P9 Towards distributed image reconstruction for radio interferometers

Jonas Schwammberger

November 19, 2019

Abstract

Naive distribution approaches are not efficient. They do not use the available resources, and get bogged down with communication. New approximation method. Way for easy parallel processing and later distribution.

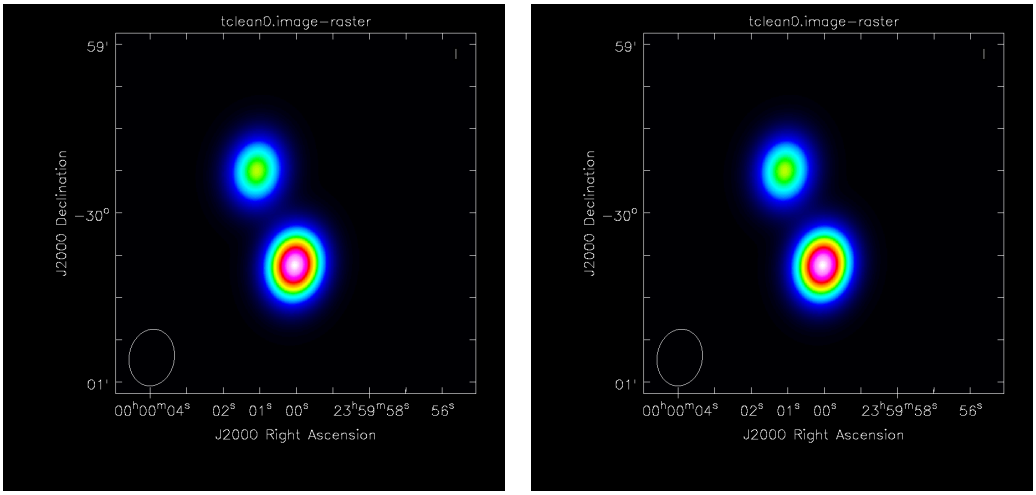
Contents

1 The image reconstruction problem of radio interferometers

In Astronomy, one goal is to find ever smaller objects in the sky. For this purpose, we build instruments with higher angular resolution. The instruments angular resolution depends on two factors: On the diameter of the antenna-dish or mirror, and on the observed wavelength. With longer wavelengths we need bigger dishes/mirrors to achieve a similar angular resolution.

This is an issue for Radio Astronomy. The long radio wavelengths require huge dishes for a high angular resolution. Of course there is a practical limit on the antenna-dish diameter we can build. The famous Arecibo observatory is one of the largest single-dish radio telescopes with a diameter of 305 meters. Antennas with such a large diameter become difficult to steer accurately, let alone the construction costs. We have reached the practical limits of single-dish telescopes. If we require higher angular resolution, we need to look at another type of instrument: The radio interferometer. They use several smaller antennas together, acting like a single large dish. An interferometer can achieve angular resolutions which are comparable to dishes with a diameter of several kilometers.

But there are drawbacks: The interferometer does not measure the sky in pixels. It measures the sky in Fourier space. As such, an interferometer produces amplitude and phase for each Fourier component that was measured. The observed image has to be reconstructed from the Fourier measurement. The measured Fourier components are called visibilities in the Radio Astronomy literature. From this point forward, we will call the measured Fourier components visibilities. The Figure 1 shows an example of the image reconstruction problem. The Figure 1a shows the measurements in the Fourier space, and the figure 1b shows the observed image of the sky, with two stars close to each other. The image reconstruction has to find the observed image 1b from the measurements 1a.



(a) Measurements of the interferometer in the Fourier space.

(b) Observed image of the sky, showing two stars.

Figure 1: The image reconstruction problem, the observed image has to be reconstructed from the Fourier measurements.

At first glance, we might believe that the image reconstruction is trivial: The interferometer measures Fourier components, and efficient algorithms for the inverse Fourier transforms are well-known. However, two properties of the measured Fourier components make the image reconstruction difficult: The measurements are both noisy and incomplete.

The atmosphere of the earth is one source that introduces noise. It adds noise to the amplitude and phase of each measured Fourier component. The atmosphere changes over time and can under the right circumstances introduce a high level of noise compared to the signal. The image reconstruction should be able to

find the observed image from potentially very noisy Fourier measurements.

The interferometer measures an incomplete set of Fourier components. Note that the Figure 1a shows the Fourier space, which has missing components. The interferometer can only measure a limited set of Fourier components. The reconstruction algorithm has to find the observed image even though important Fourier components are missing from the measurements.

These two difficulties, the noise and the incomplete measurements, lead to the fact that there are many different candidate images that fit the measurements. This is known as an inverse problem. We want to find the observed image, even though all we have are imperfect measurements. From the measurements alone, we cannot decide which candidate is the truly observed image. However, we have additional knowledge that simplifies the inverse problem: We know it is an image of the sky, which consists of stars, hydrogen clouds, etc. By including prior knowledge in the reconstruction, we can find the most likely image given the measurements.

The question remains is: How close is the most likely image to the observed one? Is exact reconstruction possible where the most likely and observed image are equal? Surprisingly the answer is yes. It is possible in theory[1, 2], and was shown in practice on low noise measurements[3, 4]. However, not all algorithms perform equally well when the noise level in the measurements is high. Also, computing resources required for each algorithm can vary significantly. In short, a reconstruction algorithm has three opposing goals:

1. Produce a reconstruction which is as close to the truly observed image as possible.
2. Robust against even heavy noise in the measurements.
3. Use as few computing resources as possible.

No reconstruction algorithm performs equally well on all three goals. One of the most widely used reconstruction algorithms is CLEAN [5, 6]. It has shown to be robust against heavy noise and, depending on the observation and is one of the oldest algorithms still in use today. As such, it was developed before the advent of distributed and GPU-accelerated computing. Today's new radio interferometers produce ever more measurements. The recently finished MeerKAT radio interferometer produces roughly 80 million Fourier measurements each second. Astronomers wish to reconstruct an image from several hours worth of measurement data. Reconstructing an image from this data volume requires GPU and distributed computation. But how to use GPU and distributed computing effectively is still an open problem.

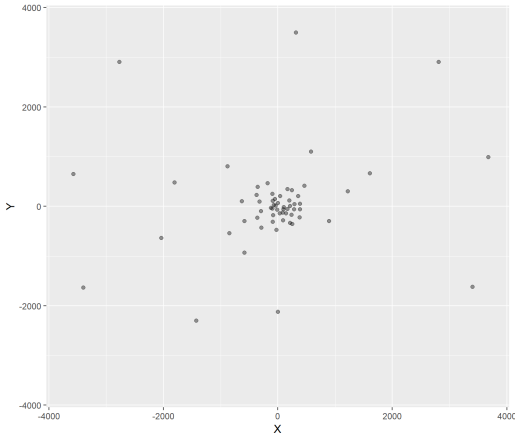
Coordinate descent methods have been successfully applied in other inverse problems, such as reconstruction of CT scans[7], or X-Ray imaging[8]. GPU accelerated[9] and distributed[10] variants have been developed. To our knowledge, coordinate descent methods have not been explored for the inverse problem in Radio Astronomy.

In this work, we develop our own proof-of-concept image reconstruction algorithm based on coordinate descent methods. We apply the reconstruction on a real world MeerKAT observation provided by SARAO. We explore the possible speedups we can achieve by using GPU and distributed computation. The algorithm is implemented platform independent in .netcore.

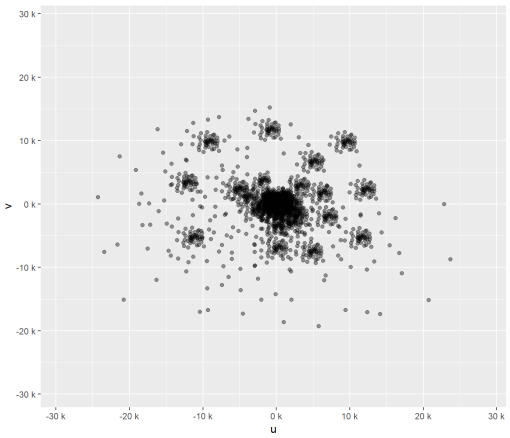
The rest of this work is structured as follows. First in section 2, we give an introduction to radio interferometric imaging, and give the theoretical background to why a reconstruction can even achieve a higher resolution than the instrument. Next we present the current state-of-the-art in image reconstruction for radio interferometers in section 3. Then we derive a basic image reconstruction algorithm based on coordinate descent in section 4, and show how we can use GPU acceleration and distribution to speed up reconstruction.

2 Introduction to radio interferometric imaging

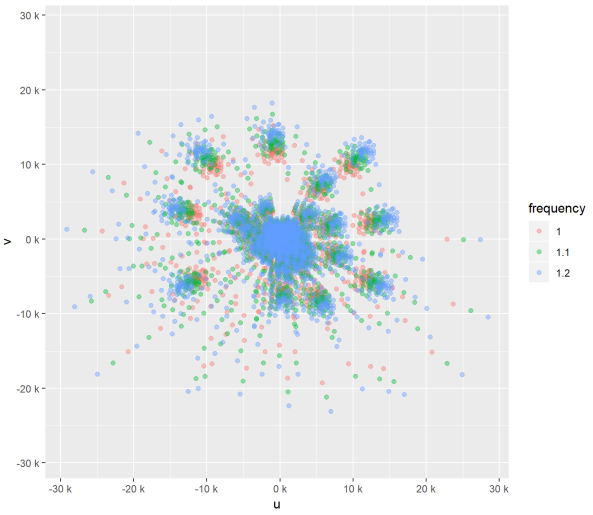
A radio interferometer consists of several antennas. Each antenna pair measures a visibility in Fourier space. Each measurement consists of an amplitude and phase at a location at a u and v location. The distance between the antennas, which we call the baseline, defines what point in the Fourier space gets sampled. The Figure 2a shows the antenna layout of the MeerKAT radio interferometer, and the Figure 2b shows the measurement points in Fourier space. Short baselines sample points close to the origin, and contain the low-frequency Fourier components. They contain information about large areas of the images. Longer baselines measure points further away from the origin. They sample the high-frequency Fourier components. They contain information about edges, and other small structures in the image.



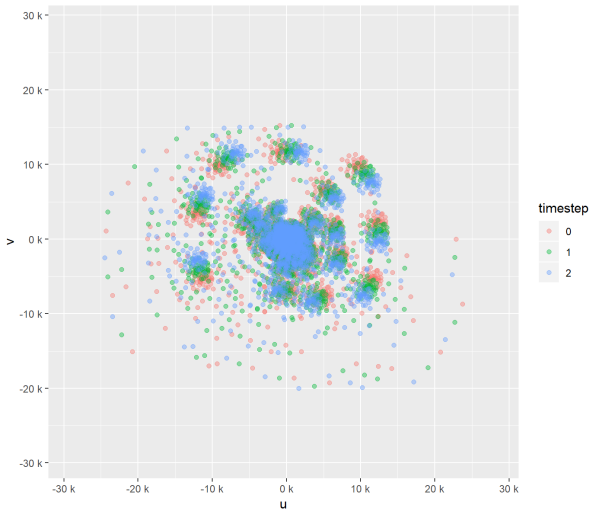
(a) Antenna layout.



(b) Visibility sampling pattern.



(c) Visibilities added from multiple channels.



(d) Visibilities added from multiple timesteps.

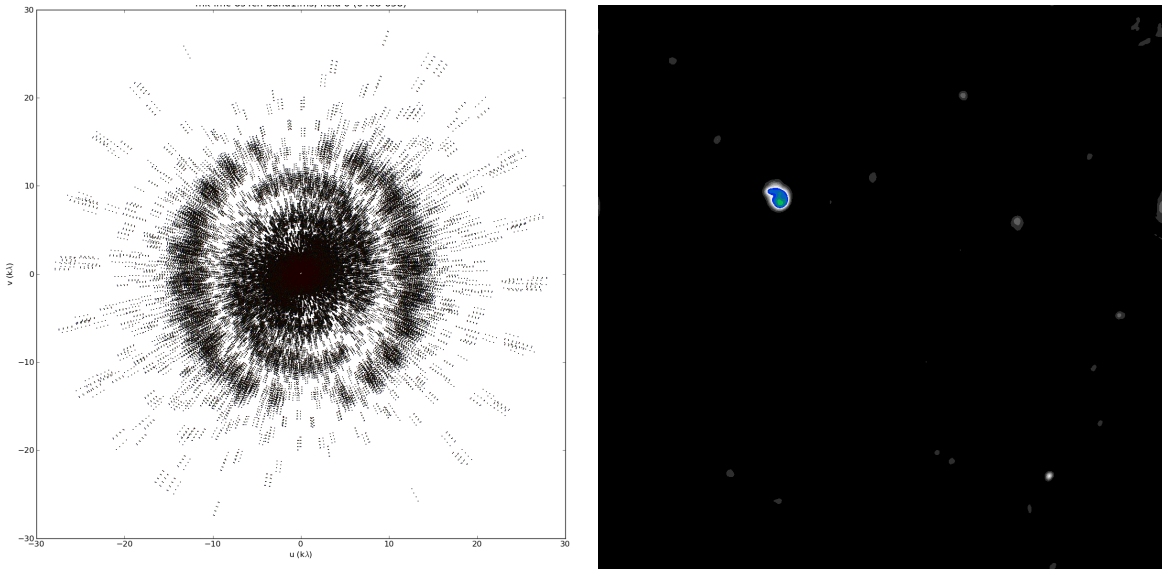
Figure 2: Sampling regime of the MeerKAT radio interferometer.

The sampling pattern of the MeerKAT interferometer is not uniform in the Fourier space. We have areas which are densely sampled, and areas which are sparsely sampled. Note that we only have a few samples of the high-frequency Fourier components. We are missing measurements from a large portion of the Fourier space.

Radio interferometers use two ”tricks” to measure more points in the Fourier space. Radio interferometers measure the sky in different radio channels simultaneously. We can add the visibility measurements from different channels together, shown in Figure 2c. Each channel measures the Fourier space using the same pattern, but scaled by the radio frequency.

The second trick is to use the earth's rotation to sample different points in the Fourier space. The earth's rotation also rotates the sampling pattern in Fourier space, shown in Figure 2d, and we can sample the Fourier space at new locations.

The MeerKAT radio interferometer measures 2016 visibilities, for each channel, at each timestep. It has 20 thousand radio channels. The time resolution can be as low as half a second. This results in roughly 80 million visibility measurements per second. In radio astronomy, we want to reconstruct several hours worth of visibility measurements.



(a) Real-world visibilities combined from different channesl and timesteps.

(b) Reconstruction of the visibility measurements.

Figure 3: Example of an image reconstruction for Fourier measurements of the MeerKAT radio interferometer

It is easy to see that the visibility measurements from MeerKAT quickly fills up the hard disk and the Fourier space with samples. The Figure 3a shows a fraction of the visibility samples from a real-world observation. The visibilities are combined from multiple channels and multiple timesteps. Although we have a large number of visibilities, we still have areas of the Fourier space without a sample. Although we have a large number of samples, the visibilities are still an incomplete. From the measurements alone, we cannot reconstruct the image shown in Figure 3a.

This is known as an ill-posed inverse problem in the literature. A problem is considered ill-posed when:

1. No solution exists.
2. There are solutions, but no unique solution exists.
3. The solution behavior does not change continuously with the initial condition (For example: a small change in the measurements lead to a very different reconstructed image).

Image reconstruction for radio interferometer is an inverse problem, because we want to find the image which the interferometer observed in Fourier space. It is ill-posed in general, because there are many different images fitting the measurements.

Note that we can reduce the resolution of the reconstructed image until the problem becomes well-posed. The Nyquist-Shannon sampling theorem states the case of radio interferometers: The highest highest Fourier-frequency we measure should be more than twice the highest frequency in the image. The center of the Fourier space in Figure 3a is densely sampled. We can reconstruct a low-resolution image that only needs the information from the densely sampled center.

However, this would reduce the effective resolution of the reconstruction. If we can solve the ill-posed inverse problem, we would be able to retrieve the observed image at a higher resolution than possible with the Nyquist-Shannon sampling theorem. As it turns out, this is possible to solve the ill-posed inverse problem by including prior information. We use a numerical optimization algorithm and find the optimal image, which is both consistent with the measurements and consistent with our prior knowledge. This is known in signal processing as compressed sensing [1, 2] in the literature. The theory of Compressed Sensing shows that, under the right prior information, we are guaranteed to reconstruct the observed image at a higher resolution than under the Nyquist-Shannon sampling theorem.

2.1 The theory of Compressed Sensing

We introduce the theory of Compressed Sensing for the problem of radio interferometric image reconstruction. As we have mentioned compressed sensing image reconstruction involves a numerical optimization algorithm to find the optimal solution which is consistent with our measurements and consistent with our prior knowledge. As we will see later, the theory of compressed sensing guarantees us exact reconstruction under certain assumptions.

Let us formulate the image reconstruction as an optimization problem.The image reconstruction wants to to find the image which is as close to the measurements as possible. Or more formally, we want to minimize the euclidean distance between the visibility measurements V and the reconstructed image x. We write it as an objective function:

$$\underset{x}{minimize} \ \|V - Fx\|_2^2 \tag{2.1}$$

We can reconstruct the image by finding the optimum of the objective function (2.1). The objective function is convex, meaning it has only one global minimum, and we can use the class of convex optimization algorithms to search the minimum. However, our measurements V are incomplete, meaning we do not have all the data we need for reconstruction. This means our objective function (2.1) does not ”point” to the observed image. It still has a global minimum, but observed image is not guaranteed to be near the global minimum.

A side note: We are guaranteed to find the observed image at the minimum of (2.1) is when the measurements fulfill the Nyquist-Shannon sampling theorem. In that case, we can find the minimum by calculating the inverse Fourier transform: $x = F^{-1}V$. We can still calculate the inverse Fourier transform when we are dealing with incomplete measurements, but it does not result in the observed image.

However, the objective (2.1) only includes information about the measurements. As we have mentioned before, we have prior knowledge about the image. We know it is likely to contain stars. Stars are radio-emissions which are concentrated around a single pixel. In that case, most pixels of the image will be zero, except for the locations where the interferometer has located stars. In other words, we know that the image is sparse. We can add a regularization to the objective function (2.1) and force the reconstructed image to be sparse. This results in the modified objective function:

$$\underset{x}{minimize} \ \|V - Fx\|_2^2 + \lambda \|x\|_1 \tag{2.2}$$

Note the two terms in the objective (2.2): We have the same term from our measurements, which we call the ”data term”. But we also have an additional ”regularization term”, which is the L1 norm[1] and forces our reconstruction to be sparse. The parameter λ represents how much emphasis we put on the regularization. The new objective function is still convex, it still has a global minimum. The regularization term simply shifted the global minimum to a different location when compared to the first objective (2.1). Now the question is: How

[1] Sum of absolute values of the pixels

likely is our modified objective (2.2) to point at the observed image? The theory of Compressed Sensing tells us that it depends on the image content of the observed image. If it only consists of stars, we are practically guaranteed to find the observed image at the minimum of (2.2), even though we are dealing with incomplete measurements.

We use the words 'practically guaranteed' because the theory of Compressed Sensing does give us guarantees to find the observed image at the minimum of (2.2) under certain assumptions. The issue is these assumptions are hard to verify for any given reconstruction problem. In reality, it is often easier to empirically show that the regularization works. For example, by creating a super-resolved[2] reconstruction from visibility measurements [3]. The assumptions do have important implications for instrument design. This project however is focused on the numerical optimization algorithm. We do not have an influence on the visibility measurements, for our intends and purposes, the measurements are fixed. That is why we first give an intuitive explanation to when the observed image is at the minimum of the objective (2.2), and what this means for the ill-posed image reconstruction next. Then we give an overview of the formal guarantees in Section 2.1.2 and the implications for the visibility measurements.

2.1.1 Exact reconstruction in theory

What is exact reconstruction: When the most likely image is equal to the observed image.

Let us go back to our example of an image containing only stars. Let us say it has 256^2 pixels and contains $S = 10$ stars. But we do not know how many it has, where they are, nor their intensity (pixel value). We just know there ara only stars in the image.

If we measure the image with a single-dish instrument, we measure every pixel after each other. Note that, because we know the image contains only stars, we already know most pixels will be zero. When the single-dish measures a zero pixel, all we learn is that there is no star at this location. We only learn something vital when the single-dish instrument hits one of the 10 stars. We learn little when the instrument measures a zero pixel, and a lot when it measures a pixel with a star. On average over all pixel measurements, we learn little about the image.

When we measure the image with a radio interferometer, we measure visibilities (Fourier components) of the image. Remember that a single visibility is a measurement over the whole image. Every pixel of the image has contributed to the amplitude and phase of the visibility. Every visibility measurement contains some information about the 10 stars in the image. On average, each visibility measurement contains more information about the stars than the average pixel measurement of the single-dish instrument.

T

Randomly sampling the visibilities tends to be optimal.

The question that remains is: What visibilities, and how many do we need to reconstruct our image? The answer depends on two components: It depends if the matrix F of our objective (2.2) fulfills the Restricted Isometry Property (RIP) and on the number of stars in the image.

The Restricted Isometry Property (RIP)[1, 2] basically says that a random subset of columns in F have to be (approximately) uncorrelated. In our example with an image containing 10 stars, it means that each star is uncorrelated from each other. Another example: they have to be roughly orthogonal. If we have found a solution with 10 stars, the RIP is the tool to prove that there exists no solution with fewer stars. Therefore the solution is optimal.

At what point we fulfill the RIP also depends on the number of stars. Intuitively this makes sense. We do not need as many data if the image is simple. Note on sparseness: also possible in different spaces. Funny thing,

[2] The radio interferometer also has a resolution limit. Super-resolved reconstructions were able to find structures which are smaller than the limit of the instrument.

when we find a space which the image is even more sparse, we need even less visibility measurements for reconstruction.

If we randomly sample the Fourier space, the resulting matrix F is likely to fulfill the RIP[15]. Likely means for all practical purposes, it might as well be guaranteed.

2.1.2 Exact reconstruction in practice

Interferometer does not sample randomly, and does not only contain stars.

A radio interferometer does not randomly sample the visibilities. As we have seen, the sampling pattern depends on the antenna layout of the interferometer. For us, the matrix F of a reconstruction problem is fixed, and cannot be changed after the observation is recorded. Do not have the guarantees. But that is not too bad, we may not need the RIP. Exact reconstruction can also work under less strict conditions[16].

What to do about extended emissions. It is a data modelling task, how do we find good regularization that leads to the sparset result possible. Problem of validation, we do not know the best regularization beforehand. We can only choose one that works ”well.

The theory of compressed sensing gives us a framework. There is a data modelling task, and a task for finding efficient algorithms.

So there is a data modelling task in finding a good sparse prior. We also do not know the proper sparse space in which radio interferometric images. We know several spaces, Curvelets [11] Starlets [12], Daubechies wavelets [22]. As of the time of writing, it is currently unknown which leads to the best reconstruction. We can also learn dictionaries.

2.1.3 General Compressed Sensing reconstruction formulation

Use everything with the L1 norm. More general formulation

F is fixed, because we cannot change the interferometer.

We do just need a sparse space.

2.2 Noise, approximations and other difficulties in radio interferometry

We give a short introduction into how the electromagnetic wave gets measured by the interferometer, turned into visibilities and finally processed into an image. Figure 4 shows a radio source and its electro-magnetic (em) wave arriving at the antennas of the interferometer. It then shows the three processes involved to arrive at an image: Correlation, calibration and image reconstruction.

First, we have a source in the sky that is emitting em-waves in the radio frequency. The waves travel to earth, through the earth’s ionosphere and finally to our interferometer. Along its path, the e-m waves may get distorted from various sources. For example, it may receive a phase shift by the ionosphere.

Then, the em-wave arrives at our interferometer. We call each antenna pair a baseline. Each baseline will end up measuring a single visibility. The distance between the antennas and their orientation to the em-wave will determine where we sample the uv-plane. Short baselines measure the uv-plane close at the origin, while long baselines sample the uv-space further away from the origin. Remember that the samples away from the uv-origin contain the information about edges and other details of our image. With a longer baseline the interferometer measures more highly resolved details, regardless of the antenna dish-diameter[3]. The figure

[3] Remember that this is the reason why we build radio interferometers. We do not need impossibly large dish diameters for a high angular resolution. We just need large distances between smaller antennas.

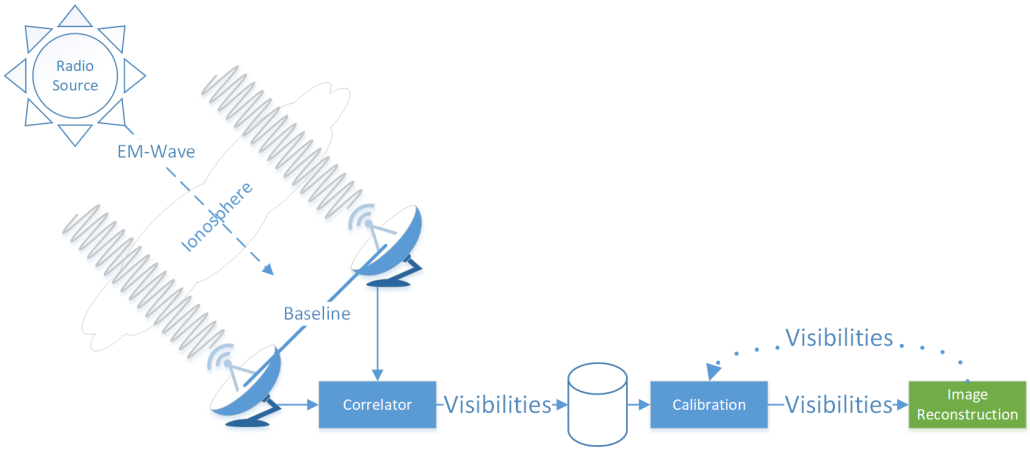


Figure 4: Radio interferometer system

4 shows the em-wave arriving at a single baseline of the interferometer. Each antenna picks up its version of the em-wave and transfers it to the correlator.

The correlator then takes the feed of each antenna and correlates the signals, which results in the amplitude and phase of the visibility component. Amplitude and phase for each visibility are measured for a short time range (i.e. fractions of a second up to several seconds). At this point, the visibilities are saved to disk for further processing. The radio interferometer produces a visibility measurement for each baseline, for each time range, for each frequency channel of the instrument. Because a single observation can take up to several hours, measured with several thousand frequency channels, radio interferometers produce an almost arbitrary large number of visibilities.

Calibration

Image Reconstruction

2.2.1 The measurement equation

As we discussed so far, the radio interferometer measures visibilities of the sky image, and we wish to find the observed image from the measurements. Put formally, we wish to invert the following system of linear equations (2.5), where V is the visibility vector[4], F is the Fourier transform matrix and I is the pixel vector of the observed image.

$$V = FI \tag{2.3}$$

We wish to find the observed image I, while we only know the visibility vector V and the Fourier transform matrix F. This is what we call the measurement equation. In most context for this project, looking is an adequate view of the image reconstruction problem. We will show why we cannot find the observed image I by simply calculating the inverse Fourier transform. However, when we need to efficiently apply the Fourier transform, we need to know F in more detail. As we will see, radio interferometers have some difficulties hidden in the Fourier transform matrix, which are difficult to handle efficiently. First, let us abandon the vector notation of (2.5), and represent the measurement equation with integrals (2.6).

$$V(u,v) = \int\int I(l,m)e^{2\pi i[ul+vm]}\,dl\,dm \tag{2.4}$$

[4]We use the lower-case v to denote the axis in the Fourier space uvw, and the upper-case letter to denote the visibility vector.

This is essentially the same problem. The main difference is that we do not represent the Fourier transform as a matrix F, but as integrals $\int\int e^{2\pi i[ul+vm]}$, where u, v are the coordinates in Fourier space and l, m are the angles away from the image center. A single pixel represents the intensity of the radio emission from the direction l, m. Note that the measurement equation (2.6) shows the fact that the visibilities are measured in a continuous Fourier space. If the Fourier space would also be discrete, we could replace the integrals with sums.

However, the measurement equation (2.6) is inaccurate in the sense that it ignores many effects that distort the signal. For example, it does not account for the distortion by the ionosphere, or the distortion introduced by real-world antennas. The measurement equation (2.6) shown here does not represent the real world. But depending on the instrument and the observation, these distortions may be negligible, and the measurement equation (2.6) is a good approximation.

When there is a distortion source that cannot be ignored, it has to be modelled in the measurement equation. As such there is no unified measurement equation for all radio interferometric observations, let alone radio interferometers. The equation shown in (2.6) can be seen as the basis that gets extended as necessary[23, 24, 25, 26].

For example, the measurement equation (2.6) is only accurate for small field of view observations, when l and m are both small angles. For wide field of view observations, we need to account for the fact that the visibilities have a third term w, and we arrive at the wide field of view measurement equation (2.7).

$$V(u,v,w) = \int\int \frac{I(l,m)}{c(l,m)} e^{2\pi i[ul+vm+w(c(x,y)-1)]}\, dl\, dm\,, \quad c(l,m) = \sqrt{1-l^2-m^2} \tag{2.5}$$

The third w-term has two effects on the measurement equation. It introduces a phase shift in the Fourier transform $e^{2\pi i[...+w(c(l,m)-1)]}$, and a normalization factor of the image $\frac{I(l,m)}{c(l,m)}$. Note that when the angles are small, i.e. $l^2 + m^2 \ll 1$ then the wide field of view measurement equation (2.7) reduces to our original (2.6). This is another way of saying that for small field of views, the measurement equation (2.6) is a good approximation under the right conditions.

In this project, we use the wide field of view measurement equation (2.7). But as we mentioned in the beginning of this section, for most contexts, it is not important whether we ignore the w-term of the visibilities or not. It is important when we design an efficient implementations for applying the wide field of view Fourier transform, because the w-term keeps us from using the Fast Fourier Transform (FFT). In every other case, we can ignore this technicality. Because even more complicated measurement equation still have a linear relationship between visibilities and image [23, 24, 25, 26]. We can view the whole reconstruction problem as a system of linear equations (2.5), where the matrix F takes care of how exactly the measurements and pixels relate in this case.

2.3 Introduction into optimization/RI reconstruction algorithms

2.3.1 Image reconstruction as deconvolution

2.3.2 CLEAN deconvolution algorithm

2.3.3 The Major/Minor cycle

3 State of the Art image reconstruction

Image reconstruction pipelines are split in two taks. Gridding and deconvolution. We introduce here the two latest gridding algorithms, w-stacking in 3.1 and Image Domain Gridder 3.2.

Deconvolution we discuss CLEAN and MORESANE.

Not all algorithms are here from

3.1 w-stacking Gridder

3.2 Image Domain Gridding Algorithm

3.3 Deconvolution

3.3.1 CLEAN

Various Improvements and speedups[6, 27, 28, 29], but the core algorithm is still this.

3.3.2 MORESANE

3.4 Coordinate Descent

4 Coordinate descent methods

In this section we describe the basic idea behind coordinate descent methods in general, and derive a serial coordinate descent deconvolution algorithm. This algorithm replaces CLEAN in the Major/Minor cycle architecture. The algorithm we describe here is serial in the sense that each step of the algorithm has to finish before the next step can be started. Each individual step can use multiple processors, as we will show with a GPU-accelerated implementation. Later in this work, in Section 7, we will introduce more sophisticated parallel coordinate descent methods.

Remember that a deconvolution algorithm in radio astronomy has three components: A numerical optimization algorithm, an objective function and a regularization. We use a serial coordinate descent method for the optimization algorithm. The objective function is:

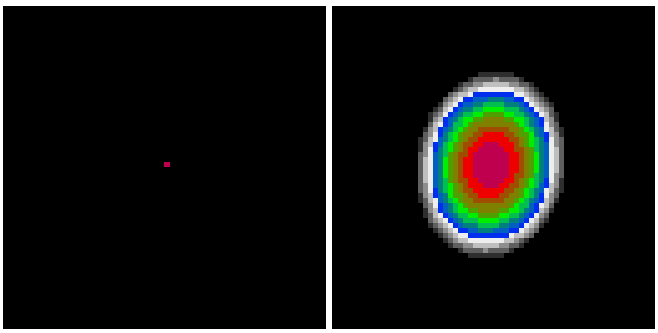
$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * PSF\right\|_2^2 + \lambda ElasticNet(x) \tag{4.1}$$

The objective consists of two parts: The data term $\|I_{dirty} - x * PSF\|_2^2$ and the regularization term $ElasticNet(x)$. The data term forces the image to be as close to the measurements as possible which forces the image to be as close to the measurements as possible, the regularization term forces the image to be as consistent as possible with our prior knowledge. The parameter λ is a weight that either forces more or less regularization. It is left to the user to define λ for each image.

And finally, the regularization we use is elastic net. We first go into more detail what the elastic net regularization is and how it influences the image. We then derive the serial coordinate descent method that minimizes the objective (8.1) in Section 4.2, and continue with its efficient implementation.

4.1 Elastic net regularization

This regularization is a mixture between the L1 and L2 regularization. The L1 regularization is simply the absolute value of all pixels, and the L2 norm is the squared sum of all pixels. he Figure 5 shows the effect of the L1 and L2 norm on a single star. The L1 regularization forces the image to contain few non-zero pixels as possible. It encodes our prior knowledge that the image will contain stars. The L2 regularization on the other hand ”spreads” the single star across multiple pixels.



(a) Effect of the pure L1 norm (λ = 1.0) on a single point source.

(b) Effect of the pure L2 norm (λ = 1.0) on a single point source.

Figure 5: Effect of the L1 and L2 Norm separately.

The L1 regularization alone models an image that consists of points sources. For extended emissions like hydrogen clouds, the L1 regularization often leads the reconstructed image to become a cluster of point sources, instead of a real extended emission.

The L2 norm alone was already used in other image reconstruction algorithms in radio astronomy[19], with the downside that the resulting image will not be sparse. I.e. all pixels in the reconstruction will be non-zero, even though all they contain is noise.

Elastic net mixes the L1 and L2 norm together, becoming ”sparsifying L2 norm”. It retains the sparsifying property of the L1 norm, while also keeping extended emissions in the image. Formally, elastic net regularization is defined as the following regularization function:

$$ElasticNet(x, \alpha) = \alpha \|x\|_1 + \frac{1-\alpha}{2} \|x\|_2 \tag{4.2}$$

The parameter α is between 0 and 1, and mixes the two norms together. A value of 1 leads to L1 regularization only, and a value of 0 leads to L2 only. The elastic net regularization has two properties, which are relevant later for the serial coordinate descent deconvolution: It is separable, and has a proximal operator.

Separability means that we can calculate the elastic net regularization penalty independently for each pixel. We arrive at the same result if we evaluate (4.2) for each pixel and sum up the results, or if we evaluate (4.2) for the whole image. This is an important property when one tries to minimize the the elastic net penalty (as we will with the serial coordinate descent deconvolution algorithm). We can also calculate how much any pixel change reduces the elastic net penalty independently its neighbors.

The proximal operator of elastic net allows us to minimize the regularization penalty. Notice that the elastic net regularization (4.2) is not differentiable (the L1 norm is not continuous). We cannot calculate a gradient, and cannot use methods like gradient descent to minimize the regularization penalty. However, it has a proximal operator defined:

$$ElasticNetProximal(x, \lambda, \alpha) = \frac{max(x - \lambda\alpha, 0)}{1 + \lambda(1-\alpha)} \tag{4.3}$$

In our deconvolution problem, we can apply the proximal operator (4.3) on each pixel, and we minimize the elastic net penalty. Again, the proximal operator can be applied on each pixel independently, as neighboring pixels do not influence its result.

The elastic net regularization is separable. We can calculate its penalty for each pixel independently of its neighbors. As such, its proximal operator is also independent of the neighbors. It is the only regularization we use in this project. We now derive a coordinate descent based deconvolution algorithm that uses the proximal operator to efficiently reconstruct an image.

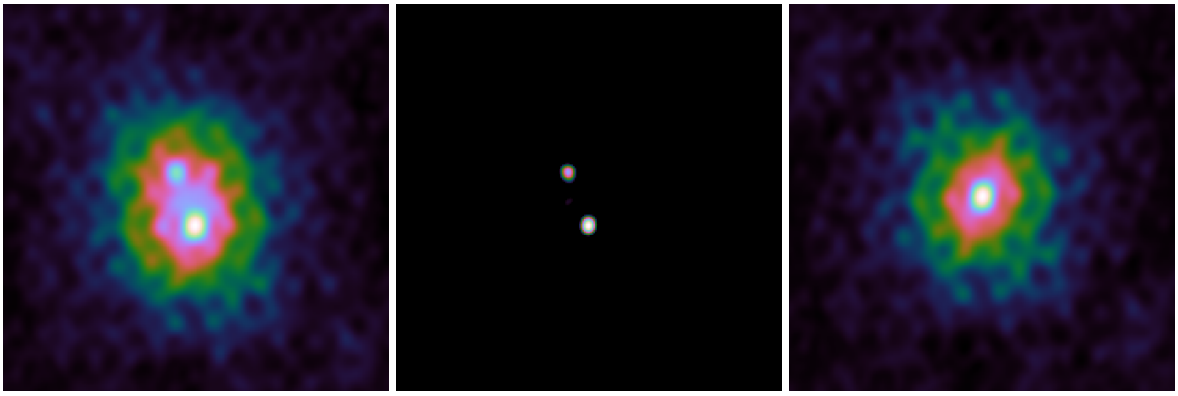
A side note on the proximal operator used in this project (4.3): The numerator always clamps negative pixels to zero. This is a conscious design decision. In radio astronomy, it is usual to constrain the reconstruction to be non-negative (because we cannot receive negative radio emissions from any direction). It is widely used in radio astronomy image reconstruction and may lead to improved reconstruction quality [30].

4.2 Serial coordinate descent deconvolution

Our serial coordinate descent deconvolution algorithm minimizes the deconvolution objective (8.1). It is a convex optimization algorithm that optimizes a single pixel (coordinate) at each iteration. Each iteration consists of two steps. Step 1: Find the best pixel to optimize. Step 2: Calculate the gradient for this pixel, take a descent step and apply the elastic net proximal operator. We repeat these steps in each iteration until coordinate descent converges to a solution.

We demonstrate the serial deconvolution algorithm with the help of a simulated MeerKAT reconstruction problem of two point sources. Figure 6a shows the dirty image of two point sources, and Figure 6c the PSF. The

deconvolved image with elastic net regularization is shown in Figure 6b. I.e. Figure 6b is the optimum x of the objective function (8.1).



(a) Dirty Image. *(b) Elastic net reconstruction* *(c) Point Spread Function.*

Figure 6: Example problem with two point sources.

The serial coordinate descent method finds the optimum over several iterations. Note that we termed the algorithm ”serial”, because step 1 (finding a pixel) first has to finish before we can continue with step 2 (optimizing the current pixel). There is always only one pixel which gets optimized at any given time.

Also note that our implementation calculates the gradient for the current pixel. This may raise the question: What exactly is the difference between gradient- and coordinate descent is that gradient descent optimizes all pixels in each iteration, while coordinate descent optimizes (generally) a single pixel at a time[5]. Also, Coordinate descent methods are not bound to use the gradient. It could use a line search approach, where we try different values and decide on the one leading to the lowest objective value.

Because coordinate descent methods only optimize a single pixel at a time, they generally need a large number of iterations to converge compared to other methods. But when each iteration is cheap to compute, coordinate descent methods can converge to a result in less time than competing methods[31, 32]. We discuss the efficient implementation in Section 4.3. First, we explain each step in the serial coordinate descent algorithm in more detail.

4.2.1 Step 1: Choosing single a pixel

Our serial coordinate descent algorithm uses a greedy strategy. From all possible pixels, it searches the pixel whose gradient has the largest magnitude. In each iteration, it chooses the pixel which reduces the objective function the most. There are two other strategies that are used in coordinate descent: Random and Cyclic.

A random strategy chooses, as the name implies, each pixel at random. Usually, the pixels are chosen from a uniform distribution. The greedy strategy leads to cheaper iteration compared to the greedy strategy, because we do not check the gradient of each pixel.

A cyclic strategy iterates over a subset of pixels until the subset converges. It then chooses another subset. Each iteration of the cyclic strategy is also cheaper than the greedy strategy.

For our image deconvolution problem, we choose the greedy strategy. Even though it is more expensive, it tends to be faster to converge. Our reconstructed image is sparse, meaning most pixels in the image will be zero. The greedy strategy tends to iterate on pixels which will be non-zero in the final reconstructed image. While the random or cyclic strategy add pixels in the intermediate image that will eventually have to

[5]There are block coordinate descent methods that optimize a block of coordinates at each iteration. They are also discussed together with parallel coordinate descent methods in Section 7

be removed. For the deconvolution problem, removing pixels from an intermediate solution seems to slow down convergence significantly. We explore the different strategies systematically for the parallel coordinate descent methods.

4.2.2 Step 2: Optimizing a single pixel

At this point, the greedy strategy has selected a pixel at a location to optimize. Now, our deconvolution objective (8.1) is reduced to a one dimensional problem:

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x_{location} * PSF\right\|_2^2 + \lambda ElasticNet(x_{location}) \tag{4.4}$$

Side note: The reason why this reduces nicely to one dimension is the elastic net regularization is separable. I.e. the regularization can be calculated independently of the surrounding pixels.

Optimizing the one dimensional problem (4.4) is a lot simpler. In essence, we calculate the gradient for the pixel at the selected location, and apply the proximal operator of elastic net. First, let us look at how the gradient is calculated and ignore the regularization. The gradient arises from the data term of the one dimensional objective (4.4) ($\|I_{dirty} - x_{location} * PSF\|_2^2$). After simplifying the partial derivative, we arrive at the calculation:

$$\begin{aligned} residuals &= I_{dirty} - x * PSF \\ gradient_{location} &= \langle residuals, PSF_{location} \rangle \\ Lipschitz_{location} &= \langle PSF_{location}, PSF_{location} \rangle \\ pixel_{opt} &= \frac{gradient_{location}}{Lipschitz_{location}} \end{aligned} \tag{4.5}$$

First, we calculate the residuals by convolving the current solution x with the PSF. Then, the gradient for the selected pixel location is the inner product(element-wise multiplication followed by a sum over all elements) of the residuals and the PSF, shifted at the current location. calculate the gradient for the selected location. The next step is to calculate the Lipschitz constant at the current location. Finally, we arrive at the optimal pixel value by dividing the gradient by the Lipschitz constant. Note, that we currently ignore the elastic net regularization.

The Lipschitz constant describes how fast a function $f(x)$ changes with x. If $f(x)$ changes slowly, we can descend larger distances along the gradient without the fear for divergence. The Lipschitz constant can be looked at as a data-defined step size.

An interesting point is that the update rule $pixel_{opt} = \frac{gradient_{location}}{Lipschitz_{location}}$ finds the optimal pixel value, if the pixel is independent. Remember that our objective function is convex. The data term of our one dimensional objective (4.4) actually forms a parabola, with the parameters: $x^2\langle PSF, PSF\rangle - 2x\langle resdiuals, PSF_{location}\rangle + c$. Calculating the optimum of the parabola $\frac{-b}{2a}$, is identical to calculating the gradient update (4.5). Note that $b = -2gradient_{location}$ and $a = Lipschitz_{location}$, and both the minimum of the parabola $\frac{-b}{2a}$ and the update rule based on partial derivatives (4.5) are identical.

This means if our reconstruction problem has point sources which are far away, such that their PSFs do not overlap, then the update rule finds the optimal value for each point source with one iteration. But when the PSFs overlap as in our example problem, shown in Figure 6, then we need several iterations over the same pixel until coordinate descent converges.

Including the elastic net regularization
So far, we ignored the regularization. We can calculate the optimal pixel value without elastic net regularization. The last step is to combine the proximal operator of the elastic net regularization (4.3) with the gradient calculation, and we arrive at the following update step:

$$pixel_{opt} = \frac{max(gradient_{location} - \lambda\alpha, 0)}{Lipschitz_{location} + (1-\alpha)\lambda} \tag{4.6}$$

This update rule now finds the optimal pixel value with elastic net regularization. If the PSFs do not overlap, we still only need one iteration per source.

4.2.3 Inefficient implementation pseudo-code

Now we put together our serial coordinate descent algorithm, and show where the bottleneck lies. In each iteration, the serial coordinate descent algorithm selects the pixel with the maximum gradient magnitude, and optimizes the selected pixel with the update rule (4.6).

```
1  dirty = IFFT(GridVisibilities(visibilities))
2  residuals = dirty
3
4  x = new Array
5  objectiveValue = 0.5* Sum(residuals * residuals) + ElasticNet(x)
6
7  do
8    oldObjectiveValue = objectiveValue
9
10   //Step 1: Search pixel
11   pixelLocation = GreedyStrategy(residuals, PSF)
12   oldValue = x[pixelLocation]
13   shiftedPSF = Shift(PSF, pixelLocation)
14
15   //Step 2: Optimize pixel
16   gradient = Sum(residuals * shiftedPSF)
17   lipschitz = Sum(shiftedPSF * shiftedPSF)
18   tmp = gradient + oldValue * lipschitz
19   optimalValue = Max(tmp - lambda*alpha) / (lipschitz + (1 - alpha)*lambda)
20   x[pixelLocation] = optimalValue
21
22   //housekeeping
23   residuals = residuals - shiftedPSF * (optimalValue - oldValue)
24   objectiveValue = 0.5 * Sum(residuals * residuals) + lambda * ElasticNet(x, alpha)
25 while (oldObjectiveValue - objectiveValue) < epsilon
```

The actual update step (line 19) is cheap to compute. We are only dealing with 4 one dimensional variables. The expensive calculations are the inner products. The gradient calculation, the Lipschitz constant and the objective value. The residuals and PSF generally contain millions of pixels. Calculating the inner product of those becomes expensive.

Also note that the greedy strategy needs to calculate the gradient for each pixel. As it is, the greedy strategy has a quadratic runtime complexity.

4.3 Efficient implementation

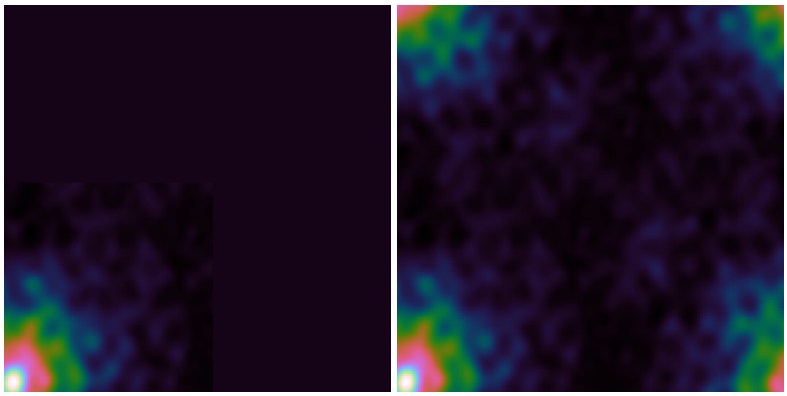
The bottleneck of the serial coordinate descent algorithm are all the inner products that need to be calculated in each iteration. In each iteration, we need to know the gradient for every pixel, and the Lipschitz constant of

the current pixel. Luckily, we can cache a map of gradients, where we save the gradient for every pixel and skip all of the inner products associated with the gradient. Also, we can efficiently calculate and cache the Lipschitz constants. We can greatly reduce the runtime cost for each iteration.

This section shows the implementation details on how we can calculate the map of gradients and the Lipschitz constant efficiently. But first we need to define another implementation detail: How we handle the edges of the convolution.

4.3.1 Edge handling of the convolution

As the reader is probably aware, there are several ways to define the convolution in image processing, depending on how we handle the edges on the image. Two possibilities are relevant for radio interferometric image reconstruction: Circular and zero padded.



(a) Zero padded convolution. *(b) Circular convolution.*

Figure 7: Comparison of the two convolution schemes.

Circular convolution assumes the image ”wraps” around itself. If we travel over the right edge of the image, we arrive at the left edge. The convolution in Fourier space is circular. Remember: A convonlution in image space is a multiplication in Fourier space, and vice versa. When we convolve the reconstructed image x with the PSF using circular convolution, then non-zero pixels at the right edge of the image ”shine” over to the left edge. This is physically impossible.

Zero padding assumes that after the edge, the image is zero. Non-zero pixels at the right edges of the image do not influence the left edge after convolution. This is the physically plausible solution. However, the zero padded convolution is more expensive to calculate. We either have to calculate the convolution in image space, which is too expensive for large kernels, or apply the FFT on a zero-padded image. Either way, it is more expensive than the circular convolution.

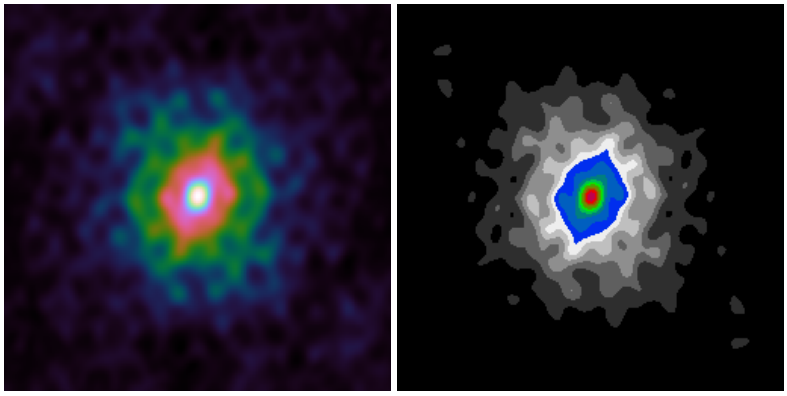
In designing a deconvolution algorithm, we have the choice between the circular and the zero-padded convolution scheme. Circular convolution is more efficient to calculate, while zero-padded convolution is closer to the reality. Both choices are possible. Some implementations leave this choice to the user [33]. We decide on using the zero-padded convolution. This choice influences how we calculate the Lipschitz and gradients efficiently.

4.3.2 Efficient calculation of the Lipschitz constants

In each iteration, we need the Lipschitz constant of the current pixel. I.e. we need the inner product $\langle PSF_{location}, PSF_{location}$ for every pixel. We can pre-calculate the Lipschitz constant before we run the serial coordinate descent algorithm. The naive way to calculate the Lipschitz constant for every pixel results

in quadratic runtime(each inner product costs us $O(n)$ operations, and we do it for all n pixels). But this is not necessary. We can pre-calculate the Lipschitz constant for every pixel in linear time.

Figure 8a shows the PSF shifted to a pixel location. The Lipschitz constant is by squaring all values of Figure 8a and summing up the values. Or in another way: We sum up all the squared values of the PSF inside a specific rectangle. All that changes for a Lipschitz calculation between different pixels is the specific rectangle.



(a) Shifted PSF. *(b) Sum of squared values.*

Figure 8: Comparison of the two convolution schemes.

This can be exploited with a scan algorithm: We first calculate the result of every rectangle we can draw from the origin, up to some pixel value. We end up with an array we call $scan[,]$. It is the same size as the PSF, but contains the sum of squares inside a specific rectangle.

```
var scan = new double[,];
for (i in (0, PSF.Length(0))
  for (j in (0, PSF.Length(1))
    var iBefore = scan[i - 1, j];
    var jBefore = scan[i, j - 1];
    var ijBefore = scan[i - 1, j - 1];
    var current = PSF[i, j] * PSF[i, j];
    scan[i, j] = current + iBefore + jBefore - ijBefore;
```

Every Lipschitz constant can be now calculated by combining the sums of different rectangles. Our example is shown in Figure 8b. We start with the total sum of all values, and subtract two rectangles. Because the subtractions overlap, we need to add the third rectangle again. we take the total value. In short, we can calculate each Lipschitz constant by at most 4 lookups in the $scan[,]$ array.

4.3.3 Using a map of gradients

For an efficient greedy strategy, we need to know the gradient for each pixel. We show how to calculate the initial map of gradients in linearithmic time($O(nlogn)$ and how to update it directly after a change in the reconstructed image x. As we will see, we can use a map of gradients and drop the residual calculation from the algorithm. The gradient map implicitly contains the information of the residuals.

Efficient calculation

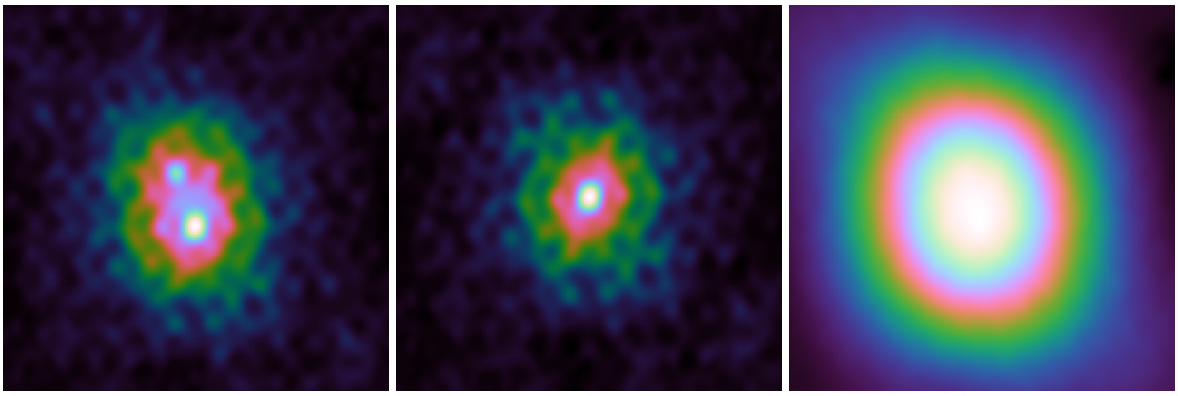
Calculating the gradient for each pixel results again in a quadratic runtime. We need to calculate $\langle residuals, PSF_{location} \rangle$ for every pixel (Similarly to the Lipschitz constants, each inner product costs us $O(n)$ operations, and we do it for all n pixels). But we can use the FFT to calculate the map of gradients in linearithmic time.

Note that the inner product is actually a correlation: We correlate the PSF with the residuals. The correlation and the convolution are related. The convolution is simply a correlation with a flipped kernel. This means we

can use the FFT to efficiently calculate the correlation of the residuals and the PSF:

$$\begin{aligned} psfFlipped &= FlipUD(FlipLR(PSF)) \\ gradients &= iFFT(FFT(residuals) * FFT(psfFlipped)) \end{aligned} \tag{4.7}$$

A convolution in image space is a multiplication in Fourier space. This fact can also be exploited for the correlation by flipping the PSF. Since the FFT takes linearithmic time $O(nlogn)$ to compute, the overall operation also takes us linearithmic time. The operation is shown in Figure 9.



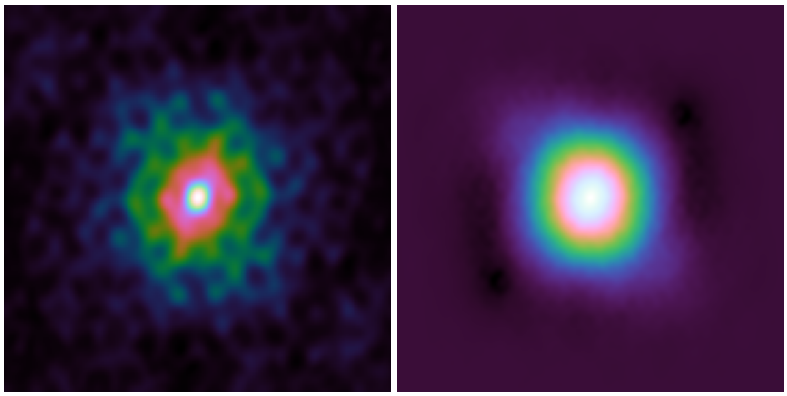
(a) Dirty Image. *(b) Point Spread Function.* *(c) Gradient map.*

Figure 9: Example of the gradient calculation.

Direct update of gradient map

Thanks to the FFT, we can efficiently calculate the map of gradients at the start of the serial coordinate descent algorithm. After we update a pixel in the reconstruction x, the map changes. We could repeat the correlation in the Fourier space from equation (4.7) at each iteration. But this is wasteful. We can update the map of gradients directly.

Note that if we add pixel in the reconstruction x, we subtract the PSF (multiplied with the pixel value) from the specific location in the residuals. The gradient map is then calculated by again correlating the PSF with the residuals. We update the residuals by subtracting the PSF at the correct location. And we update the gradient map by subtracting the PSF correlated with itself at the correct position ($PSF \star PSF$). In our simulated example, the PSF and the gradient update map is shown in Figure 10b.



(a) Point Spread Function. *(b) Gradient update: $(PSF \star PSF)$.*

Figure 10: Example problem with two point sources.

This means we can update the gradient map directly with the product of $(PSF \star PSF)$. We can simply shift $(PSF \star PSF)$ at the correct pixel location and subtract it from the gradient map directly.

There is one issue though: We use zero padded convolution. The PSF at the edges of the image is masked. This means that the product $(PSF \star PSF)$ actually changes with the pixel location. If we update a pixel in the corner of the image, the actual update $(PSF \star PSF)$ at that location looks different than what was shown in Figure 10b.

The exact update is again expensive to calculate. We need to correlate the PSF with itself for every pixel location. This is as repeating equation (4.7) in each iteration (re-calculating the PSF correlation with the residuals in each iteration). However, the exact gradient update only changes significantly at the edges of the image, when large parts of the PSF are masked by the edges. Otherwise the difference between the exact update and simply shifting $(PSF \star PSF)$ at the pixel location is small.

This is the reason why we chose to accept that the gradient update is only an approximation. The approximation only becomes inaccurate at the edges, and we use the algorithm in the major cycle framework: Every major cycle removes any inaccuracies we introduced during the previous cycle. This may potentially lead to more major cycles until the algorithm converge to the solution. But in practice the serial coordinate descent algorithm did not need more major cycles than CLEAN. We compare CLEAN and the serial coordinate descent algorithm on a real-world observation in Section 6.

4.4 Efficient implementation pseudo-code

Putting it all together

No more residual, use gradient map. Pre calculate lipschitz constants with a scan algorithm

putting it all together

```
dirty = IFFT(GridVisibilities(visibilities))
residualsPadded = ZeroPadding(dirty)

psfPadded = ZeroPadding(PSF)
psfPadded = FlipUD(FlipLR(psfPadded))
gradientUpdate = iFFT(FFT(ZeroPadding(PSF)) * FFT(psfPadded))

x = new Array[,]
gradientsMap = iFFT(FFT(residualsPadded) * FFT(psfPadded))
lipschitzMap = CalcLipschitz(PSF)

objectiveValue = 0.5* Sum(residuals * residuals) + ElasticNet(x)

do
  oldObjectiveValue = objectiveValue

  //Step 1: Search pixel
  maxAbsDiff = 0
  maxDiff = 0
  pixelLocation = (-1, -1)
  for(i in Range(0, dirty.Length(0))
    for(j in Range(0, dirty.Length(1))
      oldValue = x[i, j]
      tmp = gradientsMap[i, j] + oldValue * lipschitzMap[i, j]
      optimalValue = Max(tmp - lambda*alpha) / (lipschitz[i, j] + (1 - alpha)*lambda
          )
      diff = optimalValue - oldValue

```

```
28        if(maxAbsDiff < Abs(diff))
29          maxAbsDiff = Abs(diff)
30          maxDiff = diff
31          pixelLocation = (i, j)
32
33    //Step 2: Optimize pixel. Now all that is left is to add the maximum value at the
          correct location
34    x[pixelLocation] += maxDiff
35
36    //housekeeping
37    shiftedUpdate = Shift(gradientUpdate, pixelLocation)
38    gradientMap = gradientMap - shiftedUpdate * maxDiff
39  while maxAbsDiff  < epsilon
```

Lookups. No residuals, implicitly contained in gradientMaps. We do not need the PSF directly, we need the product of $PSF \star PSF$, which is the gradient update

Note on objective value calculation. We can skip it generally. Optimize until all pixels are below an epsilon.

4.5 GPU implementation

We implemented the serial coordinate descent algorithm on the GPU. It is implemented in C# .netcore with ILGPU[34].

What is a kernel. A kernel in the context of GPU architecture is a small program that runs in parallel on several instances on the GPU.

Our serial coordinate descent algorithm consists of two steps. Step 1, find the best pixel to optimize, and step 2: Optimize pixel and update the reconstruction x and the gradient map. The ILGPU implementation of the Serial coordinate descent algorithm uses three kernels. Kernel 1 is equivalent to step 1. Kernel 2 updates the reconstruction x and kernel 3 updates the gradient map.

Kernel 2 and 3 are straight forward to implement. The implementation of kernel 1, searching for the best pixel, is more interesting. Essentially, the first step in the serial coordinate descent algorithm is a max-reduce. We want to find the pixel with the maximum absolute step we can take in this iteration. Reduce operations can be efficiently implemented with a warp shuffle[35].

It was implemented with warp shuffle, but a simple atomic max operation was faster in practice. This leads to the following kernel:

```
1 MaxPixelKernel(x, gradienstMap, lipschitz, location)
2   oldValue = x[location]
3   tmp = gradientsMap[location] + oldValue * lipschitzMap[location]
4   optimalValue = Max(tmp - lambda*alpha) / (lipschitz[location] + (1 - alpha)*lambda
        )
5   diff = optimalValue - oldValue
6   currentPixel = (absDiff = Abs(diff), diff = diff, location = location)
7
8   AtomicMax(maxPixel, currentPixel)
```

Synchronization between the kernels. It is a serial coordinate descent method. We need to wait for all kernels to finish before we can continue.

```
1 do
2   //Step 1: Search pixel
3   maxPixel = (absDiff = 0, diff = 0, location = (-1, -1))
```

```
    ExecuteMaxPixelKernel(x, gradienstMap, lipschitz)
    SynchronizeKernels()

    //Step 2: Optimize
    ExecuteUpdateXKernel(x, maxPixel.diff, maxPixel.location)
    ExecuteUpdateGradientsKernel(gradientsMap, gradientUpdate, maxPixel.diff, maxPixel
        .location)
    SynchronizeKernels()

while maxPixel.absDiff < epsilon
```

ILGPU implementation finished.

4.6 Distributed implementaion MPI

How we distribute it with MPI.

Message pasing interface (MPI)

We split up the image and the gradient map into patches.

Each simply updates its patch.

MPI Allreduce

4.7 Serial coordinate descent and similarities to the CLEAN algorithm

When we look back at the CLEAN algorithm, we start to see similarities to the serial coordinate descent algorithm.

CLEAN finds the maximum in the residual image. Serial Coordinate descent finds the maximum in the gradient map.

CLEAN then removes a fixed fraction of the PSF at that point. Serial coordinate descent uses the Lipschitz constant, and subtracts the PSF correlated with itself at that point.

The main difference is that serial coordinate descent calculates the gradient, and CLEAN does not. CLEAN is more a matching pursuit algorithm. But the structure is the same. We can use the serial coordinate descent GPU and MPI implementation. With a few tweaks, we would arrive at the CLEAN algorithm.

By accident, we also found a GPU and MPI implementation for standard CLEAN.

Multi-scale CLEAN is what is used. Difficulty with MPI implementation because of a distributed convolution.

5 PSF approximation for parallel and distributed deconvolution

For a distributed deconvolution, we would like to deconvolve the image with as little communication as possible. This largely depends on the size of the PSF when compared to the overall image. If the PSF is for example $\frac{1}{16}$ of the total image size, we have patches of the image which are completely independent of each other. Sadly, this is not true for radio interferometers. The PSF is generally the same size as the image. We cannot deconvolve any part of the image independently of each other.

However, we have two effects of modern radio interferometers, that produce an "approximately" smaller PSF: First, we have an increasing number of visibilities. They create a PSF that increasingly resembles a Gaussian function in the center, and the rest approaches zero. And secondly, we reconstruct images with a wide field-of-view. Although the PSF is not zero the further away we move from the center, its values approach zero.

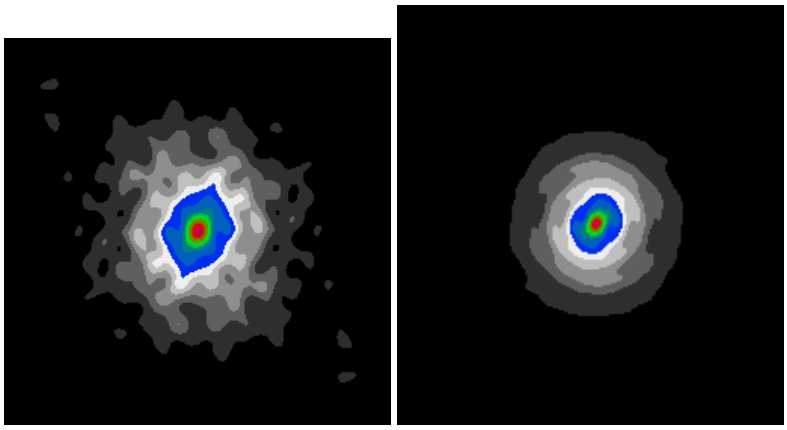


Figure 11: PSF arising from an increasing number of visibilities.

In short, with an ever increasing number of visibilities and field-of-view, the influence of far-away image sections become negligible. We can approximate the deconvolution with a fraction of the true PSF. To our knowledge, we are the first to propose such approximation methods. In this Section, we present our approximation methods. In Section 6.3, we empirically demonstrate the validity of our approximations on a real-world MeerKAT observation. In Section 7, we show more sophisticated coordinate descent methods that can exploit the smaller PSF.

5.1 Intuition for approximating the PSF

Our basic coordinate descent algorithm chooses a pixel to minimize, calculates its gradient and descends in that direction. The gradient calculation reduces itself to a correlation of the residuals with the PSF at the pixel location. In other words, we need the PSF to calculate a gradient. If we only use parts of the PSF for the calculation, we essentially approximate the gradient for the pixel. Because the PSF only has significant non-zero values in the center, we should be able to ignore most of the values and still have an adequate gradient approximation.

Furthermore, our basic coordinate descent algorithm reconstructs inside the Major/Minor cycle framework. The framework is designed to handle only an approximate PSF in the deconvolution (Remember: the w-term changes the PSF depending on the position in the image). The framework may be able to deal with further PSF approximations, namely with a PSF that is reduced in size, which makes the distributed deconvolution simpler.

The approximation methods essentially work by "cutting off" the less significant PSF values and only use a rectangle around the center. If we cut off the PSF by a factor of $\frac{1}{4}$ (If the PSF is 1024^2 pixels in size, we only use a rectangle of 256^2 of the center), we get pixels in the reconstructions that are not influenced

by each other. Starting from a cut-off factor of $\frac{1}{16}$, we can split the image into eight patches, four of which are independent of each other. This would allow us to run up to four basic coordinate descent algorithms in parallel, simplifying the distribution.

Indeed, it is possible to approximate the PSF with only a fraction of its true size, as we will demonstrate empirically in Section 6.3. But an approximation of the PSF may lead to other problems in the reconstruction:

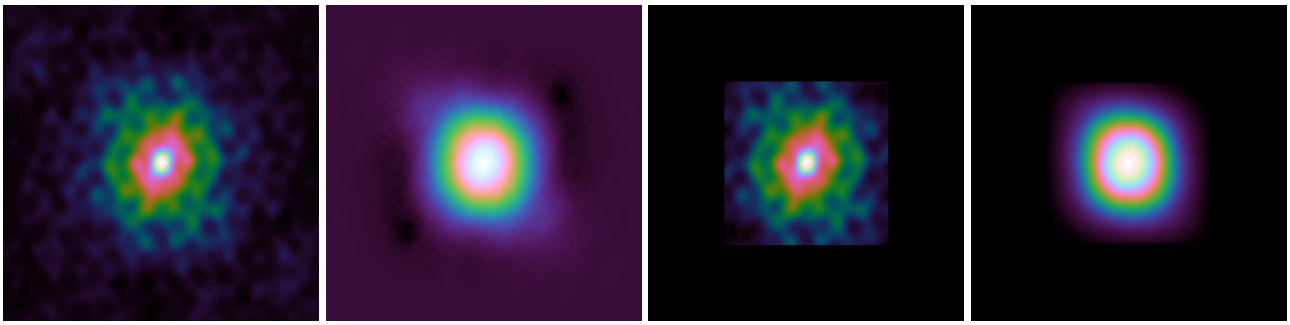
- Needs additional Major cycles to converge.
- Slow down convergence speed of coordinate descent.
- Not guaranteed to converge to the same image.

The Major cycle corrects the errors the approximate PSF introduces. The more inaccurate the PSF is, the more Major cycles we may need to converge. As we already discussed, a Major cycle is an expensive operation. Our PSF approximation should lead to as few (if any) additional Major cycles.

For the basic coordinate descent algorithm, an approximate PSF may slow down the convergence speed. In each iteration, the basic algorithm finds the optimal value for the current pixel. With an approximate PSF, we may need several iterations on the same pixel (over several major cycles) until we arrive at the same value. In short, an approximate PSF can slow down the convergence speed of coordinate descent.

Depending on how we approximate the PSF, we may not have any guarantee that we arrive at the same result. We developed two approximation methods: Method 1 uses an approximation in the gradient update step of coordinate descent. Method 2 solves an approximate deconvolution problem with only a fraction of the PSF. Only method 1 is guaranteed to converge to the same solution (with enough major cycles), but is slower to converge than method 2. Depending on the method we use, we can remedy some of the problems that approximating the PSF introduces. But there seems to be a trade-off to be made. We have not found a method that works best in every aspect.

5.2 Method 1: Approximate gradient update



(a) PSF *(b) Gradient update (c)* $cut(PSF)$ *(d) Approx. update*

Figure 12: Approximation of gradient update.

The basic coordinate descent method first correlates the residuals with the PSF. It pre-calculates the gradient for each pixel. Then, in each iteration, it directly updates the map of gradients with the product of $PSF \star PSF$ (the PSF correlated with itself). In this approximation method, we start from the same pre-calculated map of gradients, but use an approximate update. The first coordinate descent iteration of this approximation method is identical to coordinate descent with the full PSF. With each coordinate descent iterations, the gradient map becomes more inaccurate. But with enough major cycles, this method converges to the same result as when the full PSF is used.

The question is, how do we approximate the product of $PSF \star PSF$. As we have seen before, the product of $PSF \star PSF$ also approaches zero away from the center (an example was shown in Figure 10 in Section

??). A naive way to approximate the update step is to cut off the insignificant value and only use a rectangle of the center, which is a fraction of the total image size. For example: An image of size 1024^2 also has a PSF the size of 1024^2. The product $PSF \star PSF$ actually has the size of 2048^2 pixels due to the correlation. We can try to approximate the product by only using the center rectangle $\frac{1}{8}$ of the total size, (256^2 pixels). This approximation works, but leads to artifacts during deconvolution: The image will be reconstructed in visible "blocks" which are the size of the center fraction we use.

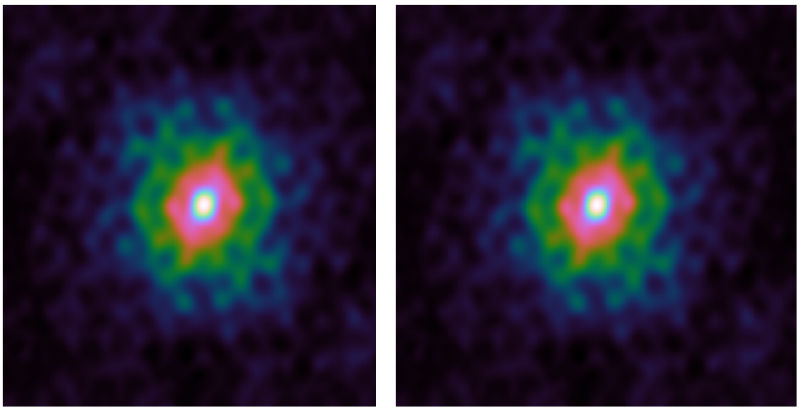
We use the approximation shown in equation (5.1), where $cut()$ is the function that cuts away everything but the center rectangle of the PSF. This is a better approximation than cutting the product of $PSF \star PSF$ directly, and leads to faster convergence.

$$PSF \star PSF \approx cut(PSF) \star cut(PSF) \tag{5.1}$$

The reason why (5.1) is a better approximation lies in the reason why we update the gradients with the product of $PSF \star PSF$ in the first place: It is the combination of two separate operations, removing the PSF from the residuals at the current position, and recalculating the correlation with the PSF. If we cut away parts of the product $PSF \star PSF$ directly, we implicitly update the residuals with a different PSF. But when we approximate the product by (5.1), we ensure that the implicit removal of the PSF from the residuals is equal to $cut(PSF)$.

In our implementation, we use one more trick to improve the approximation: we scale the product of $cut(PSF) \star cut(PSF)$ to have the same maximum as the original product $PSF \star PSF$. The approximation has a lower maximum value than the original. Over several coordinate descent iterations, we run into the danger of overestimating the pixel values. By scaling the product of $cut(PSF) \star cut(PSF)$ to the same maximum, we end up with a better approximation of the true gradient update.

5.3 Method 2:Approximate deconvolution



(a) PSF on log scale *(b) PSF Approximation*

Figure 13: Approximate deconvolution with a fraction of the PSF.

The main problem with Method 1 is that the map of gradients becomes less accurate with more coordinate descent iterations. This method sovles this problem by using an approximate deconvolution instead, but it loses the guarantee to converge to the same result as the full PSF.

This method cuts off the insignificant part of the PSF, and only uses the center rectangle of it for the whole deconvolution problem. As such, the coordinate descent method solves an approximate deconvolution problem

shown in (5.2).

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * cut(PSF)\right\|_2^2 + \lambda ElasticNet(x) \tag{5.2}$$

In essence, it uses the same basic coordinate descent method, but ignore large parts of the PSF. It pre-calculates the gradient map by correlating the residual image with $cut(PSF)$, and update the gradient map with the product of $cut(PSF) \star cut(PSF)$. The main difference between approximate gradient update and approximate deconvolution is: In approximate gradient update starts with the same gradient map as the original. The approximate deconvolution does not. With this method, the gradient map does not become more inaccurate with more coordinate descent iterations.

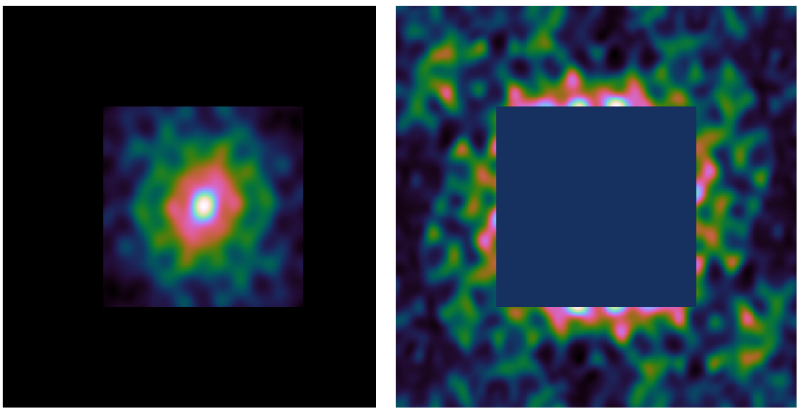
The approximate deconvolution objective (5.2) is not guaranteed to "point" to the same solution as the original. It may in reality point to a very different solution. But thanks to the Major cycle, the image retrieved by optimizing (5.2) is always "close" to the original solution.

Nevertheless, this approximation method introduces an error in the final reconstructed image. The obvious error it introduces is it under-estimates the true pixel values. Pre-calculating the gradient map with $cut(PSF)$ under-estimates gradient magnitudes, and by extend the pixel values.

To combat the under-estimation of pixel values, we reduce the regularization parameter λ for the approximate deconvolution problem. Since we cut off parts of the PSF, we also reduce the Lipschitz constant (sum of the squared PSF values) used in the approximate deconvolution. We reduce the λ parameter by the same factor that the Lipschitz constant gets reduced. This ensures that the approximate deconvolution and the original deconvolution arrive at the same pixel value for a point source. But it does not completely remove the issue for extended emissions.

5.4 Major Cycle convergence and implicit path regularization

In the two presented methods, we use a fraction of the PSF to approximate the deconvolution problem. Both methods rely on the Major Cycle to periodically reset the gradient map. By using only a fraction of the PSF, the approximate deconvolution leaves parts of the PSF in the gradient map. Figure 14 shows the fraction of the PSF that get included in the deconvolution, and "sidelobes" which get ignored.



(a) PSF cutoff *(b) Sidelobes*

Figure 14: Max sidelobe PSF.

After a number of serial coordinate descent iterations, we run into the danger of deconvolving the leftovers of the PSF approximation. In that case, the serial coordinate descent algorithm adds spurious point sources to the image. After a major cycle, we can detect them as spurious and remove them again. Degree to how bad this is depends on the maximum absolute value we cut off in the approximation, the maximum absolute value

in Figure 14b. With an aggressive approximation, we may end up oscillating from major cycle to major cycle: Adding spurious pixels and removing them again in the next, just to add the same spurious pixels in the one after.

But we can estimate at what point we are likely to add spurious pixels. This lets us estimate a minimum λ parameter for the elastic net regularization. At this major cycle iteration, we cannot go below the minimum lambda. The next major cycle, the maximum residual is lower, and so is the minimum λ

This is known as a path regularization. We start with a stronger regularization and continually reduce λ until we reached the desired value. We have intermediate results in each major cycle. Coordinate descent methods tend to be faster from a warm start, when we start from an intermediate result [36].

In this work we use the path regularization mainly for convergence of the major cycle.

How we estimate the minimum λ parameter. Imagine the interferometer only observed a point source in the center of the image. In that case, after the first serial coordinate descent iteration, our residuals image looks exactly like the Figure 14b. The serial coordinate descent algorithm should not take another step, or it adds spurious pixels. In other words, the regularization has to suppress all the values in Figure 14b

Remember the elastic net regularization: It is a mixture of the L1 and L2 norm. The L1 norm shrinks the pixel values, while the L2 norm divides them. We need the L1 norm to shrink away the values in Figure 14b.

$$\begin{aligned} gradients &= residuals \star PSF \\ maxSidelobe &= Max(gradients) * Max(Abs(Sidelobe(PSF)))) \\ \lambda_{cycle} &= \frac{maxSidelobe}{\alpha} \end{aligned} \tag{5.3}$$

This estimate is a minimum. Meaning it is possible that we still add spurious pixels. For example when the sidelobes of two point sources overlap.

The estimate however only considers point sources. For extended emissions, the estimate is too low. Extended emissions are a group of non-zero pixels. Beause they are next to each other, their sidelobes overlap and get amplified.

An estimate considering extended emissions.

$$\begin{aligned} psfSidelobe &= Max(PSF - cut(PSF)) \\ gradients &= residuals \star PSF \\ correction &= Max(1, \frac{Max(gradients)}{Max(residuals) * lipschitz}) \\ maxSidelobe &= Max(gradients) * Max(psfSidelobe) * correction \\ \lambda_{cycle} &= \frac{maxSidelobe}{\alpha} \end{aligned} \tag{5.4}$$

Only considers the ”maximum” extended emission. Extended emissions tend to contain the largest pixel value in the residual image. Because of the convolution. The correction factor gets reduced the closer the maximum pixel in the residuals. Over several major cycles, the correction factor and the $lambda_{cycle}$ become smaller. Helps convergence.

But this is not true for extended emissions.

he serial deconvolution algorithm calculates the gradient map (by correlating the PSF with the residuals).

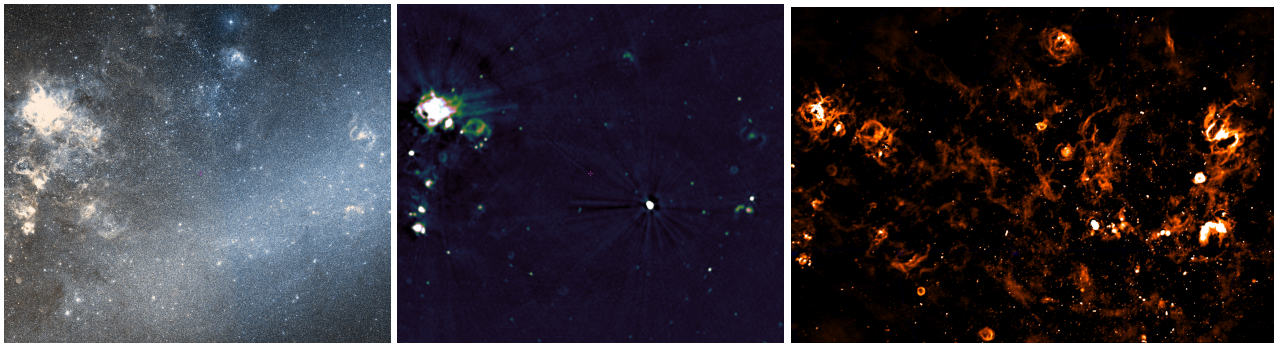
Residuals will be left.

6 Tests on MeerKAT LMC observation

The Large Magellanic Cloud (LMC) is a galaxy is the second or third closest galaxy to the Milky Way. Figure 15 shows the LMC in both optical and radio wavelenghts. The radio wavelengths was observed by the VLA radio interferometer[37] at 843MHz. In the optical wavelengths, the abundance of stars are clearly visible. The LMC is close enough to earth for individual stars are visible. But it also contains a large number of supernova remnants, gas clouds, and other extended emissions, which shine bright in the radio wavelengths.

The LMC is a region with a large number of sources at different brightness. In the lower-right quadrant of the radio-image 15b, we see the bright emission of the supernova remnant N132D, the brightest radio source in the LMC. But around the N132D are faint emissions from gas-clouds. This means faint emissions may get lost next to N132D. We need a deconvolution algorithm to uncover these faint emissions.

We received a MeerKAT observation of the LMC from SARAO for the purpose of algorithm testing. At the time of writing, the MeerKAT instrument is still being tested. The observation is only representative in the data volume. The observation is calibrated, and averaged down in both frequency and time. The averaging reduces both the disk space and the runtime costs of the gridding step. Nevertheless, the observation takes up over 80 GB of disk space (roughly $\frac{1}{30}$ of the original data). A CLEAN reconstruction of the calibrated observation is shown in Figure 15c.



(a) Optical wavelength *(b) Radio wavelength at 843MHz.* *(c) Wide band radio image by MeerKAT.*

Figure 15: Section of the Large Magellanic Cloud (LMC)

The MeerKAT observation covers a wide band of radio frequencies. The lowest frequency in the MeerKAT observation is 894 MHz, and the highest frequency is 1658 Mhz. Imaging the whole frequency band requires a wide band deconvolution algorithm. In wide band imaging, several images at different frequencies get deconvolved as an image cube. Wide band imaging again multiplies the amount of work that has to be done for reconstruction, as now we cannot deconvolve a single image, but have to deal with a whole image cube.

Wide band imaging is not possible within the time frame of this project. We take a narrow band subset of 5 channels from the original data (ranging from 1084 to 1088 MHz, about 1 Gb in size) for reconstruction. We also reduce the field-of-view to a more manageable section. Figure 16 shows the LMC image section we are using together with a CLEAN reconstruction of the narrow band data.

At the center of our image section 16 we see the N132D supernova remnant. We partially see the faint extended emissions, although they are close to the noise level. This is known as a high-dynamic range reconstruction. We have strong radio sources mixed together with faint emissions, which are only marginally above the noise level of the image.

The total field-of-view of our image section is roughly 1.3 degrees(or 4600 arc seconds). Our reconstruction has 3072^2 pixel with a resolution of 1.5 arc seconds per pixel. this is still a wide field-of-view reconstruction problem. We have to account for the effects of the w-term to achieve a high-dynamic range reconstruction.

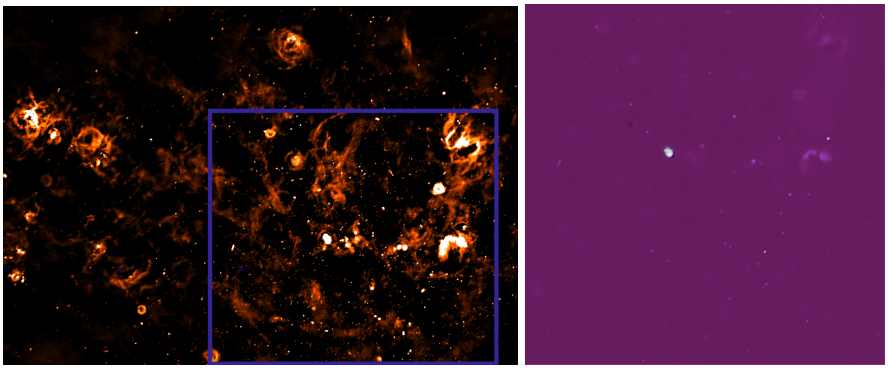


Figure 16: Narrow band image section used.

In our test reconstruction, we need to account for w-term correction and high-dynamic range. We have excluded wide-band imaging as not feasible within the time frame of this project. In Section 6.1 we compare the reconstructions of CLEAN with our coordinate descent based algorithm on the LMC observation. The next Section 6.2 presents the speedup we achieve with coordinate descent by using our distributed or GPU-accelerated implementations.

In Section 6.3 we show the core result of this project. Namely what effect has an approximate PSF on the deconvolution problem and whether we can use it to further distribute the problem. The answer to that question is affirmative: We can approximate the PSF, and we can exploit it to further distribute the deconvolution. But we need more sophisticated coordinate descent algorithms to fully benefit from it.

6.1 Comparison with CLEAN reconstructions

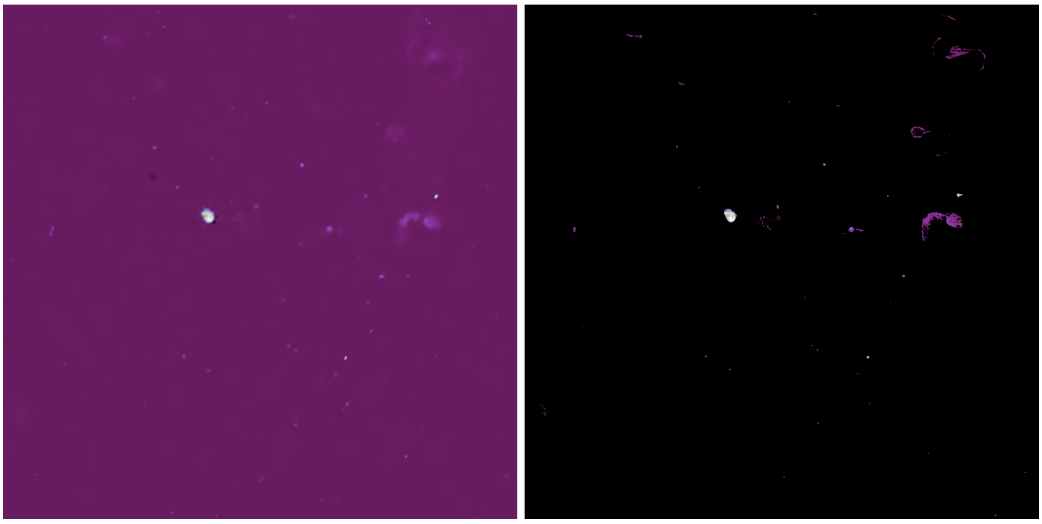
We use the WSCLEAN [38] implementation of multi-scale CLEAN. We compare our coordinate descent reconstruction with two CLEAN reconstructions, oe with naturally weighted visibilities and one with briggs weighted visibilities.

There are three main visibility weighting scheme for the gridder that lead to different PSFs from the same measurements: Natural, uniform, and Briggs[39]. Natural weighting scheme leads to an image with a lower noise level, but a wider PSF. Uniform weighting leads to a higher noise level, but to a PSF whgich is more concentrated around a single pixel. Briggs weighting is a scheme combines the best from both worlds, receiving an image with acceptable noise level while getting a more concentrated PSF. As such it is widely used in radio astronomy image reconstruction. Our gridder implements the natural weighting scheme only. Nevertheless our coordinate descent algorithm is able to retrieve structures similar to the briggs-weighted multi-scale CLEAN reconstruction, even though coordinate descent has to work with a wider PSF.

Figure 17 shows the reconstruction of both briggs-weighted multi-scale CLEAN and the naturally weighted coordinate descent reconstruction. CLEAN used 6 major cycles and 14 thousand minor cycle iterations. Our coordinate descent implementation converged after 5 major cycles and needed 100 thousand iterations to converge.

Coordinate descent needs a large number of iterations to converge when compared to multi-scale CLEAN. Note that a coordinate descent iteration is cheaper to compute than one iteration of multi-scale CLEAN. Also note that because we are searching for structures close to the noise level of the image, coordinate descent often adds pixels belonging to the noise in one major cycle, just to remove them in the next one. Path regularization[36] can combat this problem, and gets further investigated in the following Section 6.3.

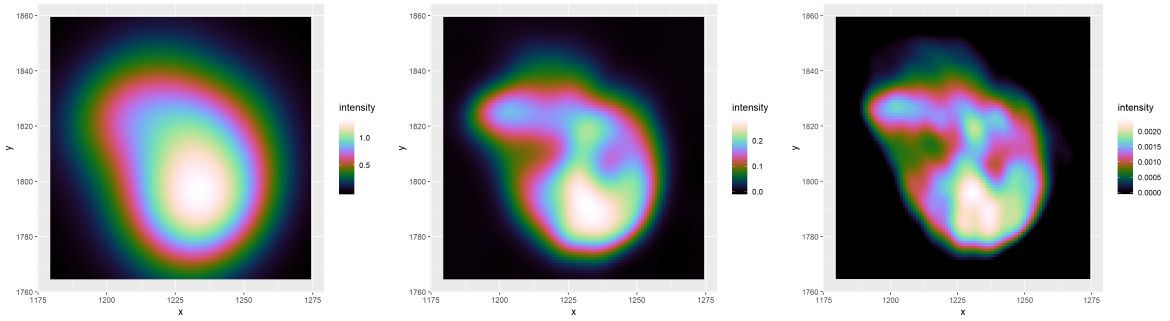
Both algorithms detect the three extended emissions at the right side of the image. They detect various point sources at the same location. Coordinate descent and multi-scale CLEAN arrive at a roughly similar



(a) Briggs weighted multi-scale CLEAN. *(b) Naturally weighted coordinate descent.*

Figure 17: Comparison of the whole image

result. Coordinate descent detects similar structures in the N132 supernova remnant, as the briggs-weighted CLEAN, but also includes calibration errors in its reconstruction of the faint extended emissions.



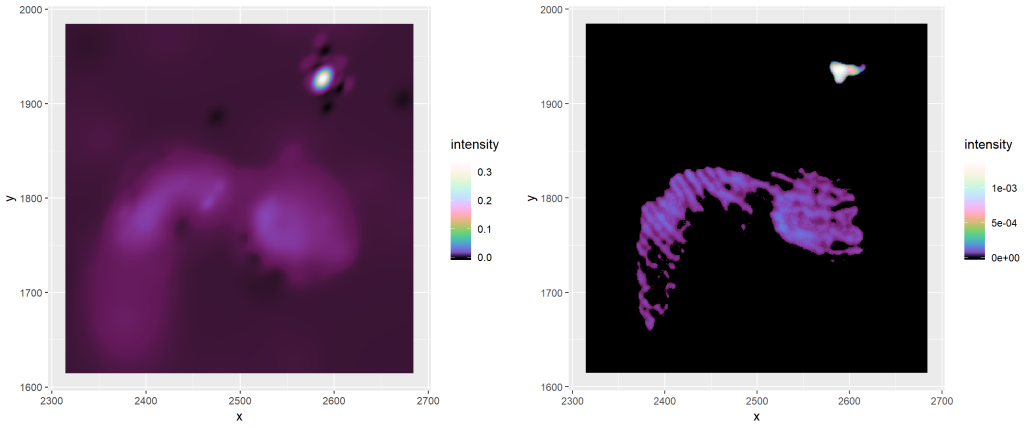
(a) CLEAN Natural weighting. *(b) CLEAN Briggs weighting.* *(c) CD Natural weighting.*

Figure 18: N132 comparison

Figure 18 compares the naturally-weighted CLEAN, briggs CLEAN and coordinate descent on the N132 supernova remnant. The naturally-weighted CLEAN and coordinate descent use the same PSF for the deconvolution. But coordinate descent finds structures in N132 similar to the briggs-weighted CLEAN. Coordinate descent arrived at a plausible higher-resolved reconstruction of N132.

Calibration errors on the other hand negatively influence the coordinate descent reconstruction. Figure 19 shows a cutout of the right hand section of the reconstruction, where a faint extended emission is next to a point source with calibration errors. Multi-scale CLEAN is able to differentiate between the ”ripples” from the calibration error, and the signal from the extended emission. Coordinate descent with the elastic net regularization includes the ripples into the reconstructed image.

The only way to exclude the ripples from the reconstruction is to increase the regularization parameter λ., such as no pixel gets included which is not above the noise level + calibration error in the image. However, that would lead to other sources being ”regularized away” in other regions of the image, which do not have a severe calibration error close by.



(a) Briggs CLEAN *(b) Coordinate Descent*

Figure 19: Influence of calibration errors

6.2 Coordinate descent acceleration with MPI or GPU

Describe hardware

Distributed with MPI

GPU implementation

Measurement of the speedup.

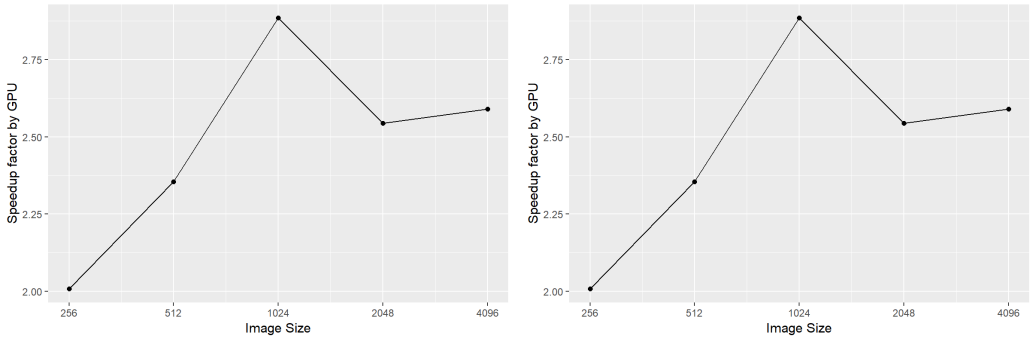


Figure 20: Speedup by using MPI or GPU acceleration

We cannot use MPI combined with the GPU. The MPI implementation uses a communication step in each coordinate descent iteration (communicating which pixel to optimize with MPI Allreduce).

6.3 Effect of approximating the PSF

As we described in Section 5, the PSF for deconvolution is as big as the image. For wide field-of-view observations of MeerKAT, the PSF is approximately a gaussian with decreasing pixel values the further we move from the center. Most of the values in the PSF are close to zero. The question is, what effect has an approximate deconvolution with a smaller PSF? If we can approximate the deconvolution with a small enough PSF, we can solve patches of the image independently of each other. However, the PSF approximation may need more major cycles to converge.

The effect of approximating the PSF are not clear. We know that thanks to the w-term in the visibilities, the PSF is not constant over the image. We already need several major cycles to converge. With a good

approximation of the PSF, we may speed up the individual iterations of coordinate descent without needing more major cycle.

We presented two methods to approximating the PSF for the deconvolution in Section 5. Method 1 updates only a fraction of the gradients, and Method 2 uses a fraction of the PSF for deconvolution. We test both methods on the LMC data and explore what effects the approximations have on the reconstruction.

6.3.1 Method 1: Approximate gradient update

Our coordinate descent method updates the map of gradients after each iteration. Method 1 starts with the same map of gradients as the original, but then only updates a fraction of the gradients in each iteration. It updates a rectangle of the most significant gradients.With each coordinate descent iteration, the map of gradients gets less accurate. Because we do an approximate update of gradients, this method should converge to the same result as the original with enough major cycles.

At the beginning of each major cycle, we calculate the objective value of the current solution. We compare the objective value and the wall-clock time of the original and the approximate gradient update. This is a minimization problem, meaning the lowest objective is the most accurate reconstruction (according to the elastic net regularization).

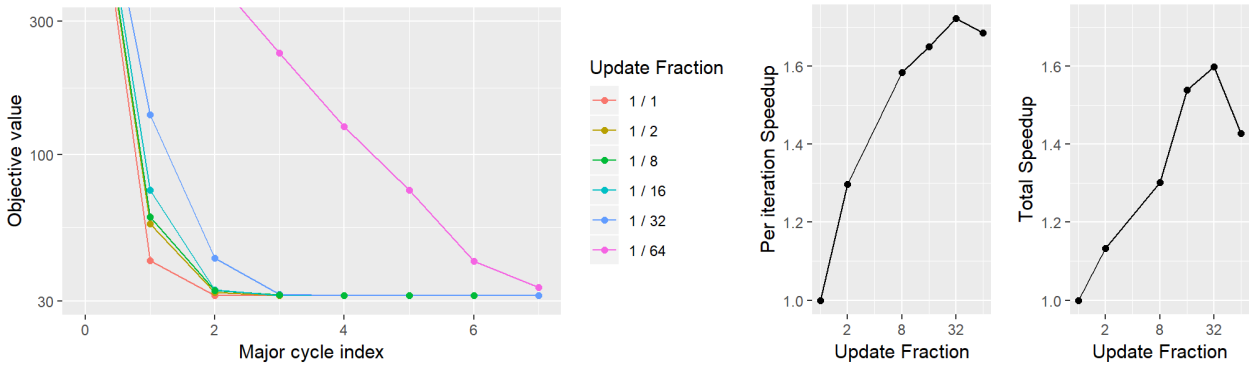


Figure 21: Effect of only updating a fraction of the gradients.

The smallest fraction of gradient updater, for which coordinate still converges is $\frac{1}{64}$ of the total update. Meaning, we can update only a rectangle of 48^2 pixels, or an image patch of 72 arc-seconds in size. However, it is obvious from the Figure 21 that coordinate descent needs more major cycles to converge with such an extreme approximation.

With less extreme approximations, we also reduce the number of necessary major cycles. The approximation of $\frac{1}{32}$ needs two more major cycle, while all other need the one more as the original method[6]. The objective values of the approximations end within 0.03% of the original(the objective value of the approximations are higher by a factor of 0.0003).

Figure 21 also shows two speedup comparisons. The speedup from the approximation is harder to quantify. For one, each iteration of coordinate descent becomes cheaper, because we only update a small part of the gradient map. This is the per iteration speedup. The second speedup Figure compares the overall time spent in deconvolution. Although each iteration becomes cheaper with the approximation, we may need more iterations to converge. Also, we have an implicit path regularization in the approximation. Therefore, a speedup per coordinate descent iteration does not necessarily lead to an overall speedup.

As we discussed in section In Section 5.4, we have a limit in each major cycle to how far we can trust our approximation. That is why in the first major cycle coordinate descent gets started with a higher λ regularization

[6] 6 cycles to converge, and one extra cycle which is here to measure the final objective

than specified. In each successive major cycle, we reduce the λ parameter until we reach the specified value. This is an implicit form of path regularization, and may help speeding up the convergence rate of coordinate descent in general [36]. In our case, we need path regularization to ensure convergence over several major cycles (Aggressive approximations like $\frac{1}{64}$ start to oscillate over several major cycle). Notice that the overall speedup in Figure 21 is lower than the speedup we get per iteration. This suggests path regularization as implemented here does not speed up convergence rate.

Overall with gradient update approximations give us a speedup factor of around 1.5. Depending on how aggressive we approximate, We need one or two more major cycles than the original coordinate descent deconvolutions. This puts gradient update approximations on par with multi-scale CLEAN in terms of major cycles.

6.3.2 Method 2: Approximate deconvolution

In this method, we use a fraction of the total PSF for deconvolution. This method solves a different deconvolution problem, where the PSF is for example only $\frac{1}{8}$ the size. The downside is that method 2 is not guaranteed to converge to the same optimum. Nevertheless, we solve the approximate deconvolution problem with several different fractions of the PSF and compare how close the approximate solution is to the original.

As before, we measure the true objective value for the approximate solution at the beginning of each major cycle iteration. The Figure 22 compares the approximate deconvolution to the original.

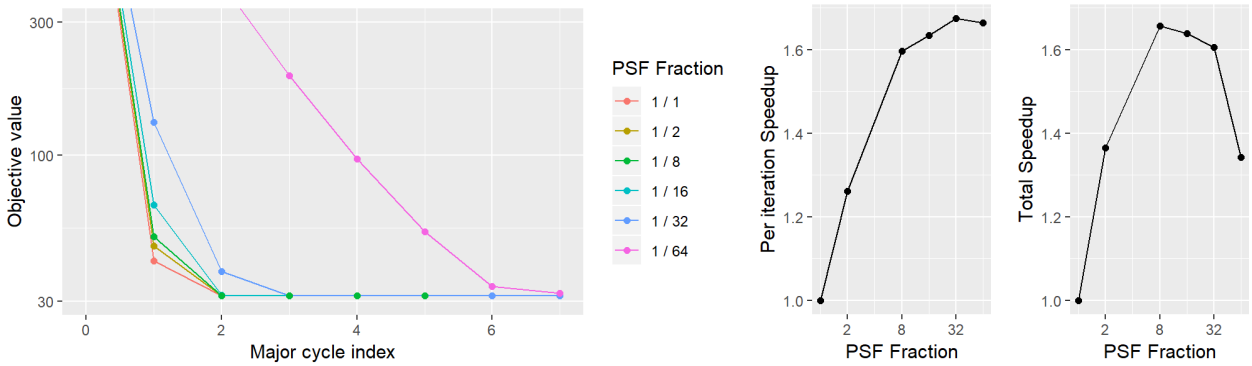


Figure 22: Effect of the L1 and L2 Norm separately.

The performance is similar to the first method at first glance. Both methods converge in a similar manner, with a similar factor of speedup. For the PSF fraction $\frac{1}{64}$ this is largely due to the same path regularization used in this method. For larger PSF's, starting from $\frac{1}{16}$, the path regularization becomes less important with this method, as it only effects the regularization parameter λ of major cycle index 0. Indeed, this method is less likely to oscillate and seems to have a more stable convergence.

The approximate deconvolution converges faster than the first method. At the start of the same major cycle, this method always has a lower objective. However, it is not guaranteed to converge to the same result. This can be seen in the fact that it never reaches the same objective value as the original. The objective value of the approximation is within 0.07% (Factor of 0.0007, roughly twice the factor of the first method). The difference gets more significant, the more extreme we chose the deconvolution approximation.

Nevertheless, the approximate deconvolution converges surprisingly close to the original solution. The question is, is this difference of 0.07% in the objective value significant? The answer to this question depends on what kind of error the approximation introduces in the image. The obvious error is that the approximation chronically under-estimates the pixel values: The maximum pixel value in N132 of the original is 0.0024 Jansky/beam, while the maximum of the $\frac{1}{16}$ approximation is 0.00235 Jansky/beam. The pixel magnitude is

important in the self-calibration regime [29] (When we take the result of the reconstruction and try to improve the calibration).

The under-approximation of pixel values is an error we cannot ignore. One naive remedy is to start with the deconvolution approximation, but switch to the original PSF after a certain number of major cycle. This would give us the guarantee to converge to the same result, but let us use an approximation at the start. We propose another solution: Combining the two approximation methods.

6.3.3 Combination of Method 1 and 2

The two approximation methods have two different shortcomings: Approximate gradient update (method 1) converges more slowly, and converges to the same result as the original. Approximate deconvolution (method 2) converges faster than method 1, but does not converge to the same result. Here, we combine the two methods to remedy the shortcomings: For the first couple of major cycles, we use approximate deconvolution, and then switch to approximate gradient update.

We compare the original, approximate gradient update, approximate deconvolution and our combination in. We use the factor $\frac{1}{16}$ for approximations, meaning we update $\frac{1}{16}$ of the gradients, and do the approximate deconvolution with $\frac{1}{16}$ of the PSF.

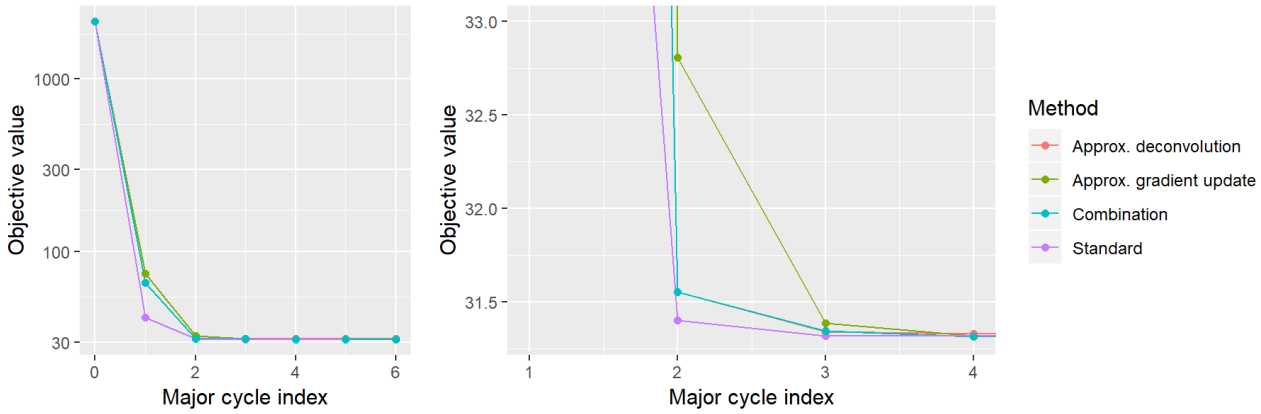


Figure 23: Comparison of the two methods using the fraction $\frac{1}{16}$ of the PSF.

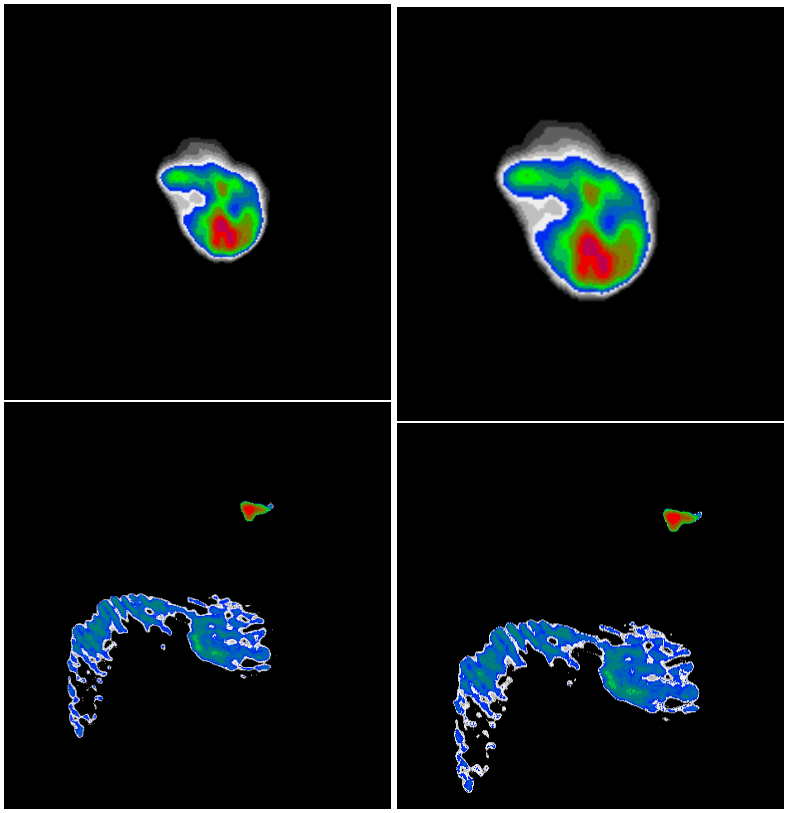
Figure 23

We switch from approximate deconvolution to gradient update after the path regularization has reached the target λ for the first time. In the case of the LMC data, the implementation uses approximate deconvolution for major cycle index 0 and 1, and then switches to approximate gradient update.

Method	Iteration Speedup	Total Speedup
Original	1.00	1.00
(Method 1) Approx. gradient update	1.66	1.55
(Method 2) Approx. deconvolution	1.67	1.68
Combination	1.69	1.68

Speedup. Combination is the fastest overall and leads to the lowest objective function. Actually beats the original by a small margin. Approximation errors are not correlated. We can use Method 1 to remove the approximation errors of Method 2.

Figure 24 compares the original result with the result of the combined approximation. Virtually identical. Same problem with calibration. Only visual difference in the extended emission with calibration errors.



(a) Original CD *(b) Combined approximation*

Figure 24: Image comparison to approximation

Difference is negligible. We can use the approximation and arrive at the same result. Speedup of a factor 1.6
How many major cycles: One more than the original. But same as multi-scale CLEAN.

7 Parallel coordinate descent methods

We demonstrated that the true PSF of the deconvolution problem can be approximated. A smaller PSF can be used, which is only a fraction of the true size. The serial coordinate descent methods, which was used so during this project, achieves a moderate speedup with the PSF approximation. Parallel coordinate descent methods may benefit more from the PSF approximation. For the rest of this work, we develop a parallel coordinate descent deconvolution algorithm. We explain why parallel methods should benefit more from the PSF approximation, and show that they indeed speed up the deconvolution problem.

Introduce the principle of parallel coordinate descent methods, and derrive a parallel coordinate descent deconvolution algorithm.

7.1 From serial to parallel

Parallel coordinate descent methods take several steps at different coordinates before they update the gradient map. The difficulty in parallel coordinate descent methods lies in dealing with correlated pixel values¿ In our deconvolution problem, two pixels which are close to each other in the image are correlated. The more they overlap, the more we over-estimate their pixel values with parallel coordinate descent methods[7]. This over-estimation leads to slow convergence. Or if we increase the number of parallel updates, can lead to a divergence. This means we cannot simply modify our serial coordinate descent algorithm to take a number of parallel steps. We need a way to deal with the over-estimation.

Our serial coordinate descent algorithm uses a greedy pixel selection strategy. Parallel coordinate descent methods on the other hand often select their pixels uniformly at random. The random selection strategy lets us estimate how much the parallel algorithm over-estimates the pixel values[41]. However, a random strategy is not efficient for our deconvolution problem.

We first explain the parallel coordinate descent method in the next section 7.2. We then introduce the modifications we developed for an efficient parallel deconvolution algorithm.

7.2 Parallel (Block) Coordinate Descent Method (PCDM)

The PCDM algorithm can be seen as a generalization of our serial coordinate descent algorithm from Section 4. The serial algorithm optimizes a single pixel at a time. PCDM can update one, or a whole block of pixels at each iteration. And as the name implies, it can update multiple blocks of pixels in parallel. In this project, we use the accelerated variant of the PCDM algorithm, named APPROX [42]. We first introduce a PCDM deconvolution algorithm and then describe the accelerated variant.

7.2.1 Serial block coordinate descent deconvolution

Here we introduce the serial block coordinate descent algorithm. Instead of optimizing a single pixel in each iteration, the serial block coordinate descent algorithm can update a block of pixels. The block size is left for the user to define. It can be any number between a single pixel, in which case the algorithm is identical to the serial coordinate descent algorithm from Section 4, or the whole image.

Remember the single pixel update from the serial coordinate descent algorithm:

$$pixel_{opt} = \frac{max(gradient_{location} - \lambda\alpha, 0)}{Lipschitz_{location} + (1-\alpha)\lambda} \tag{7.1}$$

[7] Imagine we optimize two pixels next to each other in serial coordinate descent: The serial coordinate descent algorithm optimizes he first pixel, updates the gradient, and then optimizes the second pixel. The value of the second pixel will be magnitudes lower than the first, because most of the emission in that area was already explained by the first pixel. If however we update in parallel, both pixels will end up with a similar value, and both pixels try to explain the same emission.

We optimize the pixel at the current location by taking the gradient and dividing it by the Lipschitz constant. For the serial block coordinate descent algorithm we vectorize the update rule. That means $gradient_{location}$ and $Lipschitz_{location}$ and the output $pixel_{opt}$ become vectors:

$$pixels_{opt} = \frac{max(gradients_{locations} - \lambda\alpha, 0)}{Sum(Lipschitz_{locations}) + (1-\alpha)\lambda} \tag{7.2}$$

This is the serial block coordinate descent update rule. Note that we divide the gradient for each pixel by the the block Lipschitz constant (which is the sum of every pixel Lipschitz constant in the block). The single pixel update rule can take a larger step, but only for a single pixel.

The reader might be familiar with the (F)ISTA method[43]. The block update shown in equation (7.2) is related to the (F)ISTA update step. When the block size equal to the image size (we update all pixels in the image in each iteration), then the serial block coordinate descent is equivalent to (F)ISTA.

7.2.2 Parallel block coordinate descent deconvolution

We present the parallel block coordinate descent algorithm. It is based on the PCD method[41]. In this section, we show how it can be used to solve the deconvolution problem. Our parallel coordinate descent algorithm updates t random blocks of pixels in parallel in each iteration. Each iteration is split into three steps: Step 1 is to select t unique blocks of pixels uniformly at random (we cannot select the same block multiple times). In step 2 we update in parallel each selected block in the reconstructed image x. And finally in step 3 we update the gradient map.

```
dirty = IFFT(GridVisibilities(visibilities))
residualsPadded = ZeroPadding(dirty)

psfPadded = ZeroPadding(PSF)
psfPadded = FlipUD(FlipLR(psfPadded))
gradientUpdate = iFFT(FFT(ZeroPadding(PSF)) * FFT(psfPadded))

x = new Array[,]
gradientsMap = iFFT(FFT(residualsPadded) * FFT(psfPadded))
lipschitzMap = CalcLipschitz(PSF)

objectiveValue = 0.5* Sum(residuals * residuals) + ElasticNet(x)
ESO = CalcESO(CountNonZero(PSF), t, x.Length / blockSize)

do
  oldObjectiveValue = objectiveValue

  //Step 1: select t blocks uniformly at random
  blocks = sample(t)

  //Step 2: update reconstruction
  diffBlocks = new Array
  parallel for each block in blocks
    blockLipschitz = Sum(GetBlock(LipschitzMap, block))

    //increase blockLipschitz according to the ESO
    blockLipschitz = blockLipschitz * ESO
    gradientsBlock = GetBlock(gradientMap, block)
    oldBlock = GetBLock(x, block)
    tmp = gradientsBlock + oldBlock * blockLipschitz
    optimalBlock = Max(tmp - lambda*alpha) / (blockLipschitz + (1 - alpha)*lambda)
```

```
32      diffBlock = optimalBlock - oldBlock
33
34      x[block] += diffBlock
35      diffBlocks[block] = diffBlock
36
37    //Step 3: Update gradients
38    for each block in blocks
39      diffBlock = diffBlocks[block]
40      for each pixel in block
41        diff = diffBlock[pixel]
42        shiftedUpdate = Shift(gradientUpdate, pixelLocation)
43        gradientMap = gradientMap - shiftedUpdate * maxDiff
44
45  while maxAbsDiff  < epsilon
```

The parameter t can be thought of as the number of processors. We select a block to optimize in parallel for each available processor. The parallel algorithm presented here is a synchronous implementation. Each processor waits for the others to finish in each step. The implementation used later in this section is asynchronous, where each processor separately selects a block, updates the reconstruction and updates the gradient map independent of the other processors.

Asynchronous implementation with compare exchange. gradient map gets updated asynchronously.

The core of the parallel coordinate descent algorithm is the Estimated Seperability Overapproximation (ESO). In essence, the ESO represents how much we over-estimate the pixels values when we update t random blocks in parallel. To guarantee convergence, we decrease the step size by the factor of the ESO. With more processors t involved, we have to take smaller and smaller steps to guarantee convergence. Here is a trade-off between the degree of parallelism and the overall convergence speed.

The ESO is derived from three components: The uniform sampling strategy used, the number of parallel updates, and the number of non-zero components in the PSF. We use a t-nice uniform sampling. The ESO that arises from the t-nice sampling (take t blocks uniformly at random) according to[41]:

$$CalcESO(\omega, t, n) = 1 + \frac{(\omega - 1)(t - 1)}{max(1, n - 1)} \tag{7.3}$$

Where ω is the number of non-zero entries in the PSF, t is the number of parallel updates, and n is the number of blocks in the problem. For example: The image is 256^2 pixels in size, we update blocks with a size of 4^2 pixels, the PSF has $\omega = 24$ non zero entries, and we use $t = 4$ processors, then the ESO is:

$$CalcESO(\omega = 24, t = 4, n = (256^2/4^2)) = 1 + \frac{(24 - 1)(4 - 1)}{max(1, 4096 - 1)} \approx 1.017 \tag{7.4}$$

The lowest possible value for the ESO is 1. The fewer processors t we use and the fewer non-zero components ω the PSF has, the closer the ESO is to 1. We want a small ESO with the highest number of processors possible. As we see from equation (7.3), the ESO gets smaller with fewer non-zero values in the PSF.

Remember that the PSF in radio astronomy is typically dense. Although most of its values are close to zero, it generally does not have any zero values. The PSF approximation methods we developed in Section **??** effectively reduce the number of non-zero values. For the parallel coordinate descent algorithm, this leads to an ESO closer to 1, even and we can take larger steps without diverging.

7.3 Accelerated parallel block coordinate descent method

We introduced the parallel coordinate descent algorithm in the previous section. In this section we extend the previous algorithm with gradient acceleration, similar to the APPROX method [42].

Instead of using a single gradient map and a single reconstructed image x variable, we an 'explore' and 'correction' variable of both. The algorithm uses an $xExplore$, $xCorrection$ and $gradientMapExplore$, $gradientMapCorrection$. Intuitively, the 'correction' variables contain the accelerated part of the algorithm, and the 'explore' variables the standard part. The variable θ is our acceleration variable.

```
dirty = IFFT(GridVisibilities(visibilities))
residualsPadded = ZeroPadding(dirty)

psfPadded = ZeroPadding(PSF)
psfPadded = FlipUD(FlipLR(psfPadded))
gradientUpdate = iFFT(FFT(ZeroPadding(PSF)) * FFT(psfPadded))

xExplore = new Array[,]
xCorrection = new Array[,]
gradientsMapExplore = iFFT(FFT(residualsPadded) * FFT(psfPadded))
gradientMapCorrection = new Array[,]
lipschitzMap = CalcLipschitz(PSF)

objectiveValue = 0.5* Sum(residuals * residuals) + ElasticNet(x)
ESO = CalcESO(CountNonZero(PSF), t, x.Length / blockSize)
theta =
theta0 =

do
  oldObjectiveValue = objectiveValue

  //Step 1: select t blocks uniformly at random
  blocks = sample(t)

  //Step 2: update reconstruction
  diffBlocks = new Array
  parallel for each block in blocks
  blockLipschitz = Sum(GetBlock(LipschitzMap, block))

  //increase blockLipschitz according to the ESO
  blockLipschitz = blockLipschitz * ESO
  gradientsBlock = GetBlock(gradientMap, block)
  oldBlock = GetBLock(x, block)
  tmp = gradientsBlock + oldBlock * blockLipschitz
  optimalBlock = Max(tmp - lambda*alpha) / (blockLipschitz + (1 - alpha)*lambda)
  diffBlock = optimalBlock - oldBlock

  x[block] += diffBlock
  diffBlocks[block] = diffBlock

  //Step 3: Update gradients
  for each block in blocks
  diffBlock = diffBlocks[block]
  for each pixel in block
  diff = diffBlock[pixel]
  shiftedUpdate = Shift(gradientUpdate, pixelLocation)
  gradientMap = gradientMap - shiftedUpdate * maxDiff
```

```
48
49 while maxAbsDiff  < epsilon
```

This algorithm is identical to the parallel coordinate descent algorithm when we do not modify θ.

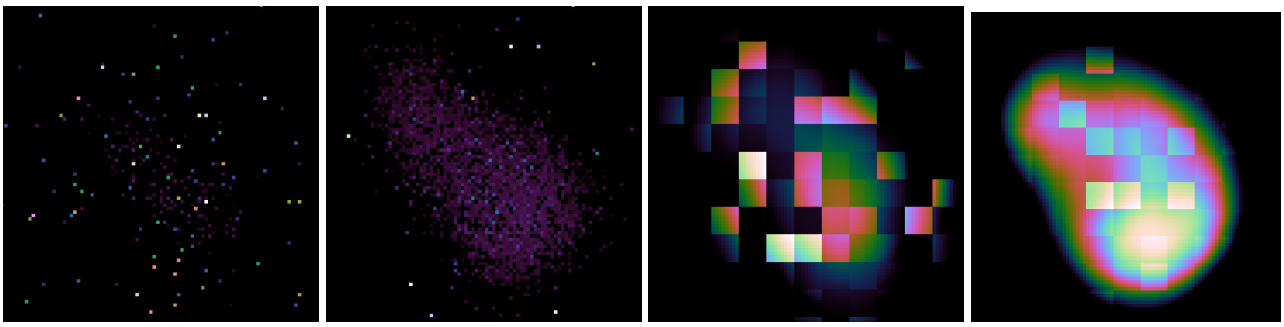
Indeed, the first iteration is identical.

We should converge faster, but we double the amount of memory we need. Also, for an asynchronous implementation the problem is the gradient maps: Asynchronous updates to the gradient map need some sort of communication between the different processes. By doubling the gradient map, we also double the need for communication.

It is not clear if the accelerated parallel coordinate descent algorithm is worth the costs. We later find out that at least for the LMC observation it is not.

7.4 The problem with random selection for deconvolution

Parallel coordinate descent methods depend on a random pixel selection strategy. As we mentioned before, a random selection strategy seems to perform badly on the deconvolution problem. We show the behavior on the N132D supernova remnant of the LMC observation shown in Figure 25.



(a) 8k iterations *(b) 80k iterations* *(c) 8k iterations, 8^2 block* *(d) 80k iterations, 8^2 block*

Figure 25: Random parallel deconvolutions on the LMC N132D supernova remnant.

Large area of the image has only zero-valued pixels. When we randomly select pixels, we are more likely to select a useless pixel. We can waste a lot of resources randomly selecting irrelevant pixels.

A random selection strategy yields an image with obvious artifacts. The reason behind this behavior lies in the first couple of random deconvolution iterations. The parallel algorithm tries to explain as much of the emission as possible with random pixels. The randomly selected pixels are too high. Any random update in the area cannot improve the objective before it has hit one of the first random pixels. We see this behavior in Figure 25a and 25b. After 8 thousand iterations we still have mostly a random pattern of pixels. After 80 thousand iterations, we can see the hints of the supernova remnant. But the randomly selected pixels still explain away too much of the emission.

The also persists when we use blocks of pixels. Figure 25c and 25d show the result of using blocks 8^2 pixels. In each iteration we select blocks of pixels. After 80k iterations the N132D supernova remnant is visible, but it still contains artifacts.

We tackled this problem with two strategies, using an active set heuristic and a pseudo-random selection strategy. The active set heuristic improves the efficiency of random block selection, while the pseudo-random selection strategy deals with the artifacts from Figure 25.

7.4.1 Active set heuristic

Active set heuristic used for cyclic coordinate descent. Choose a subset of blocks and optimize those until convergence. Then choose a new set.

How to choose the active set: We choose all blocks that can be modified currently.

What is an active set iteration. We do a set number of iterations on the active set in parallel and then check whether it makes sense to change the active set. Set of heuristic for finding out if the current active set is still valid:

Idea for checking if active set is still valid: If the pixel which would get updated by the serial coordinate descent algorithm is contained in the active set, we are likely good to go.

If the absolute maximum value of the blocks in the active set decreases faster than the abs-max pixel, we also need to restart.

7.4.2 Pseudo-random selection

Random selection leads to artifacts. Because we need to hit the same block that was randomly selected in the first few iterations. Greedy strategy possible, but the ESO is defined for selecting pixels τ-nice sampling. A greedy strategy may break the ESO from the τ-nice sampling.

But greedy strategy is also on a spectrum. We used

7.4.3 Restarting

Restarting FISTA.

Can improve convergence speed a lot.

Show code

In our case, we need to change the active set more often. Basically need to restart the acceleration parameter θ each time we change the active set.

7.5 ”Minor” cycles

PSF approximations work really well. Then we need more major cycles, which kill the benefit from the PSF approximation

How far we can go with the approximation and the current dirty image. Re-introduce something similar to the minor cycle of CLEAN. We start the minor cycle by deconvolving the dirty image with the approximate PSF. The residuals are now only an approximation, since we used the smaller PSF. We now reset the residuals, we convovle the full PSF with the reconstructed image and subtract the result from the dirty image.

We can skip some major cycles.

7.6 Parameters of the accelerated parallel coordinate descent algorithm

What was tested. Just the pure active set iterations.

We measure the time and the objective value. What parameters lead to the fastest convergence times.

7.6.1 Block size

Block size is not useful to speed up. More difficult code.

From here on out we will always use single pixels. I.e. blocks of the size of 1.

7.6.2 PSF approximation

We expect to benefit a lot more

And it does.

7.6.3 Pseudo-random strategy

By how much do we need to search the area.

And it is a tiny fraction. A lot less communication cost. Close to random, but not quite.

7.6.4 Acceleration

Not useful

7.7 Comparison to the serial coordinate descent algorithm

Which is faster

Proper comparison

Acceleration factor

7.8 Scalability of the parallel coordinate descent algorithm

How many processors can we realistically use. At what point does the ESO get too small.

8 Discussion

Problem of calibration.

CLEAN: MAsking as part for low noise. Maybe also works for Coordinate Descent

Figure 21 shows that at major cycle index 4 (the fifth major cycle), all approximations except for $frac164$ are within 0.09% of the original solution. The current implementation stops when coordinate descent converges within 1000 iterations in the current major cycle. This rule may be too strict, and our coordinate descent algorithm may be stopped earlier.

GPU Coordinate Descent

Difficult to achieve speedup with a dense PSF we approximated

8.1 Approximation of the PSF

Approximation that works. To our knowledge we are the first to explore an approximated PSF. Not relevant for all optimization algorithm, but parallel coordinate descent methods achieve a (significant) speedup.

In our test we were able to use it without needing more major cycles than multi-scale CLEAN.

The need for major cycles could be further reduced. Question of effective heuristics that work for a wide range of

Parallel deconvolutions

8.2 Calibration errors

Common experience that CLEAN based algorithms handle calibration errors better [29] than ”proper” optimization algorithms.

It is a question why. We experienced similar results, but used a simple regularization: elastic net. Other algorithms like MORESANE[4] use more sophisticated regularization.

Why does CLEAN work so well with calibration errors? Is there some sort of implicit regulariztion in the way it choses its pixels. (not in the actual regularization)

8.3 CLEAN heuristics for coordinate descent

Since CLEAN and coordinate descent methods are acting out similar things, we may be able to use CLEAN heuristics to improve coordinate descent based methods.

Masking: CLEAN uses masking to reconstruct sources below the noise level of the image. After a number of iterations, it masks all pixels which are not zero. (Ie. now all sources are detected) and clean deeper.

Similar stuff can be done for coordinate descent methods.

Greedy Coordinate descent

8.4 Approx scales

Wide field of view observations are what Approx likes.

8.5 Hydra

our coordinate descent method is not distributed yet. Hydra exists. It is APPROX in the distributed environment. We had to adapt the implementation of APPROX for the deconvolution problem. Similar adaptions necessary for a distributed algorithm. Problems of cold start, and irrelevant pixels.

8.6 Multi frequency extension

Difficult.

Regularized inverse problem [44]. Objective function How it works, adding a new term to the objective function

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - X * PSF\right\|_2^2 + \lambda ElasticNet(X) + \lambda_v \left\|DX\right\|_1 \tag{8.1}$$

Where D is the Discrete cosine transform.

Does not have a proximal operator for each pixel. problem for Coordinate descent method.

Question if each iteration can be cheap.

But may be separated with respect to frequency with Lagrangian multipliers. Question if cd methods are faster.

9 Conclusion

Works

References

[1] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. IEEE Transactions on information theory, 52(2):489–509, 2006.

[2] David L Donoho. Compressed sensing. IEEE Transactions on information theory, 52(4):1289–1306, 2006.

[3] Arwa Dabbech, Alexandru Onose, Abdullah Abdulaziz, Richard A Perley, Oleg M Smirnov, and Yves Wiaux. Cygnus a super-resolved via convex optimization from vla data. Monthly Notices of the Royal Astronomical Society, 476(3):2853–2866, 2018.

[4] Arwa Dabbech, Chiara Ferrari, David Mary, Eric Slezak, Oleg Smirnov, and Jonathan S Kenyon. Moresane: Model reconstruction by synthesis-analysis estimators-a sparse deconvolution algorithm for radio interferometric imaging. Astronomy & Astrophysics, 576:A7, 2015.

[5] JA Högbom. Aperture synthesis with a non-regular distribution of interferometer baselines. Astronomy and Astrophysics Supplement Series, 15:417, 1974.

[6] Urvashi Rau and Tim J Cornwell. A multi-scale multi-frequency deconvolution algorithm for synthesis imaging in radio interferometry. Astronomy & Astrophysics, 532:A71, 2011.

[7] Charles A Bouman and Ken Sauer. A unified approach to statistical tomography using coordinate descent optimization. IEEE Transactions on image processing, 5(3):480–492, 1996.

[8] Simon Felix, Roman Bolzern, and Marina Battaglia. A compressed sensing-based image reconstruction algorithm for solar flare x-ray observations. The Astrophysical Journal, 849(1):10, 2017.

[9] Madison Gray McGaffin and Jeffrey A Fessler. Edge-preserving image denoising via group coordinate descent on the gpu. IEEE Transactions on Image Processing, 24(4):1273–1281, 2015.

[10] Olivier Fercoq, Zheng Qu, Peter Richtárik, and Martin Takáč. Fast distributed coordinate descent for non-strongly convex losses. In 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), pages 1–6. IEEE, 2014.

[11] Jean-Luc Starck, David L Donoho, and Emmanuel J Candès. Astronomical image representation by the curvelet transform. Astronomy & Astrophysics, 398(2):785–800, 2003.

[12] Jean-Luc Starck, Fionn Murtagh, and Mario Bertero. Starlet transform in astronomical data processing. Handbook of Mathematical Methods in Imaging, pages 2053–2098, 2015.

[13] Andreas M Tillmann and Marc E Pfetsch. The computational complexity of the restricted isometry property, the nullspace property, and related concepts in compressed sensing. IEEE Transactions on Information Theory, 60(2):1248–1259, 2013.

[14] Abhiram Natarajan and Yi Wu. Computational complexity of certifying restricted isometry property. arXiv preprint arXiv:1406.5791, 2014.

[15] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. In Geometric Aspects of Functional Analysis, pages 163–179. Springer, 2017.

[16] Emmanuel J Candes and Yaniv Plan. A probabilistic and ripless theory of compressed sensing. IEEE transactions on information theory, 57(11):7235–7254, 2011.

[17] Keith Miller. Least squares methods for ill-posed problems with a prescribed bound. SIAM Journal on Mathematical Analysis, 1(1):52–74, 1970.

[18] Yves Wiaux, Laurent Jacques, Gilles Puy, Anna MM Scaife, and Pierre Vandergheynst. Compressed sensing imaging techniques for radio interferometry. Monthly Notices of the Royal Astronomical Society, 395(3):1733–1742, 2009.

[19] André Ferrari, David Mary, Rémi Flamary, and Cédric Richard. Distributed image reconstruction for very large arrays in radio astronomy. In 2014 IEEE 8th Sensor Array and Multichannel Signal Processing Workshop (SAM), pages 389–392. IEEE, 2014.

[20] Julien N Girard, Hugh Garsden, Jean Luc Starck, Stéphane Corbel, Arnaud Woiselle, Cyril Tasse, John P McKean, and Jérôme Bobin. Sparse representations and convex optimization as tools for lofar radio interferometric imaging. Journal of Instrumentation, 10(08):C08013, 2015.

[21] Rafael E Carrillo, Jason D McEwen, and Yves Wiaux. Purify: a new approach to radio-interferometric imaging. Monthly Notices of the Royal Astronomical Society, 439(4):3591–3604, 2014.

[22] Rafael E Carrillo, Jason D McEwen, and Yves Wiaux. Sparsity averaging reweighted analysis (sara): a novel algorithm for radio-interferometric imaging. Monthly Notices of the Royal Astronomical Society, 426(2):1223–1234, 2012.

[23] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-i. a full-sky jones formalism. Astronomy & Astrophysics, 527:A106, 2011.

[24] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-ii. calibration and direction-dependent effects. Astronomy & Astrophysics, 527:A107, 2011.

[25] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-iii. addressing direction-dependent effects in 21 cm wsrt observations of 3c 147. Astronomy & Astrophysics, 527:A108, 2011.

[26] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-iv. a generalized tensor formalism. Astronomy & Astrophysics, 531:A159, 2011.

[27] BG Clark. An efficient implementation of the algorithm'clean'. Astronomy and Astrophysics, 89:377, 1980.

[28] FR Schwab. Relaxing the isoplanatism assumption in self-calibration; applications to low-frequency radio interferometry. The Astronomical Journal, 89:1076–1081, 1984.

[29] AR Offringa and O Smirnov. An optimized algorithm for multiscale wideband deconvolution of radio astronomical images. Monthly Notices of the Royal Astronomical Society, 471(1):301–316, 2017.

[30] Jason D McEwen and Yves Wiaux. Compressed sensing for wide-field radio interferometric imaging. Monthly Notices of the Royal Astronomical Society, 413(2):1318–1332, 2011.

[31] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM Journal on Optimization, 22(2):341–362, 2012.

[32] Yu Nesterov. Gradient methods for minimizing composite functions. Mathematical Programming, 140(1):125–161, 2013.

[33] Jonathan S Kenyon. Pymoresane: Python model reconstruction by synthesis-analysis estimators. Astrophysics Source Code Library, 2019.

[34] Marcel Koester. Ilgpu: A modern, lightweight and fast gpu compiler for high-performance .net programs, 2019.

[35] Justin Luitjens. Faster parallel reductions on kepler, 2014.

[36] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1):1, 2010.

[37] DC-J Bock, MI Large, and Elaine M Sadler. Sumss: A wide-field radio imaging survey of the southern sky. i. science goals, survey design, and instrumentation. The Astronomical Journal, 117(3):1578, 1999.

[38] AR Offringa, Benjamin McKinley, Natasha Hurley-Walker, FH Briggs, RB Wayth, DL Kaplan, ME Bell, Lu Feng, AR Neben, JD Hughes, et al. Wsclean: an implementation of a fast, generic wide-field imager for radio astronomy. Monthly Notices of the Royal Astronomical Society, 444(1):606–619, 2014.

[39] Dan Briggs. High fidelity deconvolution of moderately resolved sources, 2019.

[40] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:1105.5379, 2011.

[41] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. Mathematical Programming, 156(1-2):433–484, 2016.

[42] Olivier Fercoq and Peter Richtárik. Accelerated, parallel, and proximal coordinate descent. SIAM Journal on Optimization, 25(4):1997–2023, 2015.

[43] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2(1):183–202, 2009.

[44] André Ferrari, Jérémy Deguignet, Chiara Ferrari, David Mary, Antony Schutz, and Oleg Smirnov. Multi-frequency image reconstruction for radio interferometry. a regularized inverse problem approach. arXiv preprint arXiv:1504.06847, 2015.

[45] Luke Pratley, Melanie Johnston-Hollitt, and Jason D McEwen. A fast and exact w-stacking and w-projection hybrid algorithm for wide-field interferometric imaging. arXiv preprint arXiv:1807.09239, 2018.

[46] Bram Veenboer, Matthias Petschow, and John W Romein. Image-domain gridding on graphics processors. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 545–554. IEEE, 2017.

List of Figures

List of Tables

10 attachment

11 Larger runtime costs for Compressed Sensing Reconstructions

The MeerKAT instrument produces a new magnitude of data volume. An image with several million pixels gets reconstructed from billions of Visibility measurements. Although MeerKAT measures a large set of Visibilities, the measurements are still incomplete. We do not have all the information available to reconstruct an image. Essentially, this introduces ”fake” structures in the image, which a reconstruction algorithm has to remove. Additionally, the measurements are noisy.

We require an image reconstruction algorithm which removes the ”fake” structures from the image, and removes the noise from the measurements. The large data volume of MeerKAT requires the algorithm to be both scalable and distributable. Over the years, several reconstruction algorithms were developed, which can be separated into two classes: Algorithms based on CLEAN, which are cheaper to compute and algorithms based on Compressed Sensing, which create higher quality reconstructions.

CLEAN based algorithms represent the reconstruction problem as a deconvolution. First, they calculate the ”dirty” image, which is corrupted by noise and fake image structures. The incomplete measurements essentially convolve the image with a Point Spread Function (PSF). CLEAN estimates the PSF and searches for a deconvolved version of the dirty image. In each CLEAN iteration, it searches for the highest pixel in the dirty image, subtracts a fraction PSF at the location. It adds the fraction to the same pixel location of a the ”cleaned” image. After several iterations, the cleaned image contains the deconvolved version of the dirty image. CLEAN accounts for noise by stopping early. It stops when the highest pixel value is smaller than a certain threshold. This results in a light-weight and robust reconstruction algorithm. CLEAN is comparatively cheap to compute, but does not produce the best reconstructions and is difficult to distribute on a large scale.

Compressed Sensing based algorithms represent the reconstruction as an optimization problem. They search for the optimal image which is as close to the Visibility measurements as possible, but also has the smallest regularization penalty. The regularization encodes our prior knowledge about the image. Image structures which were likely measured by the instrument result in a low regularization penalty. Image structures which were likely introduced by noise or the measurement instrument itself result in high penalty. Compressed Sensing based algorithms explicitly handle noise and create higher quality reconstructions than CLEAN. State-of-the-art Compressed Sensing algorithms show potential for distributed computing. However, they currently do not scale on MeerKATs data volume. They require too many computing resources compared to CLEAN based algorithms.

This project searches for a way to reduce the runtime costs of Compressed Sensing based algorithms. One reason for the higher costs is due to the non-uniform FFT Cycle. State-of-the-art CLEAN and Compressed Sensing based algorithms both use the non-uniform FFT approximation in a cycle during reconstruction. The interferometer measures the Visibilities in a continuous space in a non-uniform pattern. The image is divided in a regularly spaced, discrete pixels. The non-uniform FFT creates an approximate, uniformly sampled image from the non-uniform measurements. Both, CLEAN and Compressed Sensing based algorithms use the non-uniform FFT to cycle between non-uniform Visibilities and uniform image. However, a Compressed Sensing algorithm requires more non-uniform FFT cycles for reconstruction.

CLEAN and Compressed Sensing based algorithms use the non-uniform FFT in a similar manner. However, there are slight differences in the architecture. This project hypothesises that The previous project searched for an alternative to the non-uniform FFT cycle. Although there are alternatives, there is currently no replacement which leads to lower runtime costs for Compressed Sensing. Current research is focused on reducing the number of non-uniform FFT cycles for Compressed Sensing algorithms.

CLEAN based algorithms use the Major Cycle Architecture for reconstruction. Compressed Sensing based algorithms use a similar architecture, but with slight modifications. Our hypothesis is that we may reduce the number of non-uniform FFT cycles for Compressed Sensing by using CLEAN’s Major Cycle Architecture.

11.1 CLEAN: The Major Cycle Architecture

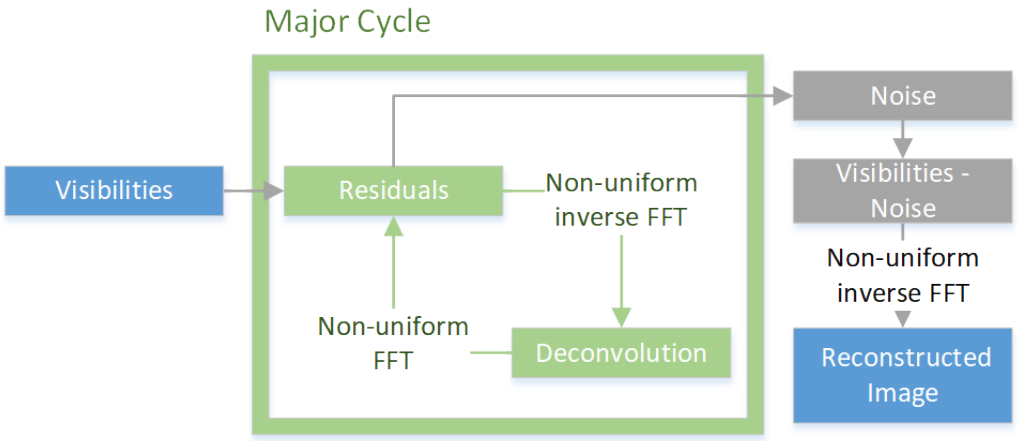


Figure 26: The Major Cycle Architecture

Figure 26 depicts the Major Cycle Architecture used by CLEAN algorithms. First, the Visibilities get transformed into an image with the non-uniform FFT. The resulting dirty image contains the corruptions of the measurement instrument and noise. A deconvolution algorithm, typically CLEAN, removes the corruption of the instrument with a deconvolution. When the deconvolution stops, it should have removed most of the observed structures from the dirty image. The rest, mostly noisy part of the dirty image gets transformed back into residual Visibilities and the cycle starts over.

In the Major Cycle Architecture, we need several deconvolution attempts before it has distinguished the noise from the measurements. Both the non-uniform FFT and the deconvolution are approximations. By using the non-uniform FFT in a cycle, it can reconstruct an image at a higher quality. For MeerKAT reconstruction with CLEAN, we need approximately 4-6 non-uniform FFT cycles for a reconstruction.

11.2 Compressed Sensing Architecture

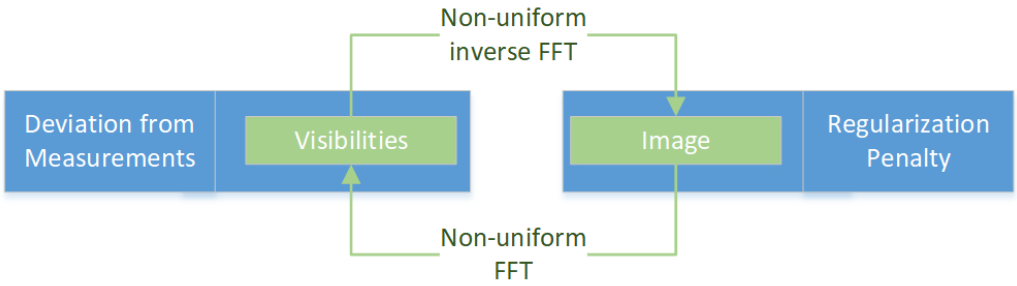


Figure 27: State-of-the-art Compressed Sensing Reconstruction Architecture

Figure 27 depicts the architecture used by Compressed Sensing reconstructions. The Visibilities get transformed into an image with the non-uniform FFT approximation. The algorithm then modifies the image so it reduces the regularization penalty. The modified image gets transformed back to Visibilities and the algorithm then minimizes the difference between measured and reconstructed Visibilities. This is repeated until the algorithm converges to an optimum.

In this architecture, state-of-the-art Compressed Sensing algorithms need approximately 10 or more non-uniform FFT cycles to converge. It is one source for the higher runtime costs. For MeerKAT reconstructions

the non-uniform FFT tends to dominate the runtime costs. A CLEAN reconstruction with the Major Cycle Architecture already spends a large part of its time in the non-uniform FFT. Compressed Sensing algorithms need even more non-uniform FFT cycle on top of the ”Image Regularization” step being generally more expensive than CLEAN deconvolution. There is one upside in this architecture: State-of-the-art algorithms managed to distribute the ”Image Regularization” operation.

11.3 Hypothesis for reducing costs of Compressed Sensing Algorithms

Compressed Sensing Algorithms are not bound to the Architecture presented in section 11.2. For example, we can design a Compressed Sensing based deconvolution algorithm and use the Major Cycle Architecture instead.

Our hypothesis is: We can create a Compressed Sensing based deconvolution algorithm which is both distributable and creates higher quality reconstructions than CLEAN. Because it also uses the Major Cycle architecture, we reckon that the Compressed Sensing deconvolution requires a comparable number of non-uniform FFT cycles to CLEAN. This would result in a Compressed Sensing based reconstruction algorithm with similar runtime costs to CLEAN, but higher reconstruction quality and higher potential for distributed computing.

11.4 State of the art: WSCLEAN Software Package

11.4.1 W-Stacking Major Cycle

11.4.2 Deconvolution Algorithms

CLEAN MORESANE

11.5 Distributing the Image Reconstruction

11.5.1 Distributing the Non-uniform FFT

11.5.2 Distributing the Deconvolution

12 Handling the Data Volume

The new data volume is a challenge to process for both algorithms and computing infrastructure. Push for parallel and distributed algorithms. For Radio Interferometer imaging, we require specialized algorithms. The two distinct operations, non-uniform FFT and Deconvolution, were difficult algorithms for parallel or distributed computing.

The non-uniform FFT was historically what dominated the runtime []. Performing an efficient non-uniform FFT for Radio Interferometers is an active field of research[38, 45], continually reducing the runtime costs of the operation. Recently, Veeneboer et al[46] developed a non-uniform FFT which can be fully executed on the GPU. It speeds up the most expensive operation.

In Radio Astronomy, CLEAN is the go-to deconvolution algorithm. It is light-weight and compared to the non-uniform FFT, a cheap algorithm. It is also highly iterative, which makes it difficult for effective parallel or distributed implementations. However, compressed sensing based deconvolution algorithms can be developed with distribution in mind.

12.1 Fully distributed imaging algorithm

Current imaging algorithms push towards parallel computing with GPU acceleration. But with Veeneboer et al’s non-uniform FFT and a compressed sensing based deconvolution, we can go a step further and create a distributed imaging algorithm.

13 Image Reconstruction for Radio Interferometers

In Astronomy, instruments with higher angular resolution allows us to measure ever smaller structures in the sky. For Radio frequencies, the angular resolution is bound to the antenna dish diameter, which puts practical and financial limitations on the highest possible angular resolution. Radio Interferometers get around this limitation by using several smaller antennas instead. Together, they act as a single large antenna with higher angular resolution at lower financial costs compared to single dish instruments.

Each antenna pair of an Interferometer measures a single Fourier component of the observed image. We can retrieve the image by calculating the Fourier Transform of the measurements. However, since the Interferometer only measures an incomplete set of Fourier components, the resulting image is ”dirty”, convolved with a Point Spread Function (PSF). Calculating the Fourier Transform is not enough. To reconstruct the from an Interferometer image, an algorithm has to find the observed image with only the dirty image and the PSF as input. It has to perform a deconvolution. The difficulty lies in the fact that there are potentially many valid deconvolutions for a single measurement, and the algorithm has to decide for the most likely one. How similar the truly observed image and the reconstructed images are depends largely on the deconvolution algorithm.

State-of-the-art image reconstructions use the Major Cycle architecture (shown in Figure 28), which contains three operations: Gridding, FFT and Deconvolution.

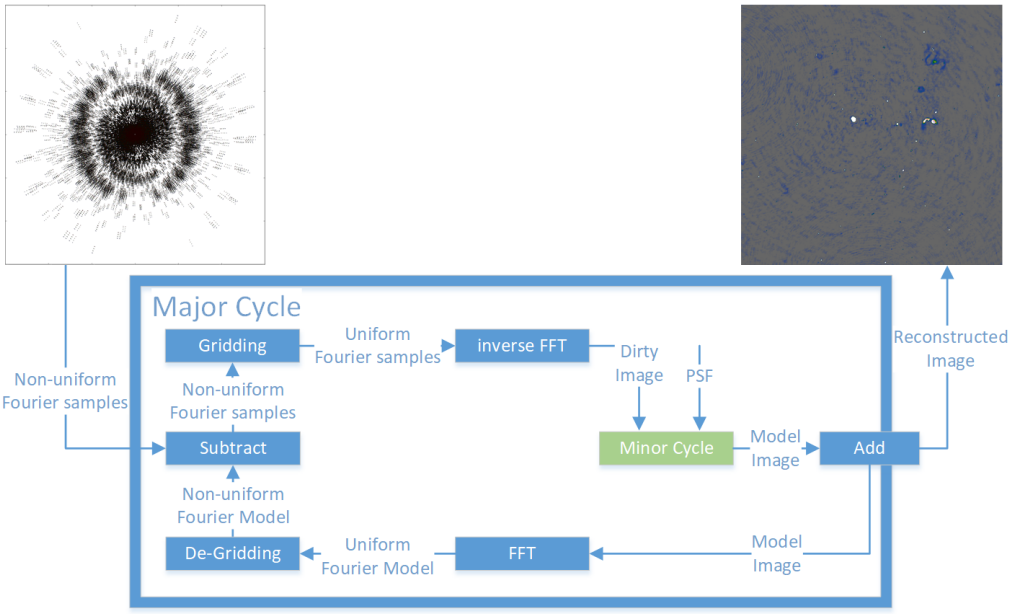


Figure 28: The Major Cycle Architecture of image reconstruction algorithms

The first operation in the Major Cycle, Gridding, takes the non-uniformly sampled Fourier measurements from the Interferometer and interpolates them on a uniformly spaced grid. The uniform grid lets us use FFT to calculate the inverse Fourier Transform and we arrive at the dirty image. A deconvolution algorithm takes the dirty image plus the PSF as input, producing the deconvolved ”model image”, and the residual image as output. At this point, the reverse operations get applied to the residual image. First the FFT and then De-gridding, arriving at the non-uniform Residuals. The next Major Cycle begins with the non-uniform Residuals as input. The cycles are necessary, because the Gridding and Deconvolution operations are only approximations. Over several cycles, we reduce the errors introduced by the approximate Gridding and Deconvolution. The final, reconstructed image is the addition of all the model images of each Major Cycle.

13.1 Distributed Image Reconstruction

New Interferometer produce an ever increasing number of measurements, creating ever larger reconstruction problems. A single image can contain several terabytes of Fourier measurements. Handling reconstruction problems of this size forces us to use distributed computing. However, state-of-the-art Gridding and Deconvolution algorithms only allow for limited distribution. How to scale the Gridding and Deconvolution algorithms to large problem sizes is still an open question.

Recent developments make a distributed Gridder and a distributed Deconvolution algorithm possible. Veeneboer et al[46] found an input partitioning scheme, which allowed them to perform the Gridding on the GPU. The same partitioning scheme can potentially be used to distribute the Gridding onto multiple machines. For Deconvolution, there exist parallel implementations for certain algorithms like MORESANE[4]. These can be used as a basis for a fully distributed image reconstruction.

In this project, we want to make the first steps towards an image reconstruction algorithm, which is distributed from end-to-end, from Gridding up to and including deconvolution. We create our own distributed Gridding and Deconvolution algorithms, and analyse the bottlenecks that arise.

13.2 First steps towards a distributed Algorithm

In this project, we make the first steps towards a distributed Major Cycle architecture (shown in figure 28) implemented C#. We port Veeneboer et al's Gridder, which is written in C++, to C# and modify it for distributed computing. We implement a simple deconvolution algorithm based on the previous project and create a first, non-optimal distributed version of it.

In the next step, we create a more sophisticated deconvolution algorithm based on the shortcomings of the first implementation. We use simulated and real-world observations of the MeerKAT Radio Interferometer and measure its speed up. We identify the bottlenecks of the current implementation and explore further steps.

From the first lessons, we continually modify the distributed algorithm and focus on decreasing the need for communication between the nodes, and increase the overall speed up compared to single-machine implementations. Possible Further steps:

- Distributed FFT
- Replacing the Major Cycle Architecture
- GPU-accelerated Deconvolution algorithm.

A state-of-the-art reconstruction algorithm has to correct large number of measurement effects arising from the Radio Interferometer. Accounting for all effects is out of the scope for this project. We make simplifying assumptions, resulting in a proof-of-concept algorithm.

14 Ehrlichkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende schriftliche Arbeit selbstständig und nur unter Zuhilfenahme der in den Verzeichnissen oder in den Anmerkungen genannten Quellen angefertigt habe. Ich versichere zudem, diese Arbeit nicht bereits anderweitig als Leistungsnachweis verwendet zu haben. Eine Überprüfung der Arbeit auf Plagiate unter Einsatz entsprechender Software darf vorgenommen werden.
Windisch, November 19, 2019

Jonas Schwammberger