.

- Machine learning (Artificial intelligence)
- Auto tuning
- CLEAN (image deconvolution)
- FFT (Fast Fourier Transform)

1.1 Machine learning

Machine learning can be used in a variety of ways, and thus it also has possible applications in radio astronomy. In a paper by Cornelis Johannes Wolfaardt[2] the possibility to use machine learning to automatically classify cases of radio frequency interference is explored.

Or in another paper by E.M. Howard[3] the problem of large amounts of data that needs processing is tackled. Datasets being produced by new telescopes like the SKA[1] are reaching terabytes of data and in the future will get up to petabytes. Analysing all that data would be a huge task for humans, machine learning might be a solution there.

1.2 Auto tuning

We found a rather interesting paper that explores using autotuning and many core accelerators to improve efficiency of other algorithms[4]. Autotuning is in essence a technique to set your parameters to their best possible configuration. And therefore can be used to set the parameters of an algorithm. In this paper they find that using autotune in combination with many core accelerators can lead to a significant boost in performance.

[1]Square Kilometre Array

1.3 CLEAN

The clean algorithm is an image deconvolution algorithm first described by J.A. Högbom in 1974.[5] Images in radio astronomy are often spread out a bit making single points look like blurs. The clean algorithm fixes this by taking the brightest point and subtracting a small portion of that brightness over the entire image. It does this repeatedly until the image looks "clean".(simplified explanation)

1.4 Fast Fourier Transform

The Fast Fourier Transform(FFT) is a mathematical transformation to a signal showing the frequencies most present in a certain signal. It it used for e.g. pulsar detection, image processing, analysing the frequency spectrum and more. The FFT is one of the most fundamental algorithms for data processing in radio astronomy.

2 Final decision

We have decided to go for the Fast Fourier Transformation, since most of the other algorithms use images that are usually already processed by an FFT. The CLEAN algorithm uses FFT images, and the machine learning in turn uses cleaned images. The FFT will provide an interesting study case, as it is relevant in radio astronomy, and still feasible to set up on our platform within the limited duration of this project.

In the next part of this paper we will explain what a Fourier transform is, how it works and how it is used in radio astronomy.

3 What is a Fourier transform?

The Fourier transform is a mathematical way to describe signals in the frequency domain. Now the signals that we are familiar with are usually in the time domain, in that domain a sine wave goes from the left to the right over time. In the frequency domain a sine wave has a single peak at it's own frequency. The difference is illustrated in figure 1.

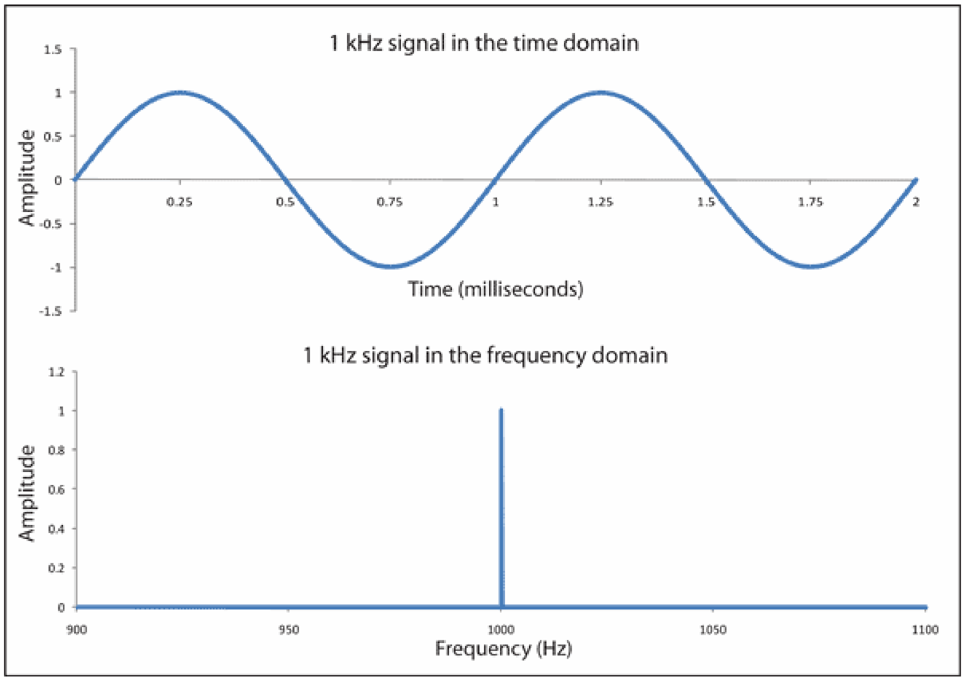

Figure 1: : time domain vs frequency domain

So the Fourier transform is a tool that can make you go from the time domain $f(t)$, to the frequency domain $F(\omega)$. It is written as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$$

There also is an inverse Fourier transform, logically this goes from the frequency domain to the time domain:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega$$

But how to make sense of this formula? Lets split it up into several parts, starting with $e^{-j\omega t}dt$. Euler's number to the power of $\sqrt{-1} = j$ so e^{j} moves around in a circle on the complex plane for every 2π units. So if we want to describe a sine wave over time (t) we can use $e^{-j2\pi t}$, but we also need the frequency (f) so that gives $e^{-j2\pi ft}$ and since $2\pi f$ equals ω we get

$$e^{-j\omega t}$$

Right so now you can plot this for a lot of frequencies (from $-\infty$ to ∞) and you'd get a bunch of different positions on a circle in the complex plane. Now if you multiply this by your time domain function $f(t)$ the positions in the circle will align just right with the frequencies that are most present in that function. That way the frequencies that are most present get magnified where frequencies that aren't present stay small. And that is how we get

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$$

3.1 Discrete Fourier Transform

In the real world however you don't measure from negative infinity to infinity, since your data will always be over a finite time frame and your sensors won't be able to generate an infinite amount of samples. In the real world you will get a set of discrete points from when you begin plotting (t_0) to the N^{th} sample (t_{N-1}) This is where the discrete Fourier transform (the DFT) comes in, it is described as

$$X(k) = \sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi kn}{N}}$$

Here it isn't an integral over infinity but a sum for all values of $(n = 0)$ to the total amount of samples $(N-1)$ Also in the exponential $\frac{k}{N}$ now describes the frequency. But now it instead of any possible frequency we are limited to a set of frequencies which are defined by the sampling frequency, and the total amount of samples (N). This is partly because of the Nyquist limit, which states that to accurately measure a frequency you need to sample at at least twice that frequency. And thus is is impossible to use any frequency higher than: $\frac{samplefreqency}{2}$

The DFT is much more suitable for computation since it is a finite set of calculations.

3.2 The Fast Fourier Transform

The computation time for computing a Fourier transform is proportional to the amount of samples (n) squared, n^2 so naturally this gets slow rather fast the higher the number of samples gets. So in 1965 James Cooley and John Tukey developed the Fast Fourier transform (the FFT). With the FFT the computation time is proportional to $\frac{n}{2}\ log_2\ n$ [6] which is significantly faster. We can see just how much faster it is if we divide n^2 by $\frac{n}{2}\ log_2\ n$ giving:

$$y = \frac{2n^2}{nlog_2n}$$

where y is the amount of times the FFT is faster and n being the nr of samples. When plotted it looks like this:

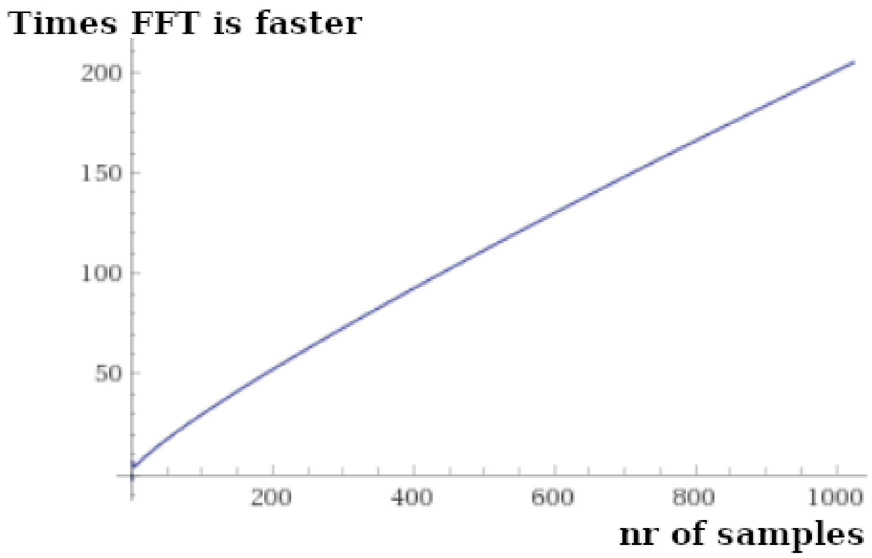

Figure 2: : Fourier vs Fast Fourier

Here we can see that at 1024 samples the FFT is about two hundred times faster than a normal Fourier transform. The FFT achieves this speed by recomposing larger point transformation out of several smaller point transformations.

4 How does the FFT work in two dimensions?

In radio astronomy the FFT is often used for image processing, images however aren't the one dimensional signals that we have been discussing so far. Images work in **two** dimensions, and these images too can be separated into a collection of sine and cosine waves. Here is an example of some two dimensional sine waves.

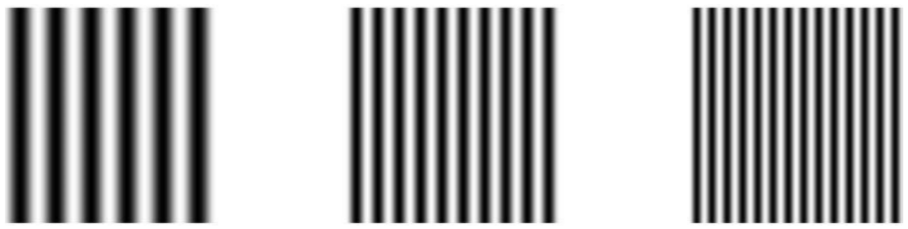

Figure 3: : Two dimensional sine waves

Now the Fourier transform of these waves is surprisingly simple, a single or a few dot(s) in the right place can be used to describe these waves, as is shown here.

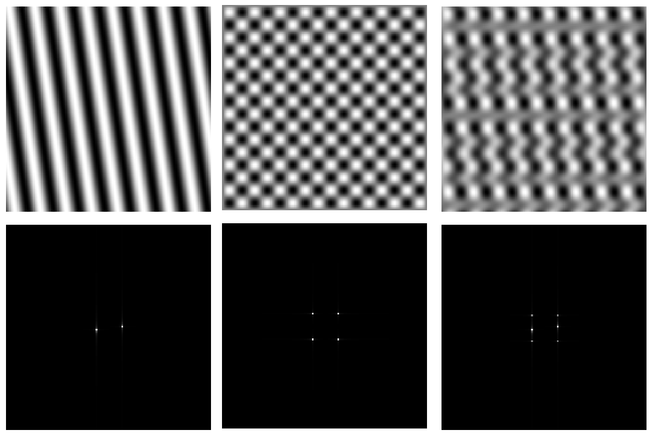

Figure 4: : Two dimensional sine waves with their Fourier transforms

Here you can already see that as more dots are added in the frequency domain the two dimensional image gets more complex. And thus you can imagine that any Two dimensional picture can be represented using a matrix of dots in the frequency domain. If colour pictures want to be achieved you will need three of these grey value matrices. One for red, one for green and one for blue when those are put on top of each other a colour picture is made.

In radio astronomy it works the other way around. Instead of having an image which has to be translated to the frequency domain, the array of radio telescopes receive their information in the frequency domain and will have to convert it back into the time domain to get a coherent image. To achieve this a reverse FFT will have to be applied to the data. This image from radioastronomydm[7] shows it quite well (transformation goes from right to left):

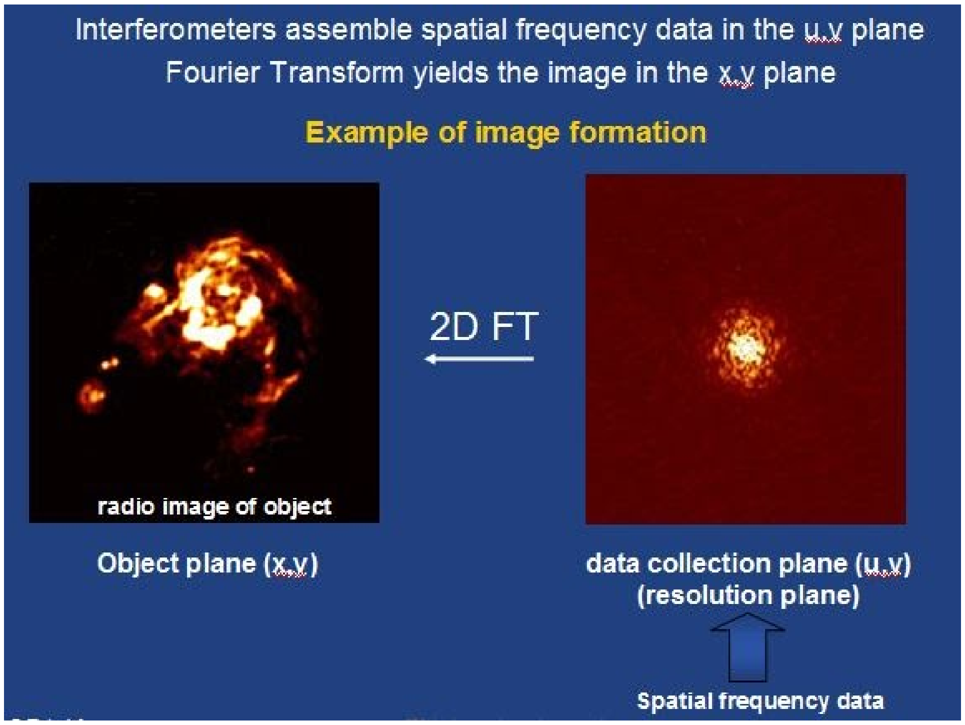
Interferometers assemble spatial frequency data in the u,v plane
Fourier Transform yields the image in the x,y plane
Example of image formation
radio image of object
2D FT
Object plane (x,y)
data collection plane (u,v)
(resolution plane)
Spatial frequency data

Figure 5: : Fourier transform use in radio astronomy

But how exactly does it do this, well that is actually quite simple. First you look at the image as a matrix of values, then you apply a Fourier transformation over the rows, and then over the columns. And that's it, now for the inverse Fourier transform you will have to apply it over the columns first and then end with the rows to get back to the original image.

5 How are we planning to set up a test platform

We want to simulate a multi processor system that preforms a two dimensional inverse FFT on an image. We want to measure how adding and subtracting processors affects Calculation time, power consumption and heat generation.
To simulate the multi processor system we will use a stack of raspberry pi's. For the measuring of the power consumption we have developed dedicated hardware, and data logging software. These will be discussed in chapter **(!!!! placeholder for chapter on hardware and logging software !!!!)** For the algorithm we want the stack to be able to compute a two dimensional FFT and, the inverse two dimensional FFT.
To test if it works we will feed the program an image of which we know what the outcome should be. Like the ones shown in the examples from figure[4]. Then if it passes that test we will feed that image back into the stack to see if it's inverse FFT transformation brings it back to the original image. If so, the program works. The process is illustrated here:

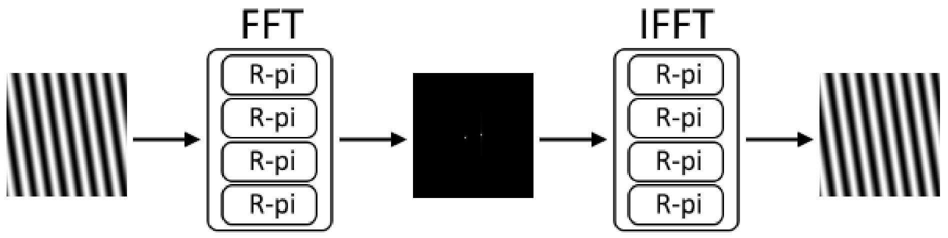

Figure 6: : Test to see if algorithm works

Now we also want to know what the influence of multi processing is on power consumption and computation speed, so we will have to design the algorithm in a way that enables us to add and subtract processors without compromising the algorithm. That way we can run the algorithm on a single to n amount of processors, and measure the effects.

6 The FFTW library

To make the project as open access as possible we have chosen for using a library that is maintained, and released under a free public license.

The FFTW[2] library is a library written in C++. It can preform Discrete FFT and IFFT, it can do this with multiple threads. And when using MPI[3] it can do it with multiple processors. The FFTW software is free software released under the GNU General Public License.
This checks all the boxes for what we want our test platform to be able to do so this is the library we will use.

First we created a program that could run the test as described in figure[6] For this we also needed a way to parse and generate bitmaps, Another free licensed library was used for that bit, bitmap_image.hpp from partow.net. For the generation of the input string (a 2d sine wave) Matlab was used. Here a pseudo version of the main function is shown:

```
int main(int argc, char **argv) {

/* create bmp file from input string */
input_string_to_bmp(real_part);
/* fill an array with data  */
fill_signal_1d(signal, real_part, imaginary_part);
/* create plan for time domain to freq domain  */
fftw_plan plan = fftw_plan_dft_2d(sqrt(NUM_POINTS),
sqrt(NUM_POINTS), signal, result,FFTW_FORWARD,FFTW_ESTIMATE);
/* create plan for freq domain back to time */
fftw_plan iplan = fftw_plan_dft_2d(sqrt(NUM_POINTS),
sqrt(NUM_POINTS), result, iresult,FFTW_BACKWARD,FFTW_ESTIMATE);
/* execute plans and print the magnitudes */
fftw_execute(plan);
calc_magnitude(result);
/* make bmp from freq domain */
FFT_string_to_bmp(result);
fftw_execute(iplan);
/* make bmp from inverse fft */
IFFT_string_to_bmp(iresult);
calc_back_to_samples(iresult);

return 0;
}
```

Listing 1: FFTW image process main.cpp

[2]Fastest Fourier Transform in the West
[3]Message Passing Interface

The code yielded the following results: First the input string is converted to bitmap to check if the input string is interpreted the right way.

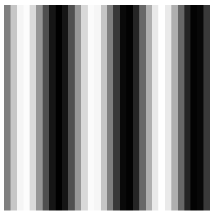

Figure 7: : The input string converted to bitmap

This two dimensional sine wave is then transformed to the Fourier domain, and if the program works correctly we should expect a mostly black image with some white dots.

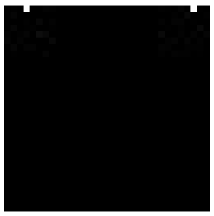

Figure 8: : The input string converted to the Fourier domain

Now finally to be one hundred percent certain that the program works an inverse Fourier is applied to the image. If this returns the original image then we can deduce that the program doesn't affect the data in it's transformation and works correctly.

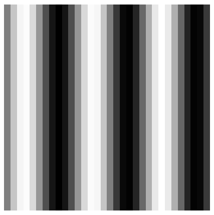

Figure 9: : The Fourier transformed string transformed back to the time domain

As shown the program works as expected and can now be used to interpret images from radio astronomy.

7 Message Passing Interface

There is however one more change that has to be made for the program to meet our requirements, multiprocessing. MPI is the protocol that multi processing systems use to communicate between processors. Using MPI with the raspberry pi can be done using one of several libraries, we decided to use the Mpich library.

This however did not turn out to be a wise choice. For some reason unbeknownst to us it just wouldn't work, the error messages were impossible to decipher and when looked up would lead you to years old mailing lists from Intel. After having done several re-installs which took quite some time, and following the provided install guide step by step. It still wouldn't work, not even the example codes included in the Mpich install package.

So having concluded that Mpich-3.3.2 does not (easily) work with the raspberry pi's, another MPI library will be chosen. The setup now uses the Open MPI library which is installed by apt, and this works without issues.

References

[1] "Astron, netherlands institute for radio astronomy."

[2] C. J. Wolfaardt, "Machine learning approach to radio frequency interference(rfi) classification in radio astronomy."

[3] E. M. Howard, "Machine learning algorithms in astronomy."

[4] A. Sclocco, "Accelerating radio astronomy with auto-tuning."

[5] J. A. Högbom, "Aperture synthesis with a non-regular distribution of interferometer baselines."

[6] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, Systems and Transforms.*

[7] "Google radioastronomydm2 interferometry."