P9 Towards distributed image reconstruction for radio interferometers

Jonas Schwammberger

December 30, 2019

Abstract

New approximation method. Gradient approximation simplifies parallel and distributed reconstruction.

Developed a parallel coordinate descent algorithm, which can Way for easy parallel processing and later distribution.

Contents

1 The image reconstruction problem of radio interferometers

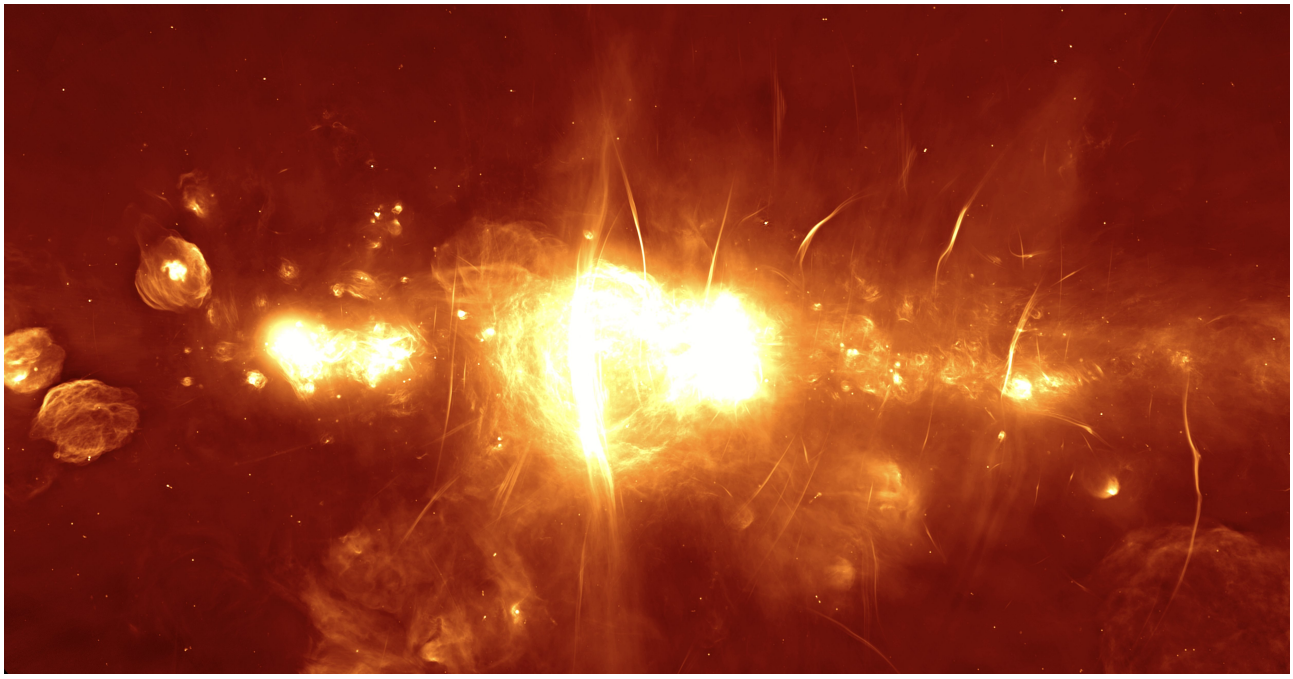


Figure 1: Center of the Milky Way galaxy at radio wavelengths. Observed by the MeerKAT interferometer.

Our observable universe emits a wide band of electromagnetic wavelengths. The spectrum visible to our eyes is only a narrow section of all the electromagnetic radiation in the universe. At different wavelengths, the same celestial objects reveal another picture. The center of the milky way galaxy is surrounded by dust clouds, making the center invisible to optical telescopes. The longer radio waves however pass right through the dust clouds, revealing the center of the milky way galaxy to the MeerKAT radio telescope in Figure 1.

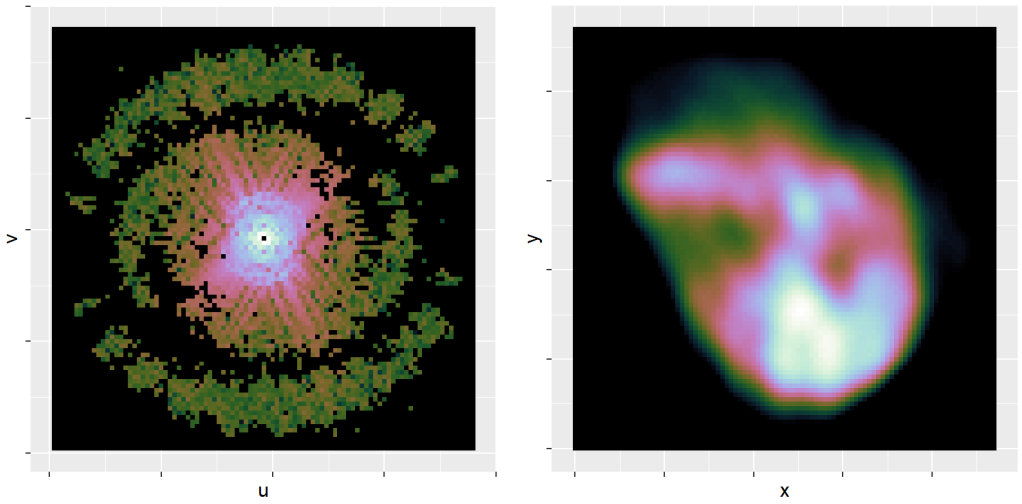
Radio telescopes are build with an ever increasing angular resolution. The higher angular resolution allows the instruments to detect ever smaller objects in the sky. But the angular resolution of single dish antennas depends on the dish-diameter and the observed wavelength. With longer wavelengths, we need bigger dishes to achieve a similar angular resolution. The long radio wavelengths require huge dishes for a similar angular resolution to optical telescopes.

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 construct economically. 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 as a single large dish. An interferometer can achieve angular resolutions which are comparable to a single dish with a diameter of several kilometers.

But there are drawbacks: The interferometer does not measure the sky in pixels, but in the Fourier space. It measures the amplitude and phase of a set of Fourier components. The image has to be calculated from the Fourier measurements. This is the image reconstruction problem of radio interferometers. The measured Fourier components are called visibilities in the radio astronomy literature. From here on forward, we will call the measured Fourier components visibilities.

The Figure 2 shows an example of the image reconstruction problem. The radio interferometer measures a set of visibilities (Fourier components) in the Fourier space shown in Figure 2a. Each point in the figure represents a Fourier component, where the color presents the magnitude of the component. The center points represent low-frequency, while the points away from the center represent the high-frequency components of

the observed image. Low-frequency components contain information about large structures in the image, while the high-frequency contain information about details, like sharp edges in the image. However, the interferometer does not measure every visibility for any reconstruction problem, shown as black dots in Figure 2a. The image reconstruction algorithm is tasked to find the observed image, shown in Figure 2b, from the incomplete set of visibilities.



(a) Measured visibilities (Fourier components) of the N132 supernova-remnant.

(b) Observed image of the N132 supernova-remnant.

Figure 2: Example of an image reconstruction problem of radio interferometers

Furthermore the visibility measurements are corrupted by noise. The atmosphere of the earth can change the true amplitude and phase of each measured visibility. Under unfavorable conditions, the atmosphere can introduce a high level of noise when compared to the signal.

These two difficulties, the noise and the incomplete measurements, lead to the fact that there are many different candidate images that fit any measurements. This is known as an ill-posed inverse problem. We want to find the observed image, even though all we have are an incomplete set of noisy visibility 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 celestial objects. The image consists of stars, hydrogen clouds, and other celestial objects which emit radio waves. 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] and its extensions. It has shown to be robust against noise in the reconstruction problem, and

T

It has shown to be robust against the noise[7] 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.

T

Coordinate descent methods have been successfully applied in other inverse problems, such as reconstruction of CT scans[8], or X-Ray imaging[9]. GPU accelerated[10] and distributed[11] 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

In radio astronomy, there are two classes of radio sources: Point sources and extended emissions. Point sources are typically far-away stars, their emission is concentrated around a single pixel in the observed image. Extended emissions are objects that span a large area of the sky, like dust or hydrogen clouds. They span an area over several pixels. Both classes emit radio waves. The radio waves travel to earth, may get distorted by Earths Ionosphere, and finally arrive at the interferometer. Figure 3 shows the radio interferometry system.

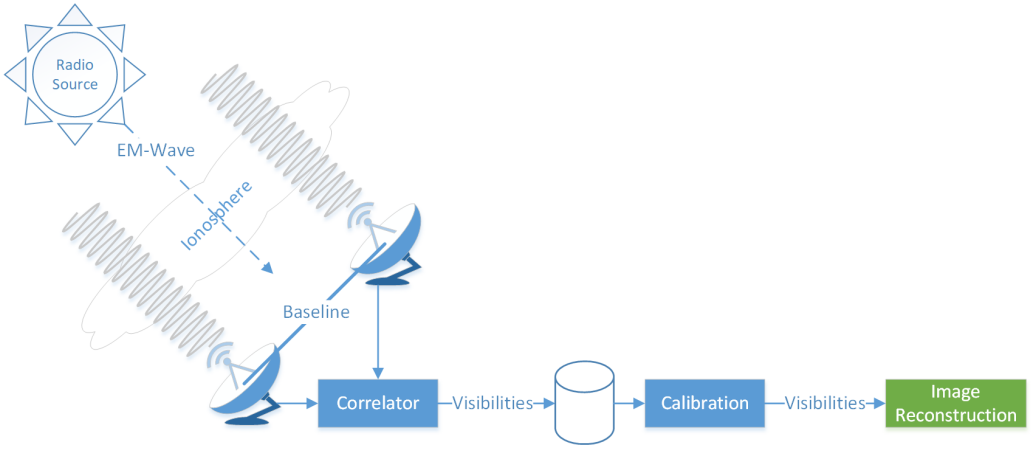


Figure 3: Radio interferometry system

The electromagnetic waves arrive at each antenna of the interferometer. The Correlator then takes the measurement of each antenna pair. Each antenna pair is called a baseline. The correlator creates a complex visibility measurement for each baseline, consisting of amplitude and phase. After this step, the visibilities are saved for later reconstruction.

Before the visibilities can be reconstructed into an image, they need to be calibrated. Each antenna has a different sensitivity which drifts over time. Also, the distortion by Earths Ionosphere can be approximated and removed, depending on the observation.

The calibration process is imperfect. The visibility measurements are still noisy, meaning their true amplitude and phase may differ from what was measured. The image reconstruction step is tasked to find the most likely image from noisy and incomplete visibilities.

This project is focused on the image reconstruction step. We will develop our own image reconstruction algorithm that finds the most-likely image given the calibrated visibility measurements. However, the specifics of radio interferometric measurements influence every part of the image reconstruction algorithm. Efficient reconstruction algorithms have to deal with the specifics of radio interferometry.

This section first introduces how image reconstruction works in theory. Then, we introduce the MeerKAT radio interferometer, for which we have real-world visibility measurements. We show the difficulties the MeerKAT interferometer introduces for the image reconstruction. Finally, we show how real-world image reconstruction algorithms work to deal with MeerKAT data, and show where our contributions lie.

2.1 Introduction to the ill-posed inverse problem

Radio interferometers like MeerKAT generally produce more visibility measurements than pixels in the reconstruction. Still, the measurements are incomplete, and from the measurements alone, we cannot reconstruct the observed image. In the literature this is known as an ill-posed inverse problem. We wish to find the observed image from measurements in the Fourier space, but the measurements alone do not contain all the

relevant information.

However, we can include prior information about the image, we can reconstruct the most-likely image given the measurements. Here we introduce how a reconstruction can find the most-likely image given the measurements. First, we formulate the image reconstruction as a minimization problem with the following objective function:

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

Where x is the reconstructed image we wish to find, V are the measured visibilities, F is the Fourier transform matrix and M is the masking matrix. The masking matrix sets the visibilities that were are missing in the image.

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.

The objective (2.1) only includes information about the measurements. As we have mentioned before, we have prior knowledge about the image. We know for exampe it is likely to contain point sources. In that case, most pixels of the image will be zero, except for the locations where the interferometer has located the point sources. 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:

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

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: Does it point to the observed image? In practice, if the regularization is a good model for the image content, the image we retrieve at the minimum of the objective (2.2) is indistinguishable from the observed image.

The reconstruction quality also depends on how well the regularization models the observed image. In our example the L1 norm is a good model for images only consisting of stars. It penalizes images that have a large number of non-zero pixels. But radio interferometer also measure extended emissions like gas-clouds. Those images lead to a large number of non-zero pixels and by extent a large regularization penalty. The L1 regularization may force non-zero pixels of the extended emission to zero.

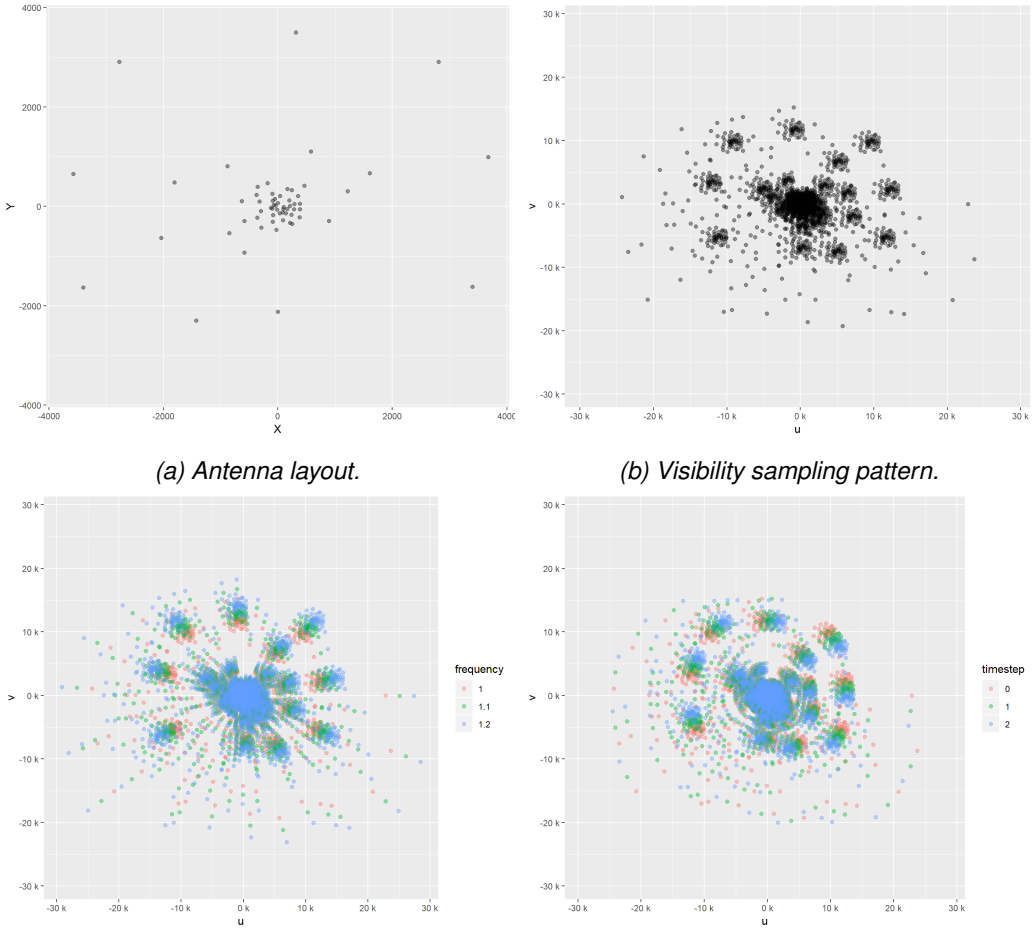
In a sense there is data modeling task involved in the image reconstruction. The better we understand what a plausible image looks like, the higher the reconstruction quality. Recent work managed to reconstruct super-resolved images for the VLA interferometer[3], where the reconstruction achieved a resolution higher than the limit of the instrument itself.

[1] Sum of absolute values of the pixels, $\sum_i \sum_j \|x_{ij}\|$

The image reconstruction problem is ill-posed, but with the right regularization we can reconstruct an image which even surpasses the accuracy limit of the instrument. However, the regularization also influences other properties of the reconstruction algorithm, such as the wall-clock time for a single reconstruction, or its behavior with very noisy visibilities. The question of the optimal regularization is, as of the time of writing, still open in radio astronomy.

2.2 Introduction to the MeerKAT observations

So far, we looked at how image reconstruction works in theory. We need a regularization to model our prior knowledge, that the image is a mixture of point sources and extended emissions. Then, all we need is a convex optimization algorithm, which finds us the most likely image. Real-world radio interferometer introduce complexities which have to be handled by the reconstruction algorithm. First, there is a large number of visibility measurements. Any operation on the measurements is potentially time consuming. Secondly, the measurements of modern radio interferometers become more complicated, as we will show in the next section. Here, we show how MeerKAT can produce an almost arbitrary large amount of visibility measurements.



(a) Antenna layout.

(b) Visibility sampling pattern.

(c) Visibilities added from multiple channels.

(d) Visibilities added from multiple timesteps.

Figure 4: Sampling regime of the MeerKAT radio interferometer.

The MeerKAT radio interferometer consists of 64 antennas. Each antenna pair creates a single baseline, and each baseline results in a single visibility. This results in 2016 visibilities for the MeerKAT interferometer. The distance between the antennas of each pair is important: The distance defines what point in the Fourier space gets sampled. A short baseline samples close to the origin of the Fourier space, and contain low-frequency Fourier components of the sky image. Long baselines measure points further away from the origin. They

sample the high-frequency Fourier components, and contain information about sharp edges in the image. Figure 4a shows the 64 antennas of MeerKAT, while the Figure 4b shows the 2016 visibility samples in the Fourier space.

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. Modern radio interferometers use two tricks to fill up the Fourier space with more visibility samples: Adding visiblities from different channels, and from different timesteps.

The interferometer measures the sky at different radio channels simultaneously. We can add the visibilities from neighboring channels together, resulting in the visibility pattern shown in Figure 4c. Each channel measures the Fourier space using the same pattern, but scaled by the radio frequency.

The interferometer observes the sky over a period of time. Depending on the observation, this can be minutes, up to several hours. During this time, the rotation of the Earth rotates the sampling pattern in Fourier space, shown in Figure 4d.

The MeerKAT radio interferometer measures 2016 visibilities, for each channel at each timestep. It has 20 thousand channels, and the time resolution can be as low as half a second. This results in roughly 80 million visibility measurements for each second of observation. A real-world observation can span over several hours, and take up several terabytes of disk space. The data volume alone creates difficulties in any practical reconstruction algorithm. The large number of visibility measurements is difficult to keep in memory. Any operation on the whole set of visibilities becomes impractical.

But MeerKAT introduces further complications apart from the large data volume. Adding the visibilities from 20 thousand channels together becomes inaccurate. For an accurate reconstruction, the reconstruction algorithm has to perform wide-band imaging: Instead of reconstructing a single image containing the visibilities from all frequencies, the algorithm reconstructs an image cube, where each image represents a single, or a small fraction of all channels. Wide-band imaging is out of scope for our project, and our reconstruction algorithm only considers narrow-band imaging.

Furthermore, the reconstructed image from MeerKAT observations generally span a wide field-of-view. For an accurate reconstruction, the algorithm has to account for the fact that the visibilities are three dimensional. We include wide field-of-view imaging in our reconstruction algorithm, and discuss the effects of the third visibility term in more detail.

2.2.1 Introduction to the third visibility term

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. We have ignored the fact that the visibility measurements not only have a $u-$ and v-, but also a third w-term. Visibilities are in fact three dimensional.

The third w component arises from the fact that the baselines (antenna pairs) are not on the same plane. The w-term becomes relevant for wide field-of-view imaging in radio interferometry. We will show in later sections how the w-term affects the design of the image reconstruction algorithm. Let us start with the measurement equation for small field-of-views:

$$V(u,v) = \int\int I(l,m) e^{2\pi i[ul+vm]}\, dl\, dm \tag{2.3}$$

The radio interferometer measures Visibilities V from the sky image I. Each visibility is a measurement over all pixels l, m. The term $e^{2\pi i[ul+vm]}$ is simply the Fourier transform. l and m are the directions from which we measured radiation. They can be thought of as the pixel indices of the image. Index $(0,0)$ is the center

pixel of the image. So far the visibilities are two dimensional. But this measurement equation is only valid for a small field-of-view. For a large field-of-view the measurement equation expands to:

$$V(u,v,w) = \int\int \frac{I(l,m)}{c(l,m)} e^{2\pi i[ul+vm+w(c(l,m)-1)]}\, dl\, dm\,, \quad c(l,m) = \sqrt{1-l^2-m^2} \tag{2.4}$$

We still have the Fourier transform $e^{2\pi i[ul+vm...]}$ at the core. But now the visibilities have a third w-term, the image has a normalization factor of $c(l,m)$ and the Fourier transform has a phase-shift term $+w(c(l,m)-1)$ added to it. The phase-shift of the Fourier transform depends on the direction from which the radiation was measured. As such, the phase shift changes over the image. It is called a Directionally Dependent Effect (DDE).

There exist several sources of DDE's. The Ionoshpere for example is a natural source that distorts the signal depending on the direction. From different directions, the distortion by the Ionosphere changes. The w-term arises from the properties of the interferometer itself. DDE's are costly in terms of computation time. The w-term breaks the two dimensional Fourier relationship between the image and the visibilities. We cannot use the FFT, resulting in a more expensive Fourier transformation. The difficulty lies in handling DDE's efficiently in the image reconstruction algorithm.

2.3 Introduction to image reconstruction algorithm for radio interferometer

This section introduces the commonly used architecture for radio interferometric image reconstruction, the Major/Minor cycle. It shows how DDE's like the w-term are handled, and introduces the basic reconstruction algorithm for radio astronomy: CLEAN.

First, let us look back at the objective function. In the previous section, the objective function (2.2) contained the visiblity measurements V in Fourier space, the reconstructed image x in image space, and the Fourier transform matrix F that represents the relationship between image and Fourier space. However, this is only one way to formulate the reconstruction problem:

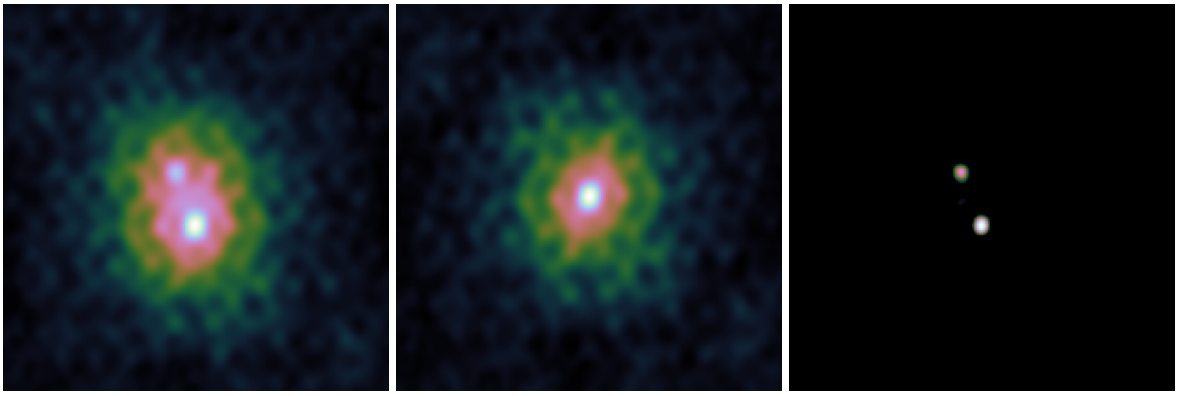
$$\begin{aligned}
&\underset{x}{minimize}\ \|V - MFx\|_2^2 + \lambda \|x\|_1 \\
&\underset{V_2}{minimize}\ \|V - MV_2\|_2^2 + \lambda \left\|F^{-1}V_2\right\|_1 \\
&\underset{x}{minimize}\ \|I_{dirty} - x * PSF\|_2^2 + \lambda \|x\|_1
\end{aligned} \tag{2.5}$$

We can also reconstruct the missing visibilities directly. In that case, we solve an in-painting problem in the Fourier space. Or we can transform the measurements into image space, and solve an equivalent deconvolution problem, where the masking matrix M in Fourier space becomes the convolution kernel called the Point Spread Function PSF (Remember that a multiplication in Fourier space is a convolution in image space.)

These three problems are equivalent in theory. The key difference between the three formulations is that the we generally have magnitudes fewer pixels than visibility measurements in the reconstruction problem. The deconvolution problem therefore consumes less memory.

Image reconstruction algorithms for radio interferometers generally use the deconvolution formulation. They first transform the visibility measurements to an image, called the dirty image. Then, they transform the masking matrix M into the PSF. At this point, the algorithms reconstruct the image by calculating a deconvolution. Figure 5 shows a simulated example of a dirt image, PSF and deconvolved image.

In radio astronomy, image reconstruction is split into two separate algorithms. The first algorithm is responsible for transforming the measurements from Fourier to image space. The second algorithm is responsible for



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

Figure 5: Example deconvolution problem with two point sources.

deconvolution. The first algorithm is referred as the Major cycle in radio astronomy, and the deconvolution algorithm as the minor cycle in radio astronomy literature. The reason why they are referred to as ”cycles” will become clear in Section 2.3.2. In short: The whole process, transforming the measurements to image space and calculating the deconvolution has to be repeated several times to correct for DDE's like the w-term. First, we will take a closer look to the most commonly used deconvolution algorithm, CLEAN.

2.3.1 CLEAN deconvolution algorithm

CLEAN[5] is the basic deconvolution algorithm in radio astronomy. It has been extended over the years, and in its more sophisticated form as multi-scale multi-frequency CLEAN [7] is still used today. We introduce the basic CLEAN algorithm in this section.

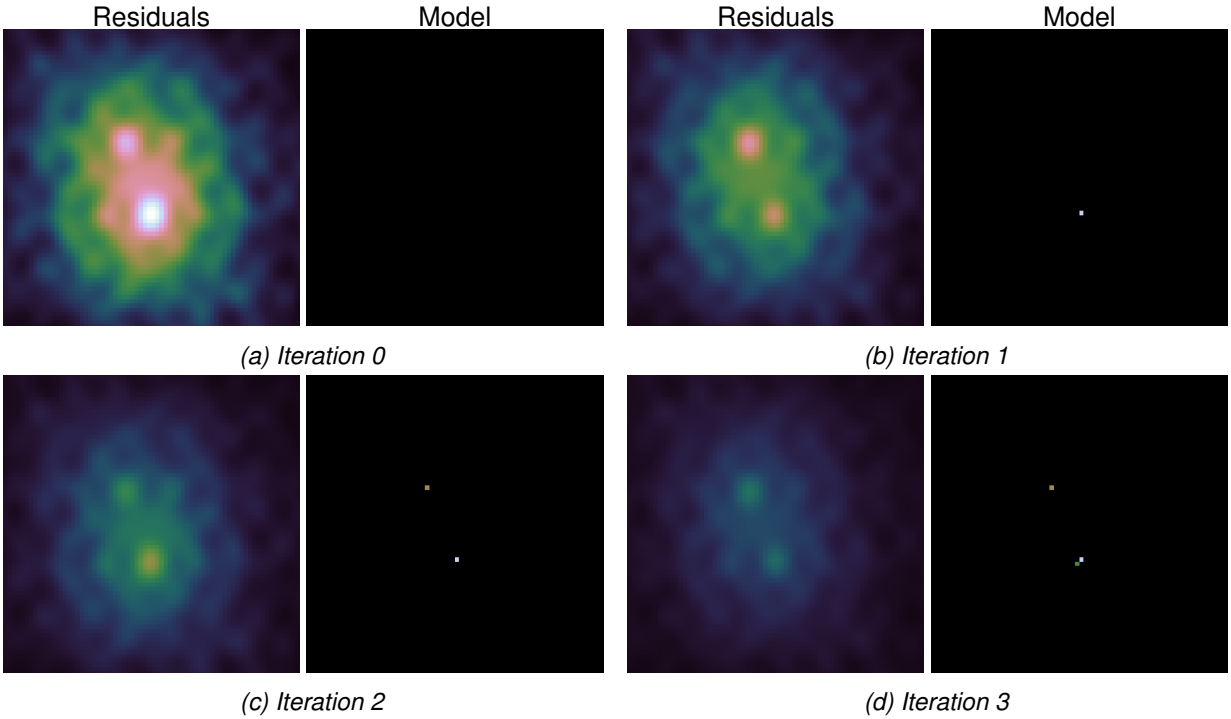
The CLEAN deconvolution algorithm keeps three images in memory: The residual image, the model image, and the PSF. In general, all three images are the same size. At the start of the algorithm, the residual image is equal to the dirty image shown in Figure 5a. The model image starts with every pixel set to zero. It is updated in each iteration and will, at the end, contain the deconvolved version of the dirty image.

A CLEAN iteration begins with searching the maximum pixel in the residual image. In the next step, it adds a fraction of the maximum value to the model image. For example, if the maximum pixel is 2.7, CLEAN adds 1.35 to the model image at the same location (This fraction is called the CLEAN gain and is one of the parameters left for the user to define). The last step is to subtract the PSF at the location. The Figure 6 shows the residual and the model image over four iterations.

After a number of CLEAN iterations, the model image contains a number of non-zero pixels. But CLEAN also leaves artifacts in the model image. As we see in Figure 6d, iteration 3 adds another non-zero pixel close to the already detected one. CLEAN brushes over the artifacts by simply blurring the model image with the ”CLEAN-beam” after CLEAN has finished its iterations. The CLEAN-beam represents the accuracy limit of the radio interferometer. By blurring the image, CLEAN reconstructs the image at the limit of the instrument. The blurring with the CLEAN-beam is shown in Figure 7

The CLEAN algorithm assumes the image contains point sources. In each iteration, it essentially adds the most-likely point source. It is a greedy algorithm, because it searches the most-likely point source in each iteration. It approximately minimizes the deconvolution objective of (2.5), and uses a similar regularization to the L1 norm. The regularization in CLEAN is implicit in the algorithm. In practice, CLEAN implementations use several heuristics, which makes an exact analysis of the regularization difficult.

T



(a) Iteration 0 *(b) Iteration 1*

(c) Iteration 2 *(d) Iteration 3*

Figure 6: CLEAN deconvolution iterations.

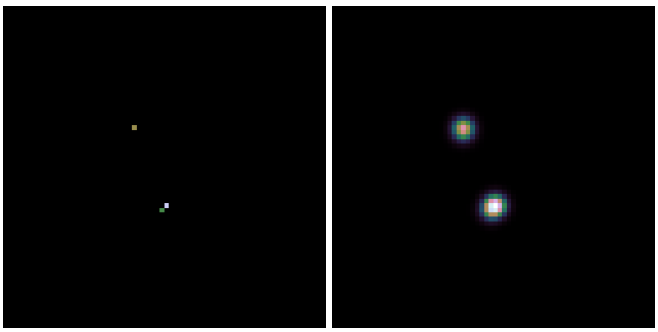


Figure 7: Blurring with the CLEAN-beam

CLEAN is a non-convex optimization

2.3.2 The Major/Minor cycle

During CLEAN deconvolutions, the algorithm assumes that the PSF is constant over the image. However, this is not the case for wide field-of-view observations. The w-term changes the PSF depending on the location in the image. CLEAN deconvolutions only approximate the true PSF. With more CLEAN iterations the residuals image becomes less accurate. To solve this issue, a deconvolution algorithm is used inside the Major/Minor cycle framework.

The Minor cycle is responsible for deconvolution. Typically, a CLEAN based algorithm is used. Every CLEAN iteration is then called a ”Minor” cycle. After a number of CLEAN iterations, the residual image is too inaccurate. Then, the Major cycle takes the model image of CLEAN, and transforms it back to the original Fourier space, called the ”model visibilities. It then subtracts the model visibilities from the measured visibilities, and transforms the residual visibilities back to the image space. Figure 8 shows the Major/Minor cycle framework in more detail.

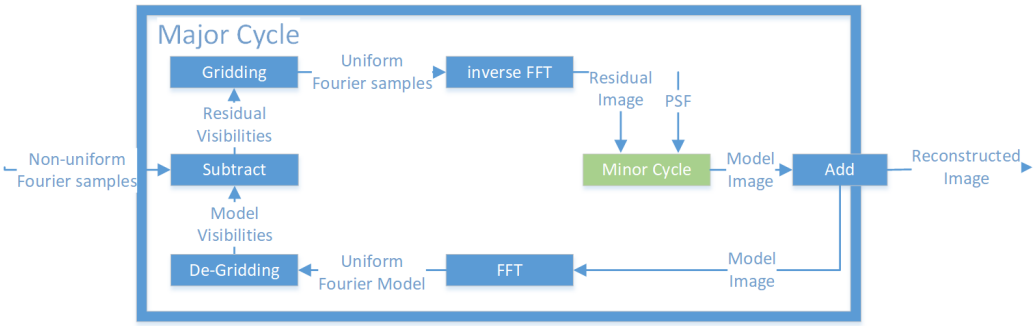


Figure 8: Major/Minor cycle

The major cycle has updated the residuals, and the deconvolution algorithm can continue. The Major cycle allows us to use an approximate PSF in the deconvolution algorithm. Typically, we need several thousand Minor cycles, and between three to ten major cycles to reconstruct an image. The exact numbers depend on the observation itself.

We already discussed the Minor cycle algorithm in detail, now let us turn to the Major cycle algorithm. The Major cycle uses two steps to transform the visibilities to image space. It first takes the visibilities and interpolates them on a uniform Fourier grid. This step is called ”Gridding”. Next, the inverse Fast Fourier Transform (FFT) is applied, resulting in the residual image. The combination of gridding and FFT is faster than any direct transformation from visibility to image space. Note that in this architecture the gridding step has to handle the w-term.

The gridding step and the Minor Cycles are the two bottlenecks in the reconstruction. The gridding step has to interpolate the visibility measurements, which generally has magnitudes more measurements than pixels, and correct for the w-term. The Minor Cycle can become more expensive if the image contains a large number of sources. Depending on the image, it can even become more expensive than the gridding step [7].

The gridding and the Minor cycle deconvolution algorithm are an active field of research in radio astronomy. Recently, the Image Domain Gridder (IDG)[12] has been developed. It uses GPU-acceleration to speed up the gridding step, shifting the bottleneck to the Minor cycle.

Our project focuses on the Minor cycle deconvolution algorithm. The Major cycle allows us to use an approximate PSF in the deconvolution. Our hypothesis is we can approximate the PSF further to simplify the deconvolution problem, and facilitate a deconvolution algorithm in the Minor cycle that uses distributed and GPU-accelerated computing.

3 State of the Art image reconstruction

We present here state of the art deconvolution algorithms. All implemented in the WSCLEAN software package. The two state-of-the-art deconvolution algorithms will be compared to our serial and parallel coordinate descent algorithms later in Section 7 on a real-world MeerKAT observation.

3.1 Multi-scale CLEAN

We introduced the standard CLEAN algorithm in the previous Section 2.3.1. Multi-scale CLEAN is an extension of the standard CLEAN algorithm. The extension allows for accurate modeling of extended emissions. The standard CLEAN algorithm has difficulties reconstructing extended emissions accurately: It assumes the image consists of point sources. When it encounters an extended emission, it simply approximates it with a cloud of point sources.

Multi-scale CLEAN solves this issue. It deconvolves the image at different resolutions (scales). The new algorithm consists of two parts: It first selects a scale (resolution) to deconvolve the dirty image. Then it performs several standard CLEAN iterations at the selected scale. If it selects the lowest scale, the CLEAN iterations are identical to standard CLEAN and point sources are added to the model image. However, if it selects a higher scale, it then adds a 2d Gaussian shaped emissions to the model image.

In pseudo-code, the multi-scale CLEAN idea leads to the following algorithm:

```
1  dirtyImage = iFFT(Gridding(visibilities))
2  model = new Array[,]
3
4  do
5    //select a scale. Large scales mean lower resolutions
6    selectedScale = 0
7    selectedMaxValue = 0
8    for each scale in scales
9      scaledDirty = Convolve(dirty, Gaussian(scale))
10     maxValue = Max(scaledDirty) * ScaleBias(scale)
11     if(selectedMaxValue < maxValue)
12       selectedMaxValue = maxValue
13       selectedScale = scale
14
15   //standard CLEAN iteration, but the dirty image gets blurred
16   scaledDirty = Convolve(dirty, Gaussian(scale))
17   scaledPSF = Convolve(PSF, Gaussian(scale))
18   scaledModel = CLEAN(scaledDirty, scaledPSF)
19
20   //update dirty and model image
21   standardModel = Convolve(scaledModel, Gaussian(scaledModel))
22   model = model + standardModel
23   dirtyImage = dirtyImage - Convolve(standardModel, PSF)
24 while
```

Note the scale-bias function: Modern multi-scale CLEAN implementations prioritize the scales, which improves the reconstruction quality [7]. Usually, the larger scales are prioritized before the lowest scale, where the algorithm is adding point-sources to the model image.

This is the multi-scale CLEAN algorithm widely used for radio interferometric reconstructions. Any real-world implementation uses various strategies that improve the convergence speed or the reconstruction quality of multi-scale CLEAN algorithm. They are too numerous to list them in this work, and we refer the reader to the CLEAN literature [5, 13, 14].

3.2 MORESANE

MOdel REconstruction by Synthesis ANalasys Estimators (MORESANE) is another multi-scale deconvolution algorithm used in radio astronomy. Instead of deconvolving the image at different scales like multi-scale CLEAN, MORESANE uses a regularization with multi-scale built into it. It uses the starlet transform [15] as regularization.

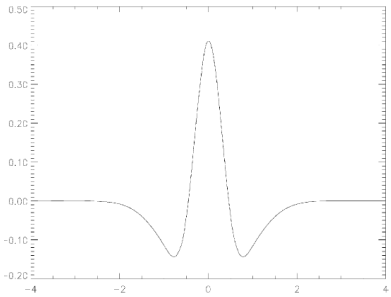


Figure 9: The starlet wavelet in one dimension. Source: [15]

The starlet transform represents an image as a combination of starlet wavelets (shown in Figure 9) at different locations and scales. At the lowest scale, each starlet wavelet is a single pixel wide. At larger scales, each starlet wavelet spans over several pixels. It is an over-complete representation, because we can use several different combinations of starlet wavelets at different scales that lead to the same image.

Normally, we can only retrieve the image from the over-complete representation and not the other way around (because there are different combinations of starlets that lead to the same image). The starlet transform has a way to estimate its wavelet components from the dirty image. The MORESANE algorithm uses this property for reconstruction.

```
1  dirtyImage = iFFT(Gridding(visibilities))
2  model = new Array[,]
3
4  do
5    //estimate starlet components, dimensions: [x, y, scale]
6    estimates = StarletTransform(dirtyImage, PSF)
7    maxIndex = ArgMax(estimates)
8
9    //find connected starlet components. Flood fill over the image(x, y) and over
         scales. Now we should have all starlets belonging to a single object (e.g. a
         hydrogen cloud or a single galaxy in the image)
10   connectedStarlets = FloodFill(starlets, maxIndex)
11
12   optimizedPixels = ConjugateGradients(connectedStarlets)
13
14   //update dirty and model image
15   model += optimizedPixels
16   dirtyImage = dirtyImage - Convolve(optimizedPixels, PSF)
17 while
```

First, it uses the starlet transform on the dirty image, and search for the largest starlet component at a certain scale. The largest starlet component explains part of a single object emitting radio waves, like a hydrogen cloud or a galaxy. Next, we use the flood-fill algorithm to find all starlet components at all scales which describe this object.

The next step is the actual minimization step in the MORESANE algorithm: We use the conjugate gradient method and find the pixel patch, which explains the estimated starlet components. It is a convex optimization

problem, similar to what we described in Section 2.1. But instead of searching for a whole image, we only minimize over a subset of the pixels:

$$\underset{\bar{x}}{minimize} \ \|s_{estimate} - NS^{\mathsf{T}}(\bar{x} \star PSF)\|_2^2 \quad \bar{x} \geq 0 \tag{3.1}$$

Where $\bar{x}$ is the subset of the image we minimize, S^{T} is the starlet transform which transforms an image to starlet space, N is a diagonal matrix which masks all non-zero starlets from the starlet transform, and $s_{estimate}$ are the connected starlet components we estimated. MORESANE reconstructs an image patch which most closely resembles the estimated starlet components.

Non-zero pixels, often used in radio interferometric image reconstruction. The MORESANE algorithm presented here is not the truth. (3.1) is taken from the original paper [4], which is already an ”approximation”.

We know the implementation in the WSCLEAN package leaves non-zero pixels in the model image.

4 Coordinate descent deconvolution

In this section introduces the coordinate descent methods. We derive a coordinate descent deconvolution algorithm which replaces CLEAN in the Major/Minor cycle architecture.

Coordinate descent is a group of numerical minimization algorithm. In each iteration, they minimize the problem along a single coordinate. In our deconvolution problem, a coordinate descent method minimizes the objective function along a single pixel in each iteration. Over several iterations, possibly minimizing the same pixel multiple times, the coordinate descent method converges to a deconvolved image.

We call the algorithm derived in this section 'serial coordinate descent'. In each iteration, the serial coordinate descent algorithm minimizes exactly one pixel. Although it is called serial, we can use multiple processors for minimizing a single pixel. In Section 6 we introduce more sophisticated parallel coordinate descent deconvolution algorithm. Parallel coordinate descent methods update multiple pixels in parallel in each iteration.

As we mentioned before, a deconvolution algorithm in radio astronomy has three components: An objective function, a regularization and a numerical optimization algorithm, which is based on coordinate descent. The objective function we use for our method is:

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

We want to find the minimum deconvolved image x, which is as close to the dirty image (remember the dirty image results from the visibilities, which get gridded and transformed to image space), but also has the smallest regularization penalty. 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.

We introduce the elastic net regularization in the next Section, and then continue with the serial coordinate descent deconvolution algorithm in Section 4.2. We then go over the efficient CPU, GPU and distributed implementations of the serial coordinate descent algorithm.

4.1 Elastic net regularization

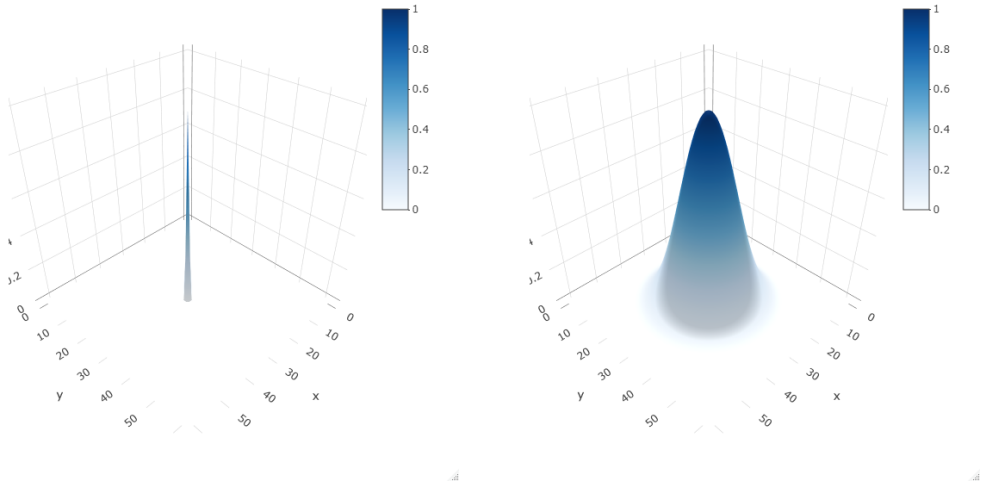
This regularization is a mixture between the L1 and L2 regularization. The L1 regularization is the absolute value of all pixels ($\|x\| = \sum_i \sum_j \|x_{ij}\|$), and the L2 norm is the squared sum of all pixels ($\|x\|_2 = \sum_i \sum_j \|x_{ij}\|^2$). he Figure 10 shows the effect of the L1 and 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.

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[16], 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:

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



(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 10: Effect of the L1 and L2 Norm separately.

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.

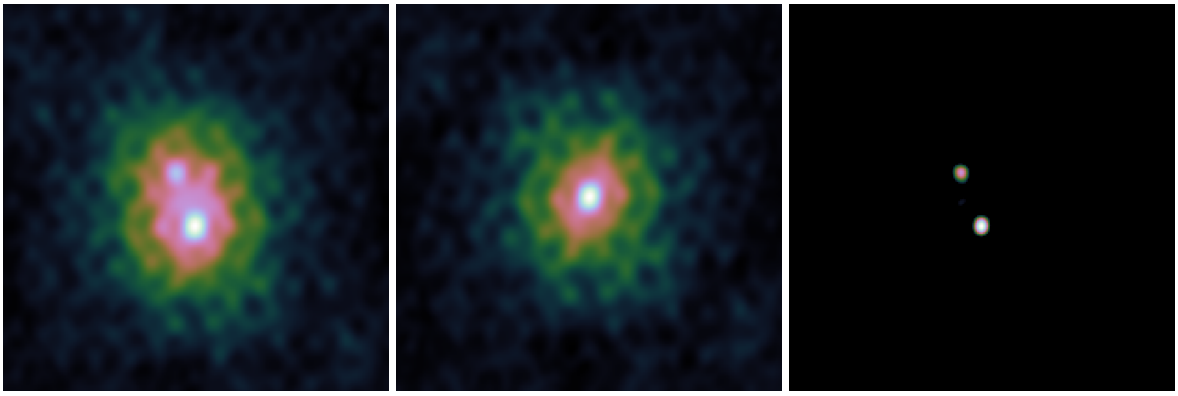
The elastic net regularization is separable. We can calculate its penalty for each pixel independently of its neighbors. As such, its proximal operator is also independent of the neighbors. It is the only regularization we use in this project. We now derive a coordinate descent based deconvolution algorithm 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 [17].

4.2 Serial coordinate descent algorithm

Our serial coordinate descent deconvolution algorithm minimizes the deconvolution objective (4.1). 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 11a shows the dirty image of two point sources, and Figure 11b the PSF. The deconvolved image with elastic net regularization is shown in Figure 11c. I.e. Figure 11c is the optimum x of the objective function (4.1).



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

Figure 11: Example problem with two point sources.

Note that our algorithm 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[2]. 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[18, 19]. 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.

[2]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 6

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 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 (4.1) 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 11, 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  diff = 0
8  do
9    oldObjectiveValue = objectiveValue
10
11   //Step 1: Search pixel
12   pixelLocation = GreedyStrategy(residuals, PSF)
13   oldValue = x[pixelLocation]
14   shiftedPSF = Shift(PSF, pixelLocation)
15
16   //Step 2: Optimize pixel
17   gradient = Sum(residuals * shiftedPSF)
18   lipschitz = Sum(shiftedPSF * shiftedPSF)
19   tmp = gradient + oldValue * lipschitz
20   optimalValue = Max(tmp - lambda*alpha) / (lipschitz + (1 - alpha)*lambda)
21   x[pixelLocation] = optimalValue
22
23   //housekeeping
24   diff = newValue - oldValue
25   residuals = residuals - shiftedPSF * (optimalValue - oldValue)
26 while epsilon < Abs(diff)
```

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.

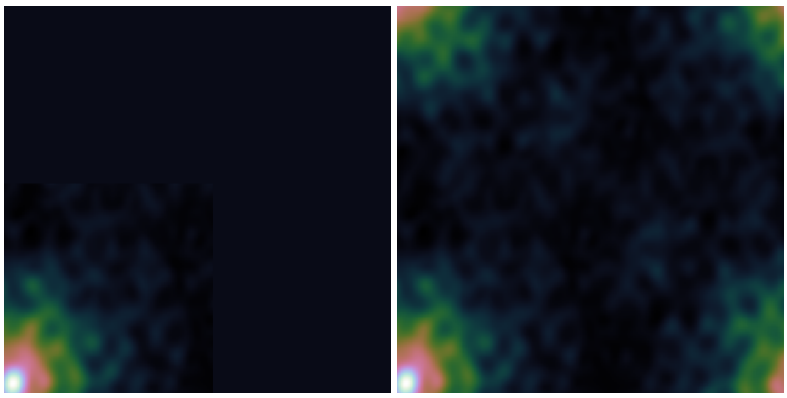
4.3 Efficient implementation

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 12: 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.

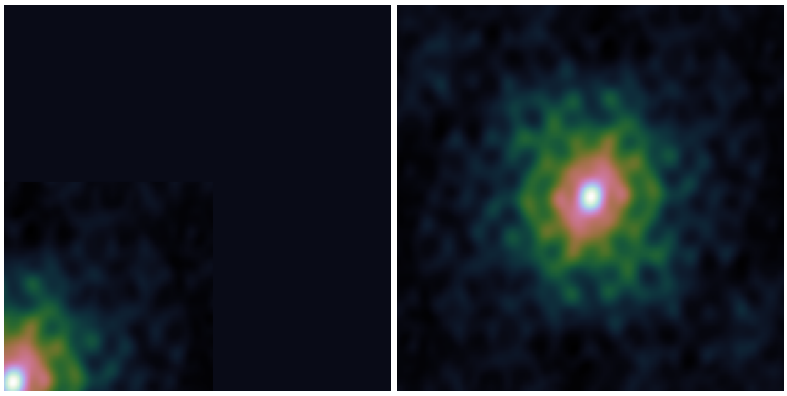
Zero padding assumes that after the edge, the image is zero. Non-zero pixels at the right edges of the image do not influence the left edge after convolution. This is the physically plausible solution. However, the zero padded convolution is more expensive to calculate. We either have to calculate the 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 [20]. 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 $\langle PSF_{location}, PSF_{location} \rangle$ 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 13a shows the PSF shifted to a pixel location. The Lipschitz constant is by squaring all values of Figure 13a 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 13: Sum of squared values for the Lipschitz constant.

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 13b. 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.

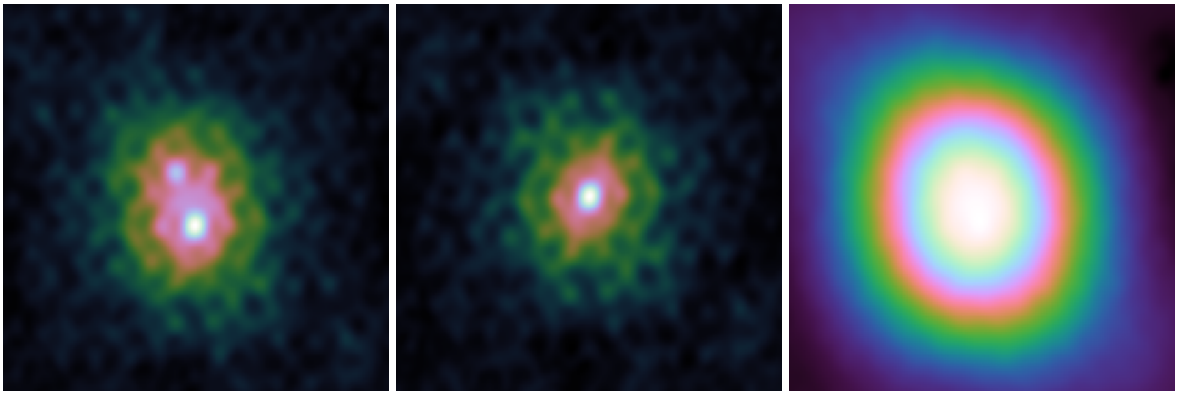
Efficient calculation

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 14.



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

Figure 14: Example of the gradient calculation.

Direct update of gradient map

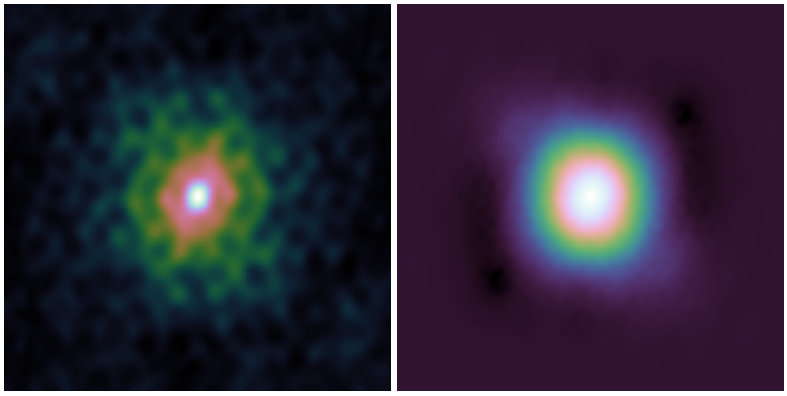
Thanks to the FFT, we can efficiently calculate the map of gradients at the start of the serial coordinate descent algorithm. After we update a pixel in the reconstruction x, the map changes. We could repeat the correlation in the Fourier space from equation (4.7) at each iteration. But this is wasteful. 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 15b.

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 15b.

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



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

Figure 15: Example problem with two point sources.

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 7.

4.4 Efficient implementation pseudo-code

Now we put all the run time improvements discussed before into the new implementation of the serial coordinate descent algorithm. We pre-calculate the Lipschitz constants and the gradient map. Then during iterations, we update the gradient map directly.

This leads to the following algorithm:

```
1  dirty = IFFT(GridVisibilities(visibilities))
2  residualsPadded = ZeroPadding(dirty)
3
4  psfPadded = ZeroPadding(PSF)
5  psfPadded = FlipUD(FlipLR(psfPadded))
6  gradientUpdate = iFFT(FFT(ZeroPadding(PSF)) * FFT(psfPadded))
7
8  x = new Array[,]
9  gradientsMap = iFFT(FFT(residualsPadded) * FFT(psfPadded))
10 lipschitzMap = CalcLipschitz(PSF)
11
12 objectiveValue = 0.5* Sum(residuals * residuals) + ElasticNet(x)
13 maxAbsDiff = 0
14 do
15   oldObjectiveValue = objectiveValue
16
17   //Step 1: Search pixel
18   maxAbsDiff = 0
19   maxDiff = 0
20   pixelLocation = (-1, -1)
```

```
21   for(i in Range(0, dirty.Length(0))
22     for(j in Range(0, dirty.Length(1))
23       oldValue = x[i, j]
24       tmp = gradientsMap[i, j] + oldValue * lipschitzMap[i, j]
25       optimalValue = Max(tmp - lambda*alpha) / (lipschitz[i, j] + (1 - alpha)*lambda
             )
26       diff = optimalValue - oldValue
27
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 epsilon < maxAbsDiff
```

In step 1, we replaced the gradient and Lipschitz calculation with lookups, reducing the runtime costs of the step. In this implementation, step 2 is trivial. We only need to update the reconstruction at the correct location. All the important work has already been completed in step 1.

The two most time consuming parts of this implementation are step 1, and updating the gradient map. Our implementation in .Net Core uses parallel computing for both parts. We search for the maximum pixel in parallel, and update the gradient map in parallel.

4.5 GPU implementation

We implemented the serial coordinate descent algorithm on the GPU. It is implemented in .Net Core with ILGPU[21]. ILGPU is a Just-In-Time compiler for high performance GPU programs written in .Net Core.

GPU programs are split into kernels. Each kernel consists of a single routine optimized for executing on the GPU. Our serial coordinate descent algorithm consists of Step 1, find the best pixel to optimize, step 2, optimize pixel and then updating the gradient map. The GPU 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, this step is a max reduce operation: We want to find the pixel with the maximum absolute step. We implemented the step 1 kernel with an atomic-max instruction:

```
1 MaxPixelKernel(x, gradienstMap, lipschitz, location, maxPixel)
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   AtomicMax(maxPixel, currentPixel)
```

The kernel is executed on multiple processors in parallel on the GPU. Each processor is checking a single pixel. Communication is done with the atomic-max instruction. The atomic-max writes on a global variable, which keeps track of the current maximum pixel. This implementation turned out to be the fastest for the MaxPixelKernel. Warp shuffle[22] was also tested, but resulted in a slower kernel.

Now to put the serial coordinate descent implementation on the GPU together, all we need to do is call the kernels, which perform the minimization on the GPU:

```
do
  //Step 1: Search pixel
  maxPixel = (absDiff = 0, diff = 0, location = (-1, -1))
  ExecuteMaxPixelKernel(x, gradienstMap, lipschitz, maxPixel)
  SynchronizeKernels()

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

while maxPixel.absDiff < epsilon
```

Note that after each kernel call, we synchronize the program. We wait for all kernels on the GPU to be finished, before we continue with the next step. As we mentioned before, this is the core behind the serial coordinate descent algorithm. We have to wait for each step to finish before we can continue with the next.

4.6 Distributed implementation MPI

We created a distributed implementation of our serial coordinate descent algorithm using the Message Passing Interface (MPI). In MPI, we use several nodes of computers to solve the problem. Each node has its own processors an main memory, and we use MPI to communicate betweeen the nodes.

We split the reconstructed image, and the gradient map into facets of equal size, and use one node for each facet. For example, our image is 1024^2 pixels in size and we use 4 nodes. This means each node reconstructs a 512^2 facet of the image, and has the matching 512^2 entries of the gradient map locally.

The distributed serial coordinate descent algorithm now searches the maximum pixel locally in each node. Then, all node agree on the best global pixel to optimize, which is an MPI_ALLREDUCE operation. After MPI_ALLREDUCE, each node knows the global pixel which gets minimized in this iteration. Then, each node updates its part of the gradient map locally.

This leads to the following pseudo-code algorithm:

```
...
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
          )
```

```
14        diff = optimalValue - oldValue
15
16        if(maxAbsDiff < Abs(diff))
17          maxAbsDiff = Abs(diff)
18          maxDiff = diff
19          pixelLocation = (i, j)
20
21    //communicate location with nodes
22    globalMaxAbsDiff, globalMaxDiff, globalLocation = MPI_ALLREDUCE(maxAbsDiff,
          maxDiff, pixelLocation)
23
24    //Step 2: Optimize pixel.
25    if(globalLocation == pixelLocation)
26      x[globalLocation] += globalMaxDiff
27
28    //housekeeping
29    shiftedUpdate = Shift(gradientUpdate, globalLocation)
30    gradientMap = gradientMap - shiftedUpdate * globalMaxDiff
31  while epsilon < maxAbsDiff
```

Each node in the distributed serial coordinate descent algorithm communicates its local pixel in each iteration. Meaning the distributed algorithm has a communication step in each serial coordinate descent iteration. The distributed algorithm only has to communicate a single pixel, but has to communicate often. This is a potential down-side of the implementation. Usually, distributed systems can speed up algorithms which communicate a large chunk of data rarely.

4.7 Serial coordinate descent and similarities to the CLEAN algorithm

When we look back at the CLEAN algorithm described in Section 2.3.1, we can see similarities in their structures: In the first step, CLEAN searches the location of the maximum pixel in the residuals, while serial coordinate descent searches the location of the absolute maximum change. In the second step, CLEAN subtracts a fraction of the PSF at that location, while coordinate descent calculates the optimal pixel value and subtracts the product of $PSF \star PSF$ from the gradient map.

CLEAN and serial coordinate descent have the same overall structure. But serial coordinate descent uses the gradient map instead of explicit residuals (the gradient map contains the information of the residuals implicitly). In a sense, the serial coordinate descent can be looked at as a CLEAN algorithm, which uses the gradient map instead of the residual map, and the product of $PSF \star PSF$ instead of the PSF. Both algorithms use essentially use the same operations, but the content in their memory is different. As such, one iteration of standard CLEAN is roughly as expensive as an iteration of serial coordinate descent.

The side effect of their similar structure is that any speedup we achieve by using GPU acceleration for serial coordinate descent can plausibly be used to speedup CLEAN to a similar amount. Our distribution scheme for serial coordinate descent can also be used for a distributed standard CLEAN.

The main difference between the two algorithms is that serial coordinate descent does not need the blurring step typically used in CLEAN algorithms. In Section 2.3.1, we introduced the CLEAN concept of the model image. Standard CLEAN detects point sources, and puts them in the model image. When the observation shows an extended emission, the standard CLEAN approximates it as a set of point sources. After CLEAN has finished finding all the point sources, it blurs the model image with the clean-beam, a 2d Gaussian representing the accuracy of the instrument, resulting in the reconstructed image.

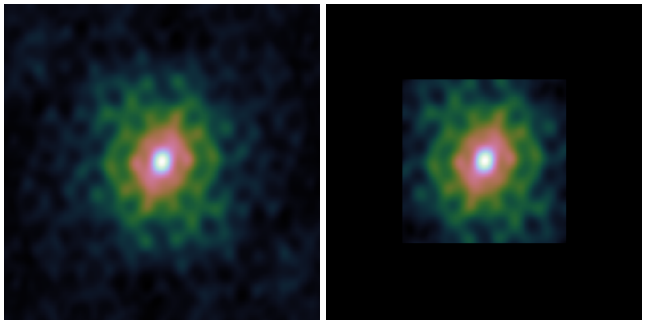
In this project, we use the elastic net regularization, which has a simple model for extended emissions, namely the L2 norm. As such our serial coordinate descent algorithm can find the reconstructed image directly without

any blurring. This may result in super-resolved reconstructions: The serial coordinate descent algorithm may find structures which are smaller than the accuracy limit of the instrument. We will show plausible super-resolved reconstructions by the serial coordinate descent algorithm with elastic net regularization in Section 7.

5 PSF approximation for fast and distributed deconvolution

This section describes our main hypothesis of this work: Our hypothesis is that we can approximate the PSF in the deconvolution problem and exploit it to speed up/distribute the deconvolution.

Our reasoning for the hypothesis comes from the Major/Minor cycle architecture. The major cycle calculates the dirty image from the visibilities. The dirty image is the product of the observed image convolved with the PSF of the instrument. The deconvolution algorithm assumes the PSF is constant over the image, but modern interferometers have a w-term, which changes the PSF depending on the pixel. This is why we need the major cycle. After a number of Minor cycles (iterations of the deconvolution algorithm), we use the Major cycle to correct for the errors we introduced with the constant PSF. The Major cycle allows us to use an approximate PSF in the deconvolution problem. We believe this can be exploited to speed up/distribute the deconvolution.



(a) Full PSF *(b) $\frac{1}{2}PSF$ approximation*

Figure 16: PSF approximation typically used in Clark CLEAN

A similar idea has been in use in CLEAN for decades. The Clark CLEAN algorithm [13]. Uses only a window of the PSF for the deconvolution. Figure 16 shows the full PSF and the approximate PSF used typically in Clark CLEAN. During deconvolution, it only uses a window around the center of the full PSF. Typically, the sides of the window is $\frac{1}{2}$ of the image size. By using only a $\frac{1}{2}PSF$ window around the center, a Clark CLEAN iteration is significantly faster than standard CLEAN. With $\frac{1}{2}PSF$, Clark CLEAN has to subtract only the $\frac{1}{2}PSF$ window from the residuals, which is 4 times smaller than the full PSF.

Using only a fraction fo the true PSF speeds up the deconvolution. But it also introduces sparsity in the deconvolution problem which, to our knowledge, has not been explored for radio interferometers. The full PSF shown in Figure 16 has significant values around the center, but they quickly approach zero the further away we move from the center. If we only use a window around the center $\frac{1}{2}PSF$ and set the rest to zero, we are using a sparse PSF. For a CLEAN algorithm, the sparse approximated PSF ignores the influence of far away pixel. If we increase the approximation, the deconvolution problem becomes increasingly separable into image facets. With a $\frac{1}{8}PSF$, we can split the image into 16 facets, 4 of which are independent from each other. This problem can be easily distributed onto 4 nodes, each deconvolving an independent facet in parallel.

The important question is, how small can the center window be? Can we guarantee that the algorithm converges to the same solution, even with an approximated PSF? We will test the effect of an approximate PSF with our serial coordinate descent algorithm on a real-world observation in Section 7.3. In this section, we show how we can incorporate an approximate PSF into our serial coordinate descent algorithm.

5.1 PSF approximation for the serial coordinate descent algorithm

The serial coordinate descent algorithm keeps the gradient map, the product $PSF \star PSF$ and the current reconstruction in memory. In each iteration, the algorithm first finds the pixel, which has the maximum possible

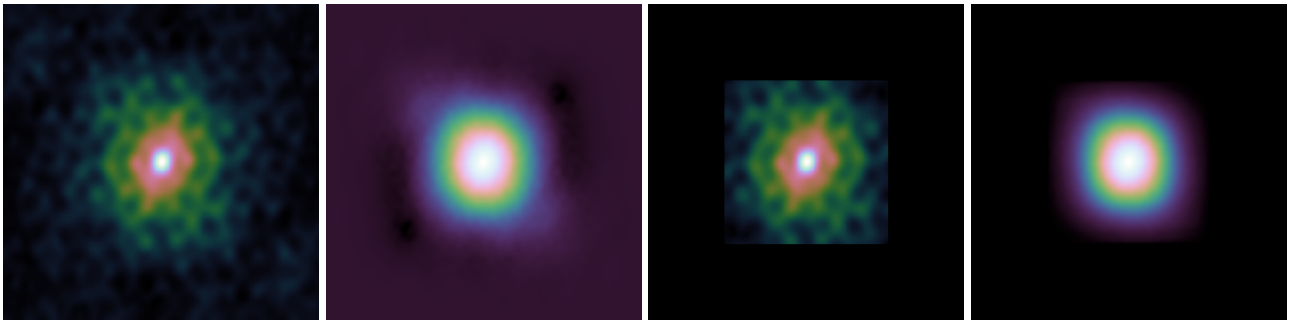
difference in this iteration. In the second step, it subtracts the product $PSF \star PSF$ at the correct location of the gradient map. The gradient map is now updated for the next iteration of the serial coordinate descent algorithm.

We approximate the PSF by only using a window around the center. Each side is only a fraction of the full PSF size. From now on, we use the method $Cut()$, which cuts out the center window of the PSF. If we use a cut fraction of $\frac{1}{4}$, we cut out a window of the PSF, each side being $\frac{1}{4}$ the length of the full PSF.

Using an approximate PSF influences three parts of the algorithm: The gradient map, the Lipschitz map and the product $PSF \star PSF$. At the beginning of the algorithm, we calculate the gradient map by correlating the dirty image with the PSF ($I_{dirty} \star PSF$) We also calculate the Lipschitz constants for each pixel. The product $PSF \star PSF$ is used to update the gradient map in each iteration. We developed two approximation methods for the serial coordinate descent algorithm. Method 1 is called 'approximate update', and method 2 is called 'approximate deconvolution'.

5.1.1 Method 1: Approximate update

The approximate update method only uses the approximate PSF for updating the gradient map. Instead of using the product $PSF \star PSF$, this method uses the product $Cut(PSF) \star Cut(PSF)$ to update the gradient map in each iteration. Figure 17 shows the effect of the approximation.



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

Figure 17: Approximation of gradient update.

This method uses the full PSF to initialize the gradient and the Lipschitz map. But it only uses an approximation for the update in each iteration of serial coordinate descent. The error we introduce with the approximation gets more severe with more iterations of serial coordinate descent. But the upside is the first iteration in every Major cycle is always identical to the serial coordinate descent algorithm without approximation. This means with enough major cycles, we are guaranteed to converge to the same result.

5.1.2 Method 2: Approximate deconvolution

This method also uses the approximate gradient update. But instead of initializing the gradient map with the full PSF, it also uses the approximate PSF for initializing the gradient and the Lipschitz map. In essence, this method solves an approximate deconvolution problem:

$$\underset{x}{minimize}\ \frac{1}{2} \|I_{dirty} - x * Cut(PSF)\|_2^2 + \lambda ElasticNet(x) \tag{5.1}$$

This method does not introduce an error in every serial coordinate descent iteration. But it comes with the downside: It is not guaranteed to converge to the same result as the serial coordinate descent without approximation. It systematically under-estimates the pixel values in the reconstructed image.

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 in theory. But it does not completely remove the issue for extended emissions.

5.1.3 Combining the two approximation methods

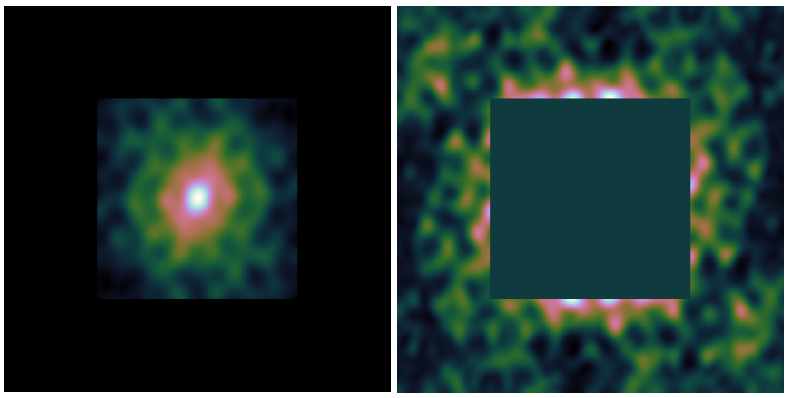
The two approximation methods developed here have opposing downsides: Method 1 is guaranteed to converge to the same result, given enough major cycles, but becomes increasingly inaccurate with more serial coordinate descent iterations. Method 2 does not become increasingly inaccurate, but is not guaranteed to converge to the same result, even with an infinite number of Major cycles.

The obvious question is, what happens when we combine the two approximation method. Indeed, this is our final solution in this project. We start out with method 2, the approximate deconvolution for a few Major cycles, and then switch to method 1, approximate update. When combining both methods, we decided to switch from method 2 to method 1 when the serial coordinate descent has converged.

5.2 Major Cycle convergence and implicit path regularization

There is one problem with the approximate PSF which is left to solve: When does the serial coordinate descent algorithm decide to use a Major cycle. Or when we combine the PSF approximation methods, how many Major cycles do we use for each method?

When we use an approximate PSF, the deconvolution algorithm will at a certain iteration start to include 'side lobes' of the PSF. The Figure 18 shows an example of the side lobes we introduce by approximating the PSF with $\frac{1}{2}$ of the center window. At a certain point, the deconvolution with an approximate PSF has to decide whether the emission is real, or whether it is an artifact from the PSF approximation, and will be removed with the next major cycle.



(a) $\frac{1}{2}PSF$ approximation *(b) PSF Sidelobes*

Figure 18: Maximum sidelobe of the PSF cutoff.

For example: if we have an observation with a single point source in the center, the first iteration of Clark CLEAN will subtract the approximate PSF from the residuals. But the residual image is left with the PSF side lobes shown in Figure 18b. The next iteration may detect 'fake' point sources at the significant side lobes of the PSF. Our serial coordinate descent algorithm does not explicitly use the residual image, but has a similar problem. The information of the residual image is implicitly contained in the gradient map. Our serial coordinate descent algorithm with an approximate PSF leaves significant gradients in the map.

This is a problem for deconvolving an image with the Major cycle. If we let our serial coordinate descent algorithm converge to a result in each Major cycle, it will include emission from the PSF side lobes. In the next Major cycle, the serial coordinate descent algorithm has to remove the side lobes from the reconstruction, but this again leaves significant side lobes in the gradient map. This can lead to an oscillation of the algorithm, where it adds and removes the same side lobes several times over different Major cycles. In extreme cases (for example when we use an aggressive approximation of the PSF), this may even lead to a diverging behavior. But even if the deconvolution algorithm converges over several Major cycles, if the algorithm spends too much time on PSF side lobes, it may become significantly slower.

To solve this problem, we use the following strategy for our serial coordinate descent algorithm: We estimate the PSF side lobes introduced by the approximation. We increase the regularization parameter λ, until the elastic net regularization excludes all side lobes from the reconstruction. Note that real emission may also be regularized away in the current Major cycle iteration. The serial coordinate descent algorithm reconstructs the image with the current λ, and lets the Major cycle remove the side lobes. It then estimates the current PSF side lobes for this Major cycle, sets a lower regularization parameter λ and again deconvolves the image.

This is known as a path regularization in optimization. We let our optimization algorithm converge on intermediate solution, decrease the λ parameter and use the intermediate solution as a 'warm start'. Coordinate descent methods may benefit from path regularization[23]. Converging on all intermediate solutions may result in a lower wall-clock time than converging directly on the final solution.

We use the path regularization to stop the algorithm from wasting computing resources on side lobes. There may be different strategies that result in an overall lower wall-clock time for the serial coordinate descent algorithm, but they were not explored in this project. We set the regularization parameter λ for each Major cycle according to the following estimate:

$$\begin{aligned} maxSidelobe &= Max(PSF - Cut(PSF)) \\ gradients &= residuals \star PSF \\ maxLobeGradient &= Max(gradients) * maxSidelobe \\ \lambda_{cycle} &= \frac{maxLobeGradient}{\alpha} \\ \lambda_{cycle} &= Max(\lambda, \lambda_{cycle}) \end{aligned} \tag{5.2}$$

We calculate the maximum side lobe of the PSF approximation. We then multiply the maximum side lobe with the maximum gradient. This is an estimate of the largest gradient value, which gets left in the gradient map by the PSF approximation. We then set the λ_{cycle} parameter to exclude all gradients equal or smaller that gradient magnitude. The maximum of the gradient map decreases over every Major cycle, which leads to a decreasing λ_{cycle} until we reached the target value λ.

This estimate works, but it has one problem for radio interferometric imaging: It does not account for extended emissions. Since they have non-zero pixel values close to each other, their PSF side lobes also overlap. Meaning the PSF side lobes of extended emissions are higher than we estimated. This is why we added a

correction factor which estimates how much the maximum in the gradient map is point-source-like:

$$\begin{aligned} maxSidelobe &= Max(PSF - cut(PSF)) \\ gradients &= residuals \star PSF \\ \textbf{correction} &= Max(1, \frac{Max(gradients)}{Max(residuals) * Lipschitz}) \\ maxLobeGradient &= Max(gradients) * maxSidelobe * correction \\ \lambda_{cycle} &= \frac{maxLobeGradient}{\alpha} \\ \lambda_{cycle} &= Max(\lambda, \lambda_{cycle}) \end{aligned} \tag{5.3}$$

It the same estimate as before, except for the correction factor. The correction factor is 1 if the maximum in the gradient map is a point source, and $1 <$ if the maximum in the gradient map is more like an extended emission. The correction factor is based on the following observation: If the image only contains point sources, then the maximum of the gradient map should be equal the maximum of the residuals times the Lipschitz constant. But if it is an extended emission, the maximum of the gradient map will be significantly larger.

This is the estimate we use for the regularization parameter λ_{cycle} for each Major cycle. The parameter λ_{cycle} decreases over each Major cycle, resulting in an implicit path regularization for our serial coordinate descent method.

6 Parallel coordinate descent methods

We demonstrated that the true PSF can be approximated with a window around the center. The window can only be a fraction of the total size of the PSF, and the serial coordinate descent algorithm converges to the same result. The approximation resulted in a speedup factor fo ≈ 1.7. But this speedup is not as large as one might expect. The serial coordinate descent method deconvolve the image with a window of $\frac{1}{16}$ of the full PSF. Or in other words: The approximation method resulted in a sparse PSF for the deconvolution algorithm, but it did not result in a large speedup. In this section, we introduce and develop a parallel coordinate descent algorithm which exploits the sparse PSF to achieve significant speedup.

The serial coordinate descent algorithm minimizes a single pixel in each iteration. Parallel coordinate descent methods can group several pixels into blocks, and minimize multiple blocks in parallel within a single iteration. We introduce the new concepts of the parallel methods in Section 6.1, and show synchronous implementations of the new parallel coordinate descent algorithm. We then show an efficient, and asynchronous implementation of our parallel algorithm in Section 6.2, where each processor updates a block independently of the other processors.

The parallel coordinate descent algorithm from Section 6.2 does not run efficiently on the deconvolution problem. We discuss the problems and our extensions we developed for an efficient parallel coordinate descent deconvolution algorithm in Section 6.3. Lastly, we test our parallel algorithm on the LMC observation. We explore the optimal settings for its tuning parameters in Section 7.4, and how the parallel coordinate descent algorithm scales with large number of processors in Section 7.6.

6.1 From serial to parallel block coordinate descent

We start with our serial coordinate descent algorithm and first modify it to a serial block coordinate descent algorithm. A block is a group of pixels. In our case, a block is always a rectangle of neighboring pixels. For example: A block of 4^2 pixels can be minimized in a single iteration of the serial block coordinate descent algorithm.

Later in Section 6.3 we introduce the main concept which allows us to adapt the serial block algorithm to a parallel block coordinate descent algorithm. We show a synchronous implementation of the parallel coordinate descent algorithm in Section 6.1.3.

6.1.1 Block coordinate descent

Instead of optimizing a single pixel in each iteration, the serial block coordinate descent algorithm can update a block of pixels. We start with the update rule of the serial coordinate descent algorithm, and show how it can be adapted to update a block of pixels in each iteration. Remember the single pixel update from the serial coordinate descent algorithm:

$$pixel_{opt} = \frac{max(gradient_{location} - \lambda\alpha, 0)}{Lipschitz_{location} + (1-\alpha)\lambda} \tag{6.1}$$

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

$$pixels_{opt} = \frac{max(gradients_{locations} - \lambda\alpha, 0)}{Sum(Lipschitz_{locations}) + (1-\alpha)\lambda} \tag{6.2}$$

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). Note that the

larger block we chose, the smaller the update becomes for each individual pixel inside the block. We have a central trade-off: We can take a large step for a single pixel, or take several smaller steps for a block of pixels.

Remember: The Lipschitz constants for neighboring pixels have a similar value. Meaning for a block of $2^2 = 4$ pixels: $Sum(Lipschitz_{locations} \approx 4Lipschitz$. Or less formally, we can either take a full step towards the minimum for a single pixel. Or if we update a block of 2^2 pixels, we take $4\ \frac{1}{4}$ steps towards the minimum.

The reader might be familiar with the (F)ISTA method[24]. The block update shown in equation (6.2) is related to the (F)ISTA update rule. 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.

The block update rule (6.2) allows us to minimize a single block of pixels in each iteration. But it comes with a trade-off: The bigger blocks we choose, the smaller steps we take for each individual pixel in the block. The reason why the serial block coordinate descent may be faster than the single pixel algorithm is when most pixels are correlated with their neighbors: Extended emissions have a large areas where the pixel values are correlated. Meaning if a pixel in the area is non-zero, then the neighboring pixels are also likely to be non-zero. A serial block coordinate descent algorithm can take more useful minimization steps in each iteration. We test different block sizes with our parallel algorithm in Section 7.4. In our tests, different block sizes did not lead to a significant speedup.

6.1.2 Estimated Separability Overapproximation (ESO)

So far, we introduced a serial block coordinate descent algorithm. If we want to update blocks of pixels in parallel, we need to estimate how much the PSFs of parallel updates 'overlap'. For example: If we update two blocks in parallel, and their combined PSFs do not overlap, then the update is independent. Updating the first block, and then the second block in a serial algorithm leads to the same result as updating both blocks in parallel.

However, if we update two blocks, which are located next to each other in the image, then their combined PSFs overlap significantly. Their updates are dependent on each other. If we update the first block, and then the second block in a serial algorithm results in significantly lower pixel values for the second block. Because their PSFs overlap, both blocks try to explain mostly the same emission. If we update both blocks in parallel, each block would try to explain the same emission, and we would over-estimate their pixel values.

This over-estimation can lead to a diverging algorithm. To guarantee the convergence of a parallel block coordinate descent, we need to estimate the overlap of the PSFs of parallel updates. This can be done with the Estimated Separability Overapproximation (ESO) developed in [25]. The ESO estimates how much the PSFs overlap, if we update τ random blocks in parallel:

$$ESO(\omega, \tau, n) = 1 + \frac{(\omega - 1)(\tau - 1)}{max(1, n - 1)} \tag{6.3}$$

Where ω is the number of non-zero entries in the PSF, τ is the number of random parallel updates in each iteration, and n is the number of blocks in the image. Let us use an example to demonstrate what the ESO means: Let us assume the PSF has $\omega = 24$ non zero entries, $\tau = 4$ processors to update in parallel, and the image is 256^2 pixels in size with a block size of 4^2 pixels. Plugging the values into the ESO gives us the following result:

$$ESO(\omega = 24, \tau = 4, n = (256^2/4^2)) = 1 + \frac{(24 - 1)(4 - 1)}{max(1, 4096 - 1)} \approx 1.017 \tag{6.4}$$

An ESO of 1 means the $\tau = 4$ parallel updates are completely independent of each other, and we do not need to account for overlapping PSFs. In our example, we arrived at an ESO of 1.017. This means every parallel

update step has to be divided by 1.017 to account for overlapping PSFs, and ensure convergence.

The ESO only needs to know the number of non-zero components. It is independent of the exact structure of the PSF. The fewer non-zero components the PSF has, the closer it is to 1, and the more effective each parallel update is. The ESO benefits from our PSF approximation. It decreases the number of non-zero components in the PSF and leads to an ESO closer to 1.

However, note that the ESO assumes we choose $\tau = 4$ blocks uniformly at random. Indeed, a uniform random selection strategy is a core assumption for the parallel coordinate descent method[25]. Random selection strategies tend to perform badly on the deconvolution problem. Later in Section 6.3, we develop a pseudo-random selection strategy which does not break the random selection assumption of the ESO, but performs better on the deconvolution problem.

6.1.3 Accelerated parallel block coordinate descent

So far, we introduced the serial block coordinate descent and the ESO. The serial block coordinate descent can update a block of pixels in a single iteration, and the ESO estimates how much PSFs overlap when we perform parallel update steps. In this section, we put this together in an accelerated, parallel block coordinate descent algorithm based on APPROX[26]. But first, we introduce gradient acceleration.

In gradient acceleration, we use the gradient from previous iterations to speed up convergence of the current iteration. We can accelerate our serial coordinate descent algorithm by extending it with an acceleration parameter θ, a copy of the gradient map and a copy of the reconstructed image x. We term one couple of gradient map plus reconstructed image as 'explore', while the other couple is called 'correction'. The 'correction' gradient map and reconstructed image contain gradient information of the previous iterations. They are used to speed up the convergence of the 'explore'. In each iteration, the acceleration parameter θ decreases, and we use more information from the 'correction' gradient map and reconstruction.

This leads to the following accelerated, parallel and block coordinate descent deconvolution algorithm:

```
1  dirty = IFFT(GridVisibilities(visibilities))
2  residualsPadded = ZeroPadding(dirty)
3
4  psfPadded = ZeroPadding(PSF)
5  psfPadded = FlipUD(FlipLR(psfPadded))
6  gradientUpdate = iFFT(FFT(ZeroPadding(PSF)) * FFT(psfPadded))
7
8  xExplore = new Array[,]
9  xCorrection = new Array[,]
10 gradientsMapExplore = iFFT(FFT(residualsPadded) * FFT(psfPadded))
11 gradientMapCorrection = new Array[,]
12 lipschitzMap = CalcLipschitz(PSF)
13
14 eso = ESO(CountNonZero(PSF), t, x.Length / blockSize)
15 theta0 = t / (x.Length / blockSize)
16 theta = theta0
17
18 do
19   oldObjectiveValue = objectiveValue
20
21   //Step 1: select t blocks uniformly at random
22   blocks = sample(t)
23
24   //Step 2: update reconstruction in parallel
25   diffBlocks = new Array
26   parallel for each block in blocks
```

```
    //increase blockLipschitz according to the ESO
    blockLipschitz = Sum(GetBlock(LipschitzMap, block))
    blockLipschitz = blockLipschitz * eso

    oldBlock = GetBlock(xExplore, block)
    tmp = theta^2 * GetBlock(gradientsMapCorrection, block)
      + GetBlock(gradientsMapExplore, block)
      + GetBLock(xExplore, block) * blockLipschitz
    optimalBlock = Max(tmp - lambda*alpha) / (blockLipschitz + (1 - alpha)*lambda)
    diffBlock = optimalBlock - oldBlock

    xExplore[block] += diffBlock
    xCorrection[block] += diffBlock * (-(1.0f - theta / theta0) / theta^2)
    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)

      gradientsMapExplore = gradientsMapExplore - shiftedUpdate * diff
      gradientsMapCorrection = gradientsMapCorrection - shiftedUpdate * diff *
          (-(1.0f - theta / theta0) / theta^2)

  theta = (Sqrt((theta^2 * theta^2) + 4 * (theta^2)) - theta^2) / 2.0f
while maxAbsDiff < epsilon

output = new float[,]
for(i in in Range(0, dirty.Length(0))
  for(j in in Range(0, dirty.Length(0))
    output[i, j] = theta * xCorrection[i, j] + xExplore[i, j];
```

In each iteration, the parallel algorithm first samples τ unique blocks uniformly at random (we cannot select the same block more than in a single iteration). In the second step, we then update each block in parallel. Note that we multiply the block Lipschitz constant with the ESO, which ensures convergence for parallel updates. In the third step, we update the two gradient maps. The final image is a combination of the two reconstructed images x from the 'explore' and 'correction' couple.

This algorithm is parallel, but it is still synchronized: It updates each block in parallel, but waits for all updates to finish before continuing with the next iteration. In the next Section 6.2, we introduce an asynchronous implementation, where the individual processors do not wait for each other.

The accelerated, parallel coordinate descent algorithm reduces itself to a non-accelerated variant, if we do not modify θ in each iteration. In that case, the 'correction' gradient map and reconstruction x stay zero over the course of the algorithm. Note that due to gradient acceleration, we need twice the memory (for the 'correction' maps), and twice the number of operations to update a single block. Gradient acceleration allows us to take larger steps towards the optimum in each iteration. As such, it should need fewer iterations to converge than the non-accelerated variant. But a single iteration of the accelerated variant is more expensive.

6.2 Asynchronous implementation

In this section, we show how the accelerated, parallel coordinate descent algorithm can be implemented asynchronously. Each processor selects a random block, and updates the gradient maps and reconstructed

images independently of the other processors. The asynchronous processors have to communicate three parts: The 'explore' gradient map, the 'correction' gradient map, and what block they have currently selected to update.

For the asynchronous implementation, we introduce the 'blockLocks' map. Each asynchronous processor writes its processor id at the location of the block it is currently updating. A processor can only select a block which is not updated by another processor. The write to the blockLocks map has to be atomic, such as the updates on the 'explore' and 'correction' gradient map. This leads to the following algorithM:

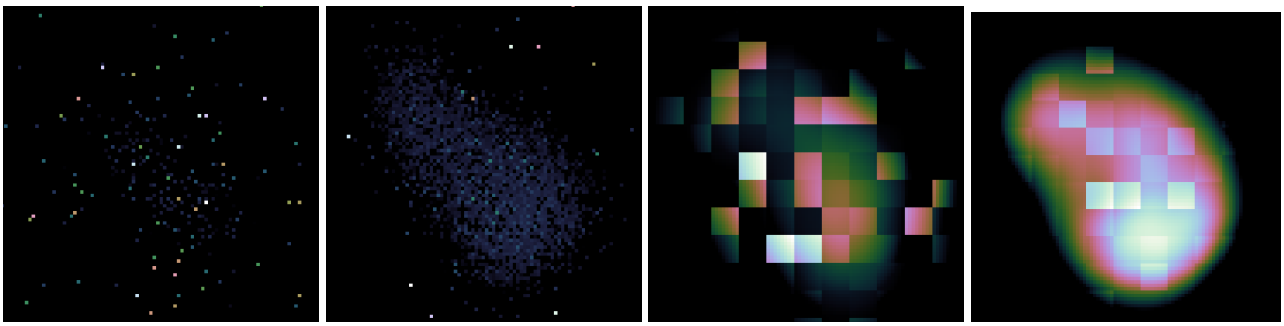
```
...
concurrentIterations = 1000
blockLocks = new Array[,]
...
do
  //asynchronous iterations
  parallelDiffs = new Array[]
  parallel for each processorId in (0:t)+1
    for concurrentIter in 0:concurrentIterations
      block = AtomicLockRandomBlock(blockLocks, processorId)

      ...
      //Step 2: update block according to the same update rule
      ...

      parallelDiffs[processorId -1] = Max(parallelDiffs[processorId -1], diffBlock)

      //Step 3: Update gradients
      for each pixel in block
        diff = diffBlock[pixel]
        shiftedUpdate = Shift(gradientUpdate, pixelLocation)

        AtomicSum(gradientsMapExplore, -shiftedUpdate * diff)
        AtomicSum(gradientsMapCorrection, -shiftedUpdate * diff * (-(1.0f - theta /
            theta0) / theta^2))

      //unlock block
      blockLocks[block] = 0

      theta = (Sqrt((theta^2 * theta^2) + 4 * (theta^2)) - theta^2) / 2.0f

  maxParallelDiff = Max(parallelDiffs)
  ...
while maxParallelDiff < epsilon
```

On modern CPU's, we can use the compare-exchange instruction to ensure atomic writes/updates on the blockLocks and the two gradient maps. If a processor selects a block which does not overlap with the PSF of another selected block, it can update the block with minimal communication costs. In our deconvolution problem, the chance that two processors update the same position in the gradient maps at the same time depends on the size of the PSF. The smaller the PSF, the smaller the chance is that more than one processor tries to update the same position.

With an asynchronous implementation, our parallel coordinate descent algorithm benefits in two ways from a smaller PSF: First, a smaller PSF leads to an ESO closer to 1. With an ESO close to 1, our parallel updates become as efficient as the equivalent number of serial updates. And second, by decreasing the chance of two processors updating the same memory location at the same time, decreasing the communication costs of the algorithm.

6.3 The problem with random selection for deconvolution

Our parallel coordinate descent algorithm developed in this section does not perform well on the LMC dataset. The reason lies in the random selection strategy: In the first few iterations, the deconvolution algorithm selects blocks at random, and tries to explain the whole emission in that area. The emission in this area of the image is 'locked' inside a few blocks. Before the parallel algorithm can make a meaningful update for a neighboring block, it first needs to select the same block again. In short, the first iterations always over-estimate the block-values, which leads to slow convergence rates.



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

Figure 19: Random parallel deconvolutions on the LMC N132D supernova remnant.

The Figure 19 shows the behaviour on the LMC observation. The reconstructions receive obvious artifacts from the random selection strategy. The blocks, which get selected in the first few iterations, keep their over-estimated values. The parallel algorithm needs to select them several times to reduce their value. That is why even after 80k iterations, the N132D supernova remnant gets only hinted at in Figure 19b. Until the over-estimated blocks get selected again, the algorithm cannot do useful updates in that region.

This behavior is pronounced when we choose a block size of one pixel (i.e. we do not group pixels into blocks). A naive solution is to increase the block size. This leads to fewer possible blocks in the image, and obviously an increased chance to select the same block again in later iterations. But as we see in Figure 19d, the same problem exists with larger block sizes, although less pronounced. After 80k iterations the N132D supernova remnant is visible, but a few random blocks still contain too much of the emission in that area.

A random selection strategy needs a prohibitive large number of iterations to converge. But we cannot simply switch out the selection strategy. The random selection strategy is at the core of the Parallel coordinate descent methods. Remember the ESO arises from the fact that we select tau pixels uniformly at random. When we select τ-pixels with a greedy strategy, we might break the ESO, and the parallel algorithm may not converge at all.

To solve this behavior, we introduce the pseudo-random selection strategy: We select a block at random, but greedily search in the neighborhood for the optimal block to optimize. The size of the neighborhood can be defined by the user. It is essentially a mix between a greedy and a random selection strategy. If we choose the neighborhood to be the whole image, we arrive at a greedy strategy. If we choose the neighborhood to be just one block, we are back at a random strategy. The mixture of the greedy and random strategy allows us to fix the problems with the pure random strategy, without breaking any assumptions from the Parallel Coordinate Descent Method's ESO. The mixture of the greedy and random strategy is represented in a 'Search Fraction' parameter. It is a tuning parameter of our parallel coordinate descent algorithm. The optimal value for the Search Fraction is explored later in Section 7.4.3. We now introduce three related extensions, which speed up the parallel coordinate descent in practice: An active set heuristic, Restarting heuristic and a 'Minor' cycle.

6.3.1 Active set heuristic

The active set heuristic is typically used in cyclic coordinate descent: It chooses a subset of blocks, and optimizes the set until it converges. Then it chooses a new set. We use the active set heuristic together with our pseudo-random selection strategy. A large portion of the blocks in the image will be zero. If we select blocks at pseudo-random, we are likely to select a block that will never contain non-zero values and we wasted computing resources by trying to update this block. The active set heuristic increases the likelihood that the pseudo-random strategy selects a relevant block.

At the start of the parallel deconvolution algorithm, we initialize the active set by iterating over all blocks. We add all blocks which contain non-zero pixels, or pixels which may become non-zero by a serial coordinate descent iteration. Now during asynchronous parallel coordinate descent iterations, each processors only selects blocks from the active set. This increases the chance that each processor selects a block where pixel values can actually be modified.

Note that the algorithm only adds blocks to the active set, which can be changed to a non-zero value at the start of deconvolution. Over several iterations, there may be blocks that are not in the active set, but are part of the optimal solution. This is remedied with a restarting heuristic.

6.3.2 Restarting heuristic

In accelerated gradient methods like APPROX or (F)ISTA, restarting the acceleration can lead to a significant speedup[27]. In our accelerated variant, we use the current reconstructed image as the starting point, and reset the 'correction' gradient map and image x to zero. The question is, at what point is it useful to restart our parallel coordinate descent algorithm?

We implemented two restarting strategies: One strategy is based on Glasmachers et al.[28] and restarts the algorithm when the acceleration likely benefits from it. The other heuristic was developed by us and restarts the when the active set is likely to be missing blocks. There may be non-zero blocks in the image, which were not included when we initialized the active set. Our strategy estimates when the active set is likely missing relevant blocks, and restarts the algorithm with a new active set. In our tests, we always needed to restart due to the active set, and never due to the heuristic by Glasmachers. This is why we focus on our own developed restarting strategy, and ignore Glasmachers' in the pseudo-code.

Our own restarting heuristic is based on the following idea: We compare the maximum pixel difference after a number of asynchronous, parallel updates, to the difference a single step of serial block coordinate descent would produce. When the active set contains all relevant blocks, the parallel deconvolutions converge at a similar rate as the serial block coordinate descent. If the active set is missing important blocks, then the updates of the parallel coordinate descent start to converge, while the serial block coordinate descent update stays similar.

We extend the asynchronous parallel coordinate descent implementation with the active set and restarting heuristic:

```
1  ...
2  do
3    lastMaxDiff = GetGreedyMaxBlockDiff(gradientsMapExplore, xExplore)
4
5    parallelDiffFactor = 0
6    for activeSetIteration in activeSetIterations
7      parallelDiffs = new Array[]
8      ...
9      //asynchronous iterations
10     ...
11
```

```
    maxParallelDiff = Max(parallelDiffs)

    //restarting heuristic
    if(parallelDiffFactor = 0)
      parallelDiffFactor = lastMaxDiff / maxParallelDiff

    currentMaxDiff = GetGreedyMaxBlockDiff(gradientsMapExplore, xExplore)
    activeSetInvalid = lastAbsMax / maxParallelDiff > parallelDiffFactor * 2
    activeSetInvalid = activeSetInvalid | currentMaxDiff > lastAbsMax & lastAbsMax /
        parallelDiffFactor > concurrentFactor
    if activeSetInvalid
      Restart()
      parallelDiffFactor = 0
    lastMaxDiff = currentMaxDiff
  ...
while lastMaxDiff < epsilon
```

In each 'active set iteration', we let the asynchronous processors deconvolve the image for a set number of iterations. For example: Each processor deconvolves 1000 blocks asynchronously. If we use $\tau = 8processors$, then this results in a single active set iteration consisting of 8000 asynchronous iterations. Remember that we are now using a pseudo-random strategy. After enough parallel iterations, we are practically guaranteed to have selected the same block as a single serial block coordinate descent algorithm, if it is contained in the active set.

After the first active set iteration, we save the factor of how much the parallel update how close the maximum parallel update is to the best greedy step. The ratio of maximum greedy update and maximum parallel update should stay similar over the course of the algorithm, if the active set is valid. If the active set invalid, if it is missing important blocks, the algorithm will encounter ever smaller values for $maxParallelDiff$, while $lastMaxDiff$ does not decrease significantly over the active set iterations. In that case, we restart the algorithm with a new active set.

In several tests, we also observed that the maximum greedy update may actually increase from one active set iteration to the next. This is possible when we were unlucky and did not select the maximum block within one of the many asynchronous iterations, or when the active set is invalid (the maximum block is not contained in the active set). In the latter case we want to restart the algorithm. This is why we added a more aggressive condition which flags the active set as invalid, if $currentMaxDiff$ increases over active set iterations.

6.3.3 Re-introduction of a 'Minor' cycles

As we will demonstrate in Section 7, the parallel coordinate descent deconvolution algorithm benefits significantly from our PSF approximation method. The drawback of our PSF approximation is that it needs more major cycles to converge. We re-introduce a similar minor cycle to the Clark CLEAN algorithm [13], and reduce the number of necessary major cycles.

The CLEAN algorithm developed by Clark also uses only a fraction of the PSF during CLEAN deconvolutions. After a number of iterations, the residuals of the Clark algorithm are inaccurate, and it resets the residuals with the full PSF. We use a similar idea: We run our parallel coordinate descent deconvolution algorithm and retrieve the intermediate solution. We then decide whether we reset the residuals using the full PSF (the 'minor' cycle), or we use the major cycle.

Resetting the residuals with the full PSF is done as follows:

```
residuals = iFFT(Gridding(visibilities))     //Major cycle
x = DeconvolveParallel(residuals, Cut(PSF))      //Deconvolve with approximate PSF
residuals_minor = residuals - iFFT(FFT(x) * FFT(PSF)) //Update with full PSF
```

We convolve the intermediate solution x with the full PSF in Fourier space, and subtract the result from the original residuals from the major cycle. This allows us to remove some of the errors which the PSF approximation introduces, and reduce the number of Major cycles.

The question that remains is when to use a Major cycle or a 'Minor' cycle to reset the residuals. Remember from Section 5, we introduced a heuristic based on the PSF side lobe: When we deconvolve using only a fraction of the full PSF, we leave side lobes in the residual image. In each major cycle, we can only run the deconvolution algorithm up to a certain point, before we include PSF side lobes in the reconstructed image. In Section 5 we created a path regularization, which estimates a λ_{cycle} for each Major cycle. We used the largest PSF side lobe (largest value not included in the PSF window around the center) to estimate the minimum regularization λ_{cycle} for each Major cycle.

Now with the addition of a 'Minor' cycle, we use the same path regularization twice: We have two minimum regularization parameters, λ_{cycle}^{minor} and λ_{cycle}^{major}. The parameter λ_{cycle}^{minor} decreases for each 'Minor' cycle, λ_{cycle}^{major} decreases for each Major cycle. We the 'Minor' cycle as long as λ_{cycle}^{minor} is larger than λ_{cycle}^{major}. Otherwise, we start a new major cycle. We use the largest PSF side lobe to estimate λ_{cycle}^{minor} (the same as for the λ_{cycle} before). For the second regularization parameter λ_{cycle}^{major}, we do not have a natural PSF side lobe left in our approximation method. We chose to use the PSF side lobe, which is outside the $\frac{1}{2}$ center window of the PSF to estimate λ_{cycle}^{major}.

In total, the major and 'Minor' cycle for our parallel coordinate descent algorithm is implemented as follows:

```
residualVis = visibilities
x = new Array[,]

for each cycle in Range(0, maxMajorCycles)
  residuals = iFFT(Gridding(residualVis))
  residualsMinor = residuas

  lambdaMajor = Estimate(residuals, PSF, 2)
  lambdaMajor = Max(lambdaMajor, lambda)
  lambdaMinor = 0

  do
    lambdaMinor = Estimate(residualsMinor, PSF, psfFraction)
    lambdaMinor = Max(lambdaMinor, lambdaMajor)

    x_current = DeconvolveParallel(residuals, Cut(PSF, psfFraction), lambdaMinor)
    x += x_current
    residuals_minor = residuals - iFFT(FFT(x) * FFT(PSF))
  while(lambdaMajor < lambdaMinor)

  modelVis = DeGridding(FFT(x))
  residualVis = visibilities - modelVis
```

In each 'Minor' cycle, we estimate the current λ_{cycle}^{minor} regularization. We start the parallel coordinate descent algorithm with an approximate PSF and the current λ_{cycle}^{minor} regularization parameter. When the parallel algorithm has finished, we use the full PSF to update the residuals. We restart the 'Minor' cycle if $\lambda_{cycle}^{major} < \lambda_{cycle}^{minor}$.

An aggressive PSF approximation leads to a sharp increase in the number of necessary Major cycles. With

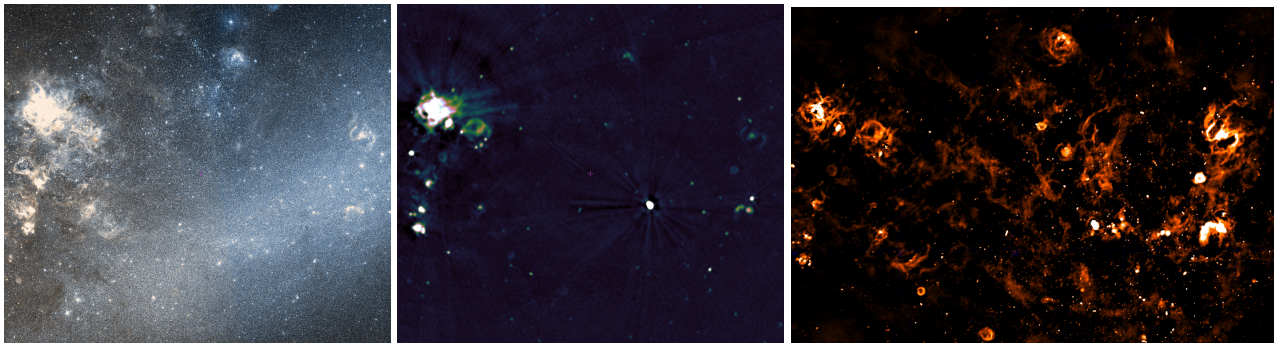
the re-introduction of 'Minor' cycles, we keep the total number of Major cycles comparable to other deconvolution algorithms like serial coordinate descent or CLEAN.

7 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 20 shows the LMC in both optical and radio wavelenghts. The radio wavelengths was observed by the VLA radio interferometer[29] 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 20b, 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 20c.



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

Figure 20: 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 21 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 21 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.

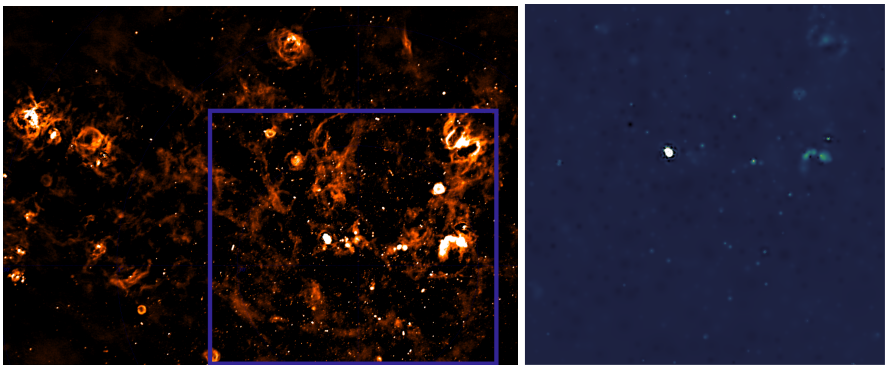


Figure 21: Narrow band image section used.

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 7.1 we compare the reconstructions of CLEAN with our serial coordinate descent algorithm on the LMC observation. The next Section 7.2 presents the speedup we achieve with serial coordinate descent by using our distributed or GPU-accelerated implementations.

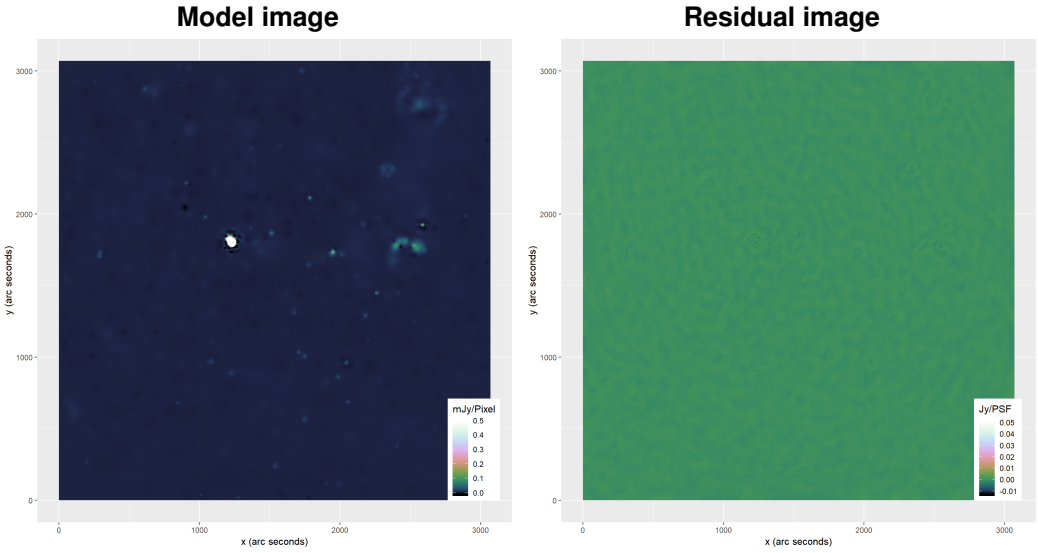
In Section 7.3 we show the core result of this project. Namely what effect has an approximate PSF on the deconvolution problem and whether we can use it to further distribute the problem. The answer to that question is affirmative: We can approximate the PSF, and we can exploit it to further distribute the deconvolution. But we need more sophisticated coordinate descent algorithms to fully benefit from it.

7.1 Comparison with state-of-the-art reconstruction algorithms

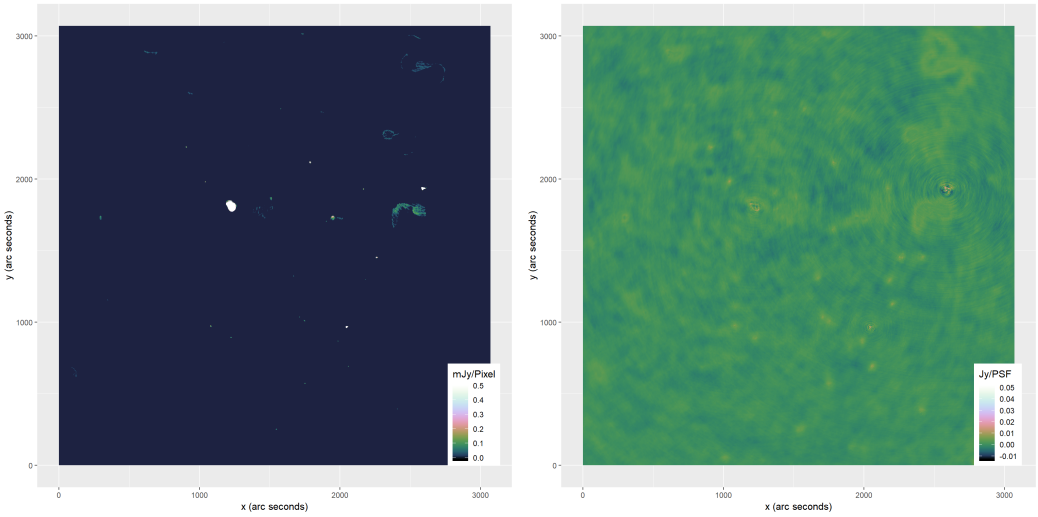
We compare our serial coordinate descent algorithm with the state-of-the-art algorithms multi-scale CLEAN and MORESANE on the LMC data set. We test against the WSCLEAN [30] implementation of multi-scale CLEAN and MORESANE (IUWT)[3], and compare the resulting model and restored images. The model image is the direct output of the deconvolution algorithm. The restored image is convolved with the 'clean-beam', a 2D Gaussian function representing the instrument's accuracy[4]. A reconstruction algorithm achieves super-resolution, if the model image it produces contains plausible structures.

We first compare the model and the residual images of the three algorithms. The reconstruction algorithm should detect all sources (we have a non-zero pixels for all point- and extended sources) and leave as little of the true emissions in the residuals. Meaning we should not see any remaining structures of the image 21 in the residuals. The Figure 22 shows the model- and residual image of multi-scale CLEAN, serial coordinate descent and MORESANE.

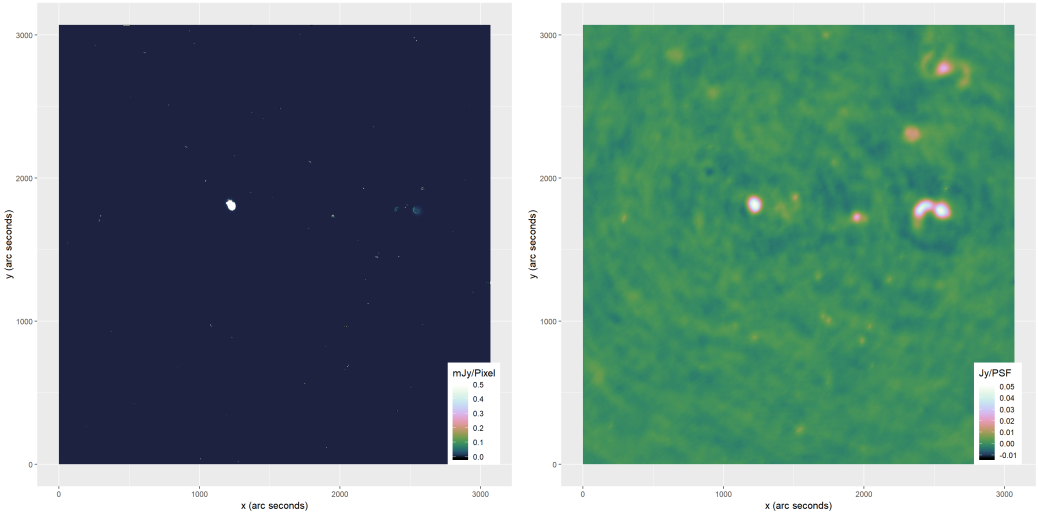
[3]The WSCLEAN package has a re-implementation of the original MORESANE algorithm[4], which is called IUWT.
[4]Usually, the residuals are added to the restored image. We compare the restored images without the added residuals.



(a) Multi-scale CLEAN.



(b) Coordinate descent with elastic net.



(c) MORESANE (IUWT).

Figure 22: Comparison of the whole image

Multi-scale CLEAN detects all extended emissions (we have non-zero pixels in the model for every source). It has the smallest residual magnitudes of the three algorithms, and has barely visible structures left in the residual image. Multi-scale CLEAN has reconstructed the image close to the noise level. The serial coordinate descent reconstruction does not contain any non-zero pixels in the model image, but its residual image contains still visible structures. The MORESANE reconstruction surprisingly has the largest magnitudes in the residual image. It still has clear structures belonging to the extended sources in the residuals. Also note that MORESANE has not detected the top-right extended emission. It is missing in the model image.

The WSCLEAN software package has implemented an auto-masking strategy for multi-scale CLEAN and MORESANE, which helps the deconvolution algorithm reconstruct an image close to the noise level. At a certain point in the deconvolution, the 'mask' forbids the algorithm to add new sources to the model image. It can only change existing sources to explain the emission left in the residual image. This helps the deconvolution algorithm to reconstruct closer to the noise level, since it is forbidden to add new structures likely to be due to noise.

Even with an auto-masking strategy, the MORESANE algorithm left notable emissions in the residual image. The serial coordinate descent algorithm with elastic net regularization does not use an auto-masking strategy. Nevertheless, its residual image has lower magnitudes than MORESANE. These results may be improved further by extending the serial coordinate descent algorithm with an auto-masking strategy.

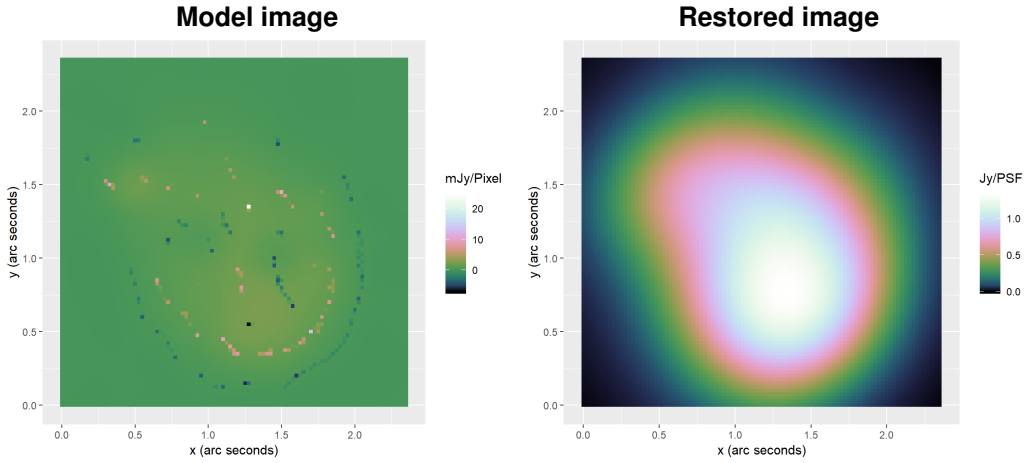
We now compare two extended emissions in the image in detail: We compare the N132D supernova-remnant, the bright radio source in the center of the image. We also compare the extended emission at the right-hand side of the image, since it contains significant calibration errors. First, we compare the three algorithms on the N132D supernova-remnant

7.1.1 Super-resolution of the N132D supernova-remnant

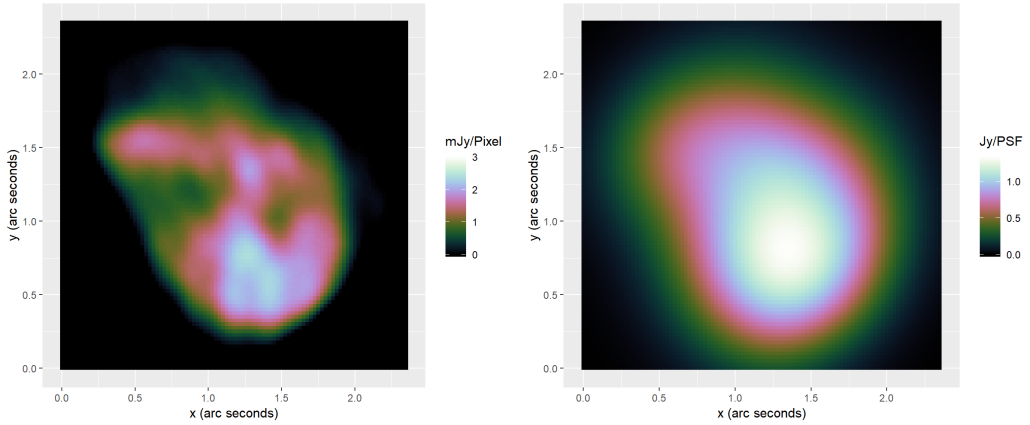
In Figure 23. We compare the model- and restored images of the three algorithms. The restored images are almost identical in structure for the three algorithms. The main difference is the resulting magnitude: The restored image of multi-scale CLEAN has the highest pixel magnitude, while the restored image of MORESANE leads to the lowest pixel magnitude. Remember: MORESANE has left significant emissions in the residual image, while multi-scale CLEAN has removed almost all the relevant emission. This leads to a difference in the magnitudes of the restored image.

The model images on the other hand have considerable differences. Multi-scale CLEAN chose to reconstruct the supernova-remnant with mostly single pixels, containing both significant positive and negative values. The large negative values forced us to use a different color axis for the model image. With multi-scale deconvolution, CLEAN has a model for extended emissions. But in this case, CLEAN has chosen to explain the emission mainly with single pixels. Multi-scale CLEAN has used negative pixels instead of reducing the magnitude of the pixels already in the model image. It has surrounded the supernova-remnant with negative pixels, and added a few negative pixels inside the remnant.

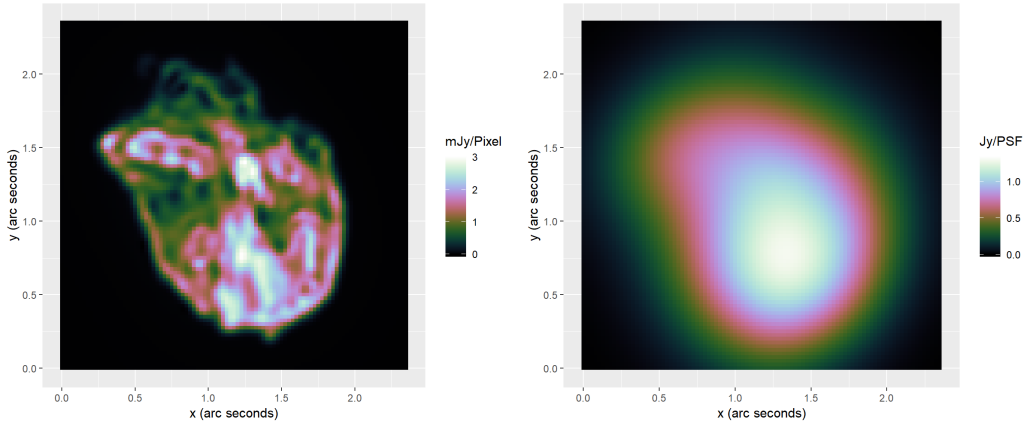
Coordinate descent with the elastic net regularization has reconstructed a smoother version of the multi-scale CLEAN model image. It does not contain any negative pixel values, and the resulting structure seems plausible. The MORESANE model image contains similar structures, with bright pixels at similar locations. But the MORESANE model image contains more details. Now the question is whether the retrieved structures by coordinate descent and MORESANE are plausible, and may have super-resolved the N132D supernova-remnant.



(a) Multi-scale CLEAN.



(b) Coordinate descent with elastic net.



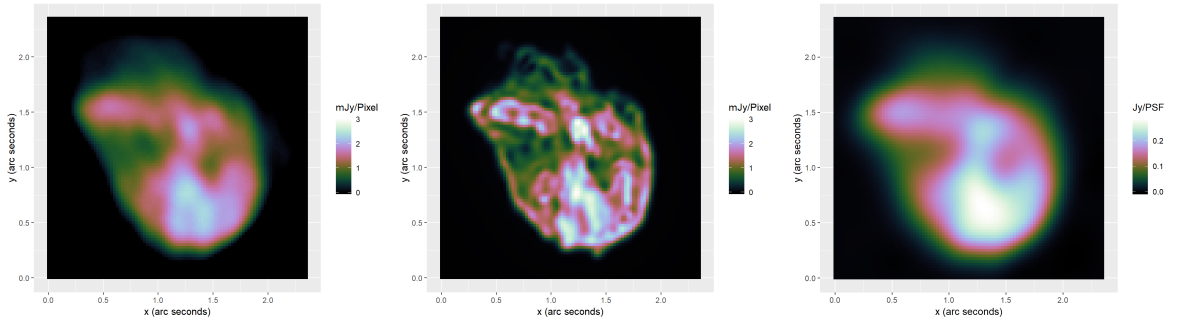
(c) MORESANE (IUWT).

Figure 23: Comparison on the N132D supernova-remnant.

To ensure the structures of the coordinate descent and MORESANE model images are plausible, we compare them to the restored image of briggs-weighted multi-scale CLEAN on the same data. There are three main visibility weighting scheme for the gridder that lead to different PSFs from the same measurements: Natural, uniform, and Briggs[31]. 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 which 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.

We compare the coordinate descent and MORESANE reconstruction with the briggs-weighted multi-scale CLEAN restored image in Figure 24. Here, we compare if the structures of the MORESANE and coordinate descent model images are plausible, and therefore a super-resolved reconstruction of the N132D supernova-remnant.



(a) CD *(b) IUWT* *(c) Briggs MS-CLEAN*

Figure 24: Comparison with briggs-weighted multi-scale CLEAN as Ground Truth.

The model image of the coordinate descent reconstruction closely resembles the structures in the briggs-weighted multi-scale CLEAN restored image. Coordinate descent reconstructed a bright core and a distinct 'horizontal' arm, similar to briggs-weighted CLEAN. The MORESANE (IUWT) model image contains roughly similar structures, but adds more details. In this work, we cannot verify the more detailed structures of the MORESANE model image of the N132D supernova-remnant.

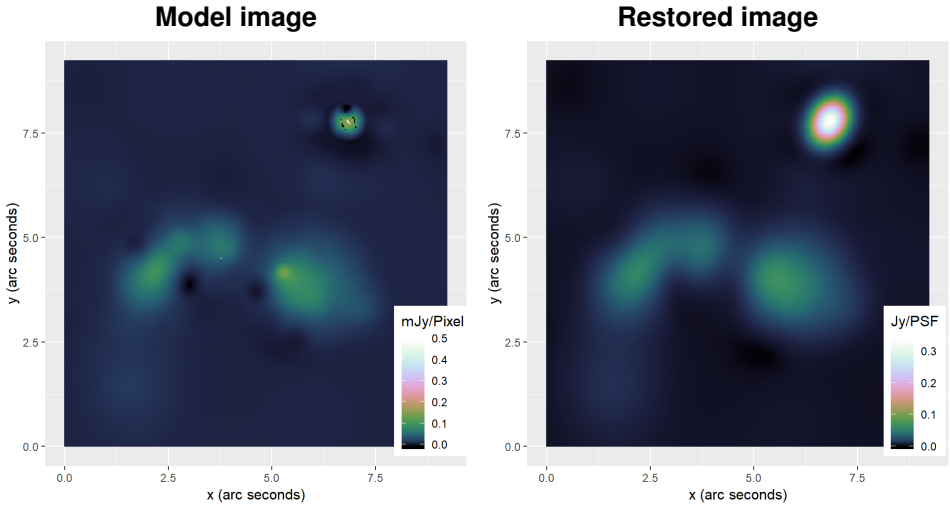
7.1.2 Influence of calibration errors

In this section, we compare the model and restored images of an area influenced by calibration errors. The image section is located on the center-right of the full image. We see two faint extended emissions, next to a point source with calibration errors. The phase calibration in this area of the image is imperfect, which leads to a de-correlation artifacts (wave-like artifacts around the point source). The de-correlation artifacts are mainly visible in the residual images. But they also influence the model- and restored images. Figure 26 compares the model and restored images of the three algorithms.

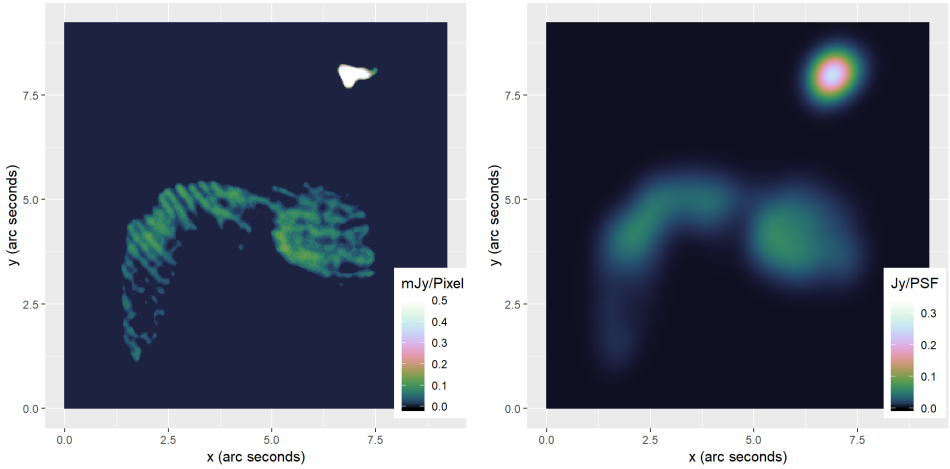
The multi-scale CLEAN and MORESANE model images contain significant negative pixel values in this section of the reconstruction. For a reasonable comparison, we had to clip the negative values, which are now shown as black pixels in the model images. Due to the bright point source in the image, we also clipped the maximum positive value for all model images.

The model image of Multi-scale CLEAN contains a straight-forward representation of the two extended emissions. Significant negative pixels are placed around the extended emissions. The point source contains several 'rings' of negative and positive pixels, related to the calibration errors. The serial coordinate descent algorithm is forbidden to use negative pixels. The calibration errors result in a 'smearing' of the point source in the horizontal directions. The two extended emissions also get influenced by the calibration errors with coordinate descent. MORESANE on the other hand misses large parts of the left-hand-side extended emission. The point source in the image is reconstructed similar to multi-scale CLEAN, containing rings of significant negative pixels.

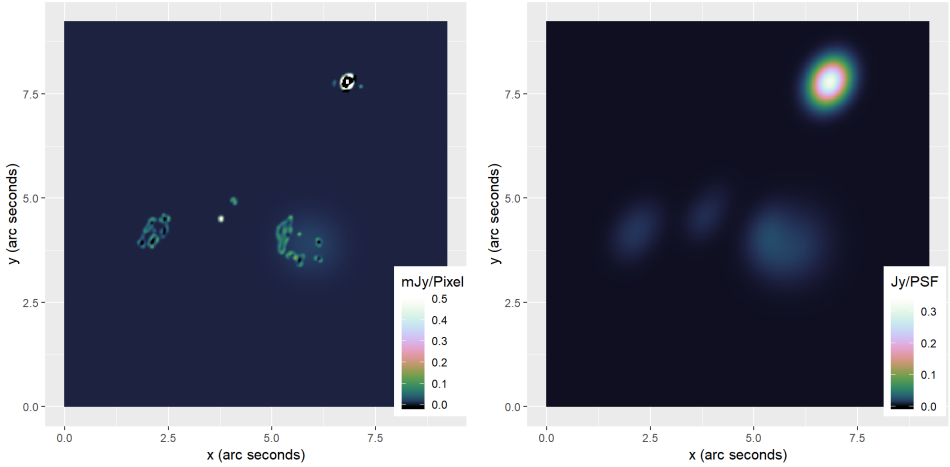
These differences in the model images also result in variations of the restored image. The multi-scale CLEAN restored image contains several areas of negative emission. The coordinate descent restored image does not contain any negative pixels. The two extended emissions are also similar in shape and pixel value to multi-scale CLEAN. The restored image of MORESANE is missing parts of the extended emission.



(a) Multi-scale CLEAN



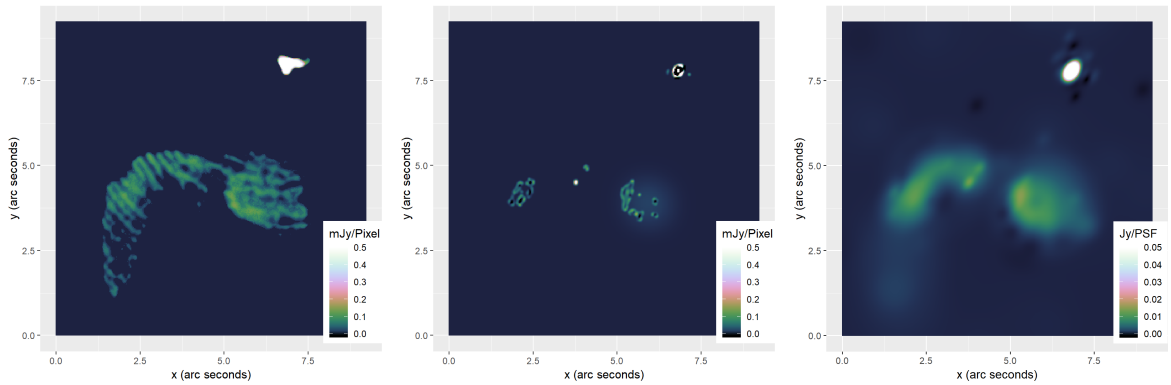
(b) Coordinate Descent with elastic net.



(c) MORESANE (IUWT)

Figure 25: Influence of calibration errors

Lastly, we again compare the model images of coordinate descent and MORESANE (IUWT) with the restored image of briggs-weighted multi-scale CLEAN. The briggs-weighted restored image has reconstructed more details in the two extended emissions. The left-hand-side emission contains two point sources at its border,



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

Figure 26: Comparison to briggs-weighted multi-scale CLEAN as Ground Truth.

while the right-hand-side emission contains a crescent. The model image of the coordinate descent algorithm has detected structures of the crescent, but has missed the two point sources. MORESANE did detected the point sources and the crescent in the extended emission, but did miss large parts of the actual emission.

Overall, serial coordinate descent with elastic net regularization has produced a restored image with similar structures to multi-scale CLEAN on the same observation. Coordinate descent has left a residual image with a higher pixel magnitudes than CLEAN. This can be improved by adding a similar auto-masking strategy to serial coordinate descent. The model images resulting from the elastic net regularization is more plausible than multi-scale CLEAN on this observation. It has produced a plausible super-resolution of the N132D supernova-remnant, and has found other plausible structures in extended emissions. However, its model image also seems to be more susceptible to calibration errors.

The MORESANE algorithm was reported to generally produce higher-quality reconstructions than multi-scale CLEAN, while also needing more computing resources [4, 7]. In our test, multi-scale CLEAN has produced a higher quality reconstruction. MORESANE has various tuning parameters which influence the reconstruction quality. It is possible that we used sub-optimal parameters for the reconstruction. We modified several parameters of the MORESANE algorithm, but could not improve the results reported here.

The WSCLEAN software package has an option for multi-scale CLEAN, which forces the model image to be non-negative. However, we have not compared those results to the serial coordinate descent reconstruction. This option has lead multi-scale CLEAN to an overall worse reconstruction, with higher pixel magnitudes in the residual image, and longer wall-clock time. With our tuning parameter setting, it required more than twice the number of Major cycles than previous (14 Major cycles compared to 6).

The serial coordinate descent algorithm needed more computing resources than multi-scale CLEAN. The main difference is in the number of Minor cycles: Multi-scale CLEAN needed overall 15'000 iterations, while serial coordinate descent needed 100'000 iterations. The number of Major cycles was similar for both algorithms: Multi-scale CLEAN converged within 6 Major cycles, and serial coordinate descent within 5. Note that the number of Major cycles of the two algorithms is only roughly comparable. In this project, we use a different gridder than WSCLEAN, and we have not ensured our stopping criterion is identical to WSCLEAN.

We now test how we can speed-up first serial coordinate descent, and then parallel coordinate descent to beat multi-scale CLEAN in terms of reconstruction time.

7.2 Serial coordinate descent speedup with MPI or GPU

In this section we test how much we can speed-up our serial coordinate descent algorithm by using distributed computing with MPI, or GPU acceleration.

We test the distributed serial coordinate descent on a shared memory system(Meaning all CPUs have access to the same main memory) with 32 CPUs. This is a best-case scenario for our implementation, because each node runs on the same physical machine: Our implementation needs a communication step between the nodes for each serial coordinate descent iteration. It needs a low-latency connection between each node to run as efficient as possible. Since each node runs on the same physical machine, it has a low latency times and therefore a low communication time. In this implementation, each node uses a single processor. We compare the speedup the algorithm achieves by adding more nodes/processors.

For the GPU we used personal computer level hardware, and tested on a nVidia Quadro M1200. We compare the speedup we achieve to the on-board CPU, which is an Intel Xeon E3-1505M with 8 logical processors at different image sizes. The speedup is shown in Figure 27.

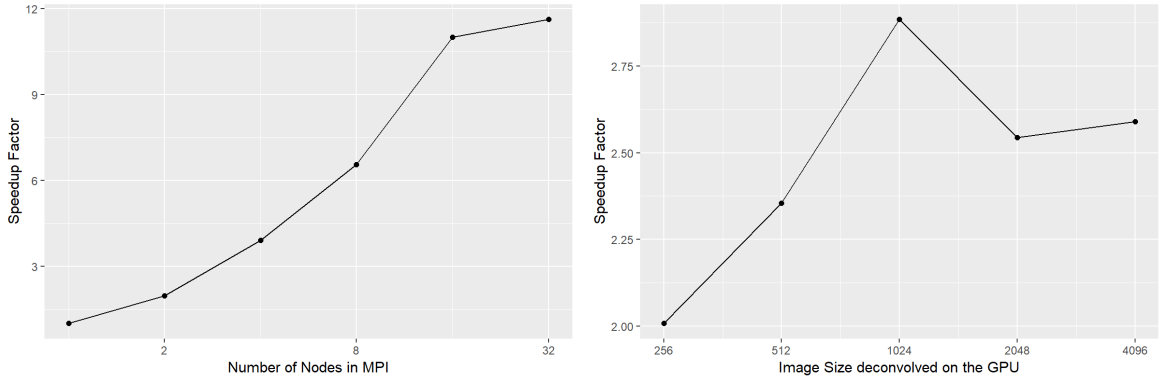


Figure 27: Speedup by using MPI or GPU acceleration

As we mentioned before, the speedup we achieve by using nodes/processors in MPI is the best-case scenario. As the best case scenario, we achieve a significant performance increase by using more nodes/processors. Up to 16 processors, the speedup we achieve is at least linear. Afterwards however the speedup diminishes. With 32 processors, we are only marginally faster than with 16. If the nodes would run two different physical machines, the speedup we achieve would depend mainly on the latency of the connection.

The speedup we achieve by using GPU acceleration is fairly constant over the image size. The speedup factor varies around 2.5. Ideally, we would like to combine the distributed and the GPU implementation. However, this is not useful with the current implementation: The main bottleneck in the MPI implementation is the communication step in each iteration.

We do not have a CLEAN implementation in our .Net Core pipeline. However, we mentioned the similarities between the serial coordinate descent and CLEAN in Section 4.7. A single serial coordinate descent iteration is roughly equivalent to a standard CLEAN iteration. If we assume we need 14'000 CLEAN iterations to reconstruct an image, we can get a rough estimate by comparing the wall-clock time of 14'000 serial coordinate descent iterations. In this case, CLEAN is roughly 7 times faster than the current serial coordinate descent iteration.

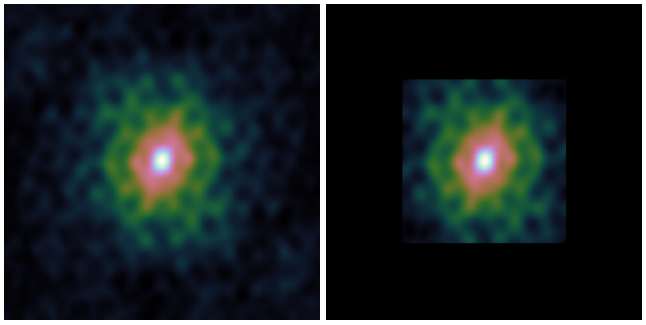
Overall the serial coordinate descent algorithm with GPU acceleration is still several factors slower than CLEAN. The speedup we achieve with MPI is significant. But because serial coordinate descent and CLEAN have such a similar structure, we can expect a similar speedup with a CLEAN-MPI implementation. We need another method to speed up our deconvolution algorithm.

7.3 PSF approximation with serial coordinate descent

In this section, we test our main hypothesis on the MeerKAT LMC data: The Major cycle allows us to use an approximation of the true PSF in Minor cycle. We believe this can be exploited. We may be able to use a

fraction of the true PSF in the serial coordinate descent algorithm, potentially speeding up the deconvolution and simplifying distribution.

An example of the PSF approximation is shown Figure 28. Instead of the full PSF, we use a window around the center of the PSF. The sides of the window are a fraction of the full PSF.



(a) Full PSF *(b) $\frac{1}{2}PSF$ approximation*

Figure 28: Example of a PSF approximation.

In this section, we test ever smaller window sizes and observe the convergence speed of our serial coordinate descent algorithm. We developed two methods to exploit an approximate PSF in our serial coordinate descent algorithm: Method 1, approximate update and method 2, approximate deconvolution. At the start of each Major cycle, we calculate the objective value of the current solution, and track its development for each Major cycle. Finally, we combine the two approximation methods and compare how much we can speed up our serial coordinate descent method.

7.3.1 Method 1: Approximate update

As described in Section 5.1.1, we normally use the product of $PSF \star PSF$ to update the gradient map in each iteration of serial coordinate descent. This method uses the approximate PSF to update the gradient map: $Cut(PSF) \star Cut(PSF)$. With each iteration, the gradient map becomes less accurate. This approximation method is guaranteed to converge to the same result, if we have unlimited number of major cycles.

To test the convergence behaviour, we calculate the objective value of the current solution at the start of each Major cycle. We compare the objective value and the wall-clock time of the original serial coordinate descent and the serial coordinate descent using an approximate gradient update. This is a minimization problem, meaning the lowest objective is the most accurate reconstruction (according to the elastic net regularization).

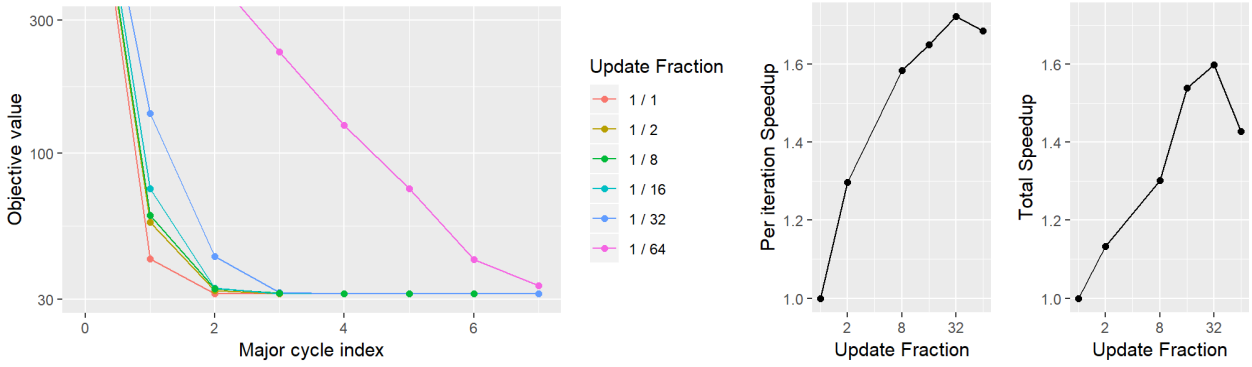


Figure 29: Convergence of the approximate update method.

Figure 29 shows the convergence rate of the serial coordinate descent algorithm with the approximate update

method. Each line represents a run with a specific PSF window. The size of the window are a fraction of the full PSF. For example, the full PSF in this test has 3072^2 pixels. The approximation $\frac{1}{64}$ uses a window around the center of the PSF, which is is 48^2 pixels in size. The $\frac{1}{64}$ approximation is uses the smallest window around the PSF for which the serial coordinate descent algorithm still converges. If we use less aggressive approximations, we see the serial coordinate descent algorithm reach a similar objective value within three Major cycles.

We compare two different wall-clock times: The average time necessary for a single serial coordinate descent iteration, and the total time spent in the algorithm. We see from the two speedup curves in Figure 29 that the speedup per iteration increases less than linear with the PSF fraction used in the approximation. Although each iteration becomes cheaper with a smaller PSF window, we may need more iterations to converge to the same result. This is indeed the case for a PSF window smaller than $\frac{1}{32}$. We see a significant drop total speedup. Overall with gradient update approximations give us a speedup factor of 1.6, if we use $\frac{1}{32}$ of the full PSF.

7.3.2 Method 2: Approximate deconvolution

As described in Section 5.1.2, this approximation method uses the same approximation to update the gradient map($Cut(PSF) \star Cut(PSF)$). This approximation method also uses the approximate PSF to initialize the gradient map: $gradients = residuals \star Cut(PSF)$. As such, this approximation method deconvolves the image not with the full PSF, but with the center window. This method is not guaranteed to converge to the same result. Nevertheless, we run our serial coordinate descent algorithm on different PSF fractions and compare the true objective value at the start of each Major cycle. The results are shown in Figure 30.

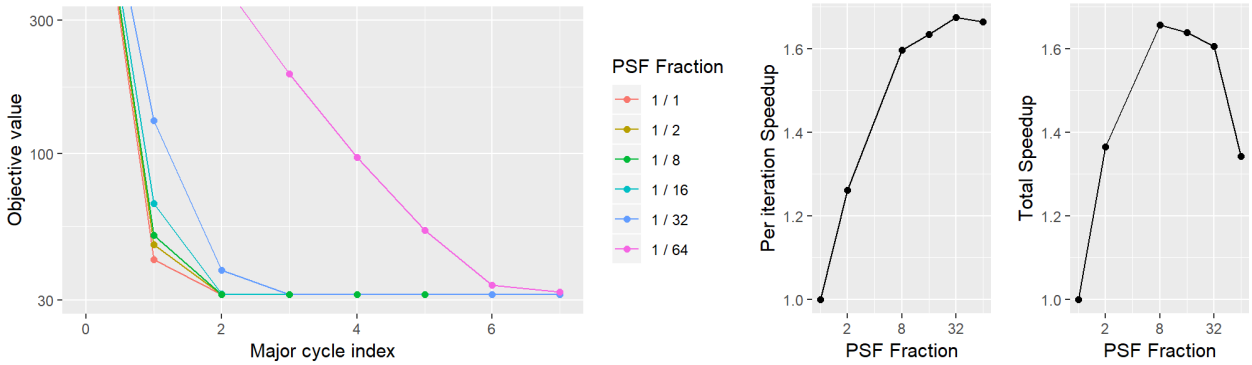


Figure 30: Convergence of the approximate deconvolution method.

Again, the smallest window for which the serial coordinate descent algorithm converges is $\frac{1}{64}$. Similar to method 1, the serial coordinate descent algorithm reaches a similar objective value within 3 Major cycles, for all PSF windows except $\frac{1}{64}$. The total speedup however is notably different: It reaches a larger total speedup. The total speedup curve generally matches better with the 'per iteration speedup' curve. This suggests that the approximate deconvolution method does not require significantly more iterations to converge, even when we use a PSF window of $\frac{1}{16}$.

The approximate deconvolution method is not guaranteed to lead to the same result. Nevertheless, the true objective value of the results are practically identical in Figure 30. However, as we mentioned before, this approximation method systematically under-estimates the true pixel values. The pixel magnitude is important in the self-calibration regime [7]. Self-calibration uses intermediate results of the image reconstruction to improve the calibration. A systematic under-estimation of the pixel values leads to a bias in self-calibration.

7.3.3 Combination of Method 1 and 2

The approximation method 2, approximate deconvolution, leads to a systematic under-estimation of the true pixel values. This is undesirable for a reconstruction algorithm. The obvious question is, how does the serial coordinate descent algorithm perform when we combine the two approximation methods: We start with method 2. When the algorithm converged, we switch to method 1 and correct the systematic error introduced by method 2. We compare the speedup we achieve by combining the two approximation methods in Table 1.

Method	PSF Fraction	Iteration Speedup	Total Speedup
Original	$\frac{1}{16}$	1.00	1.00
(Method 1) Approx. gradient update	$\frac{1}{16}$	1.66	1.55
(Method 2) Approx. deconvolution	$\frac{1}{16}$	1.67	1.68
Combined	$\frac{1}{16}$	1.69	1.68

Table 1: Results when combining the two approximation methods

Combining the two approximation methods leads to a total speedup with a PSF window of $\frac{1}{16}$. The combined approximation method achieves a larger speedup than method 1 or method 2 on the same PSF window size. We suspect the error each approximation method introduces is uncorrelated. Meaning the error we introduce with approximation method 2 can be efficiently corrected with approximation method 1, which leads to a higher total speedup. In total, the serial coordinate descent algorithm with a combined PSF approximation method converged within 1486 seconds, or 24 minutes.

Although the serial coordinate descent algorithm can be sped-up with the PSF approximation, it does not lead to an algorithm with a comparable wall-clock time to CLEAN or multi-scale CLEAN. The speedup it achieves per iteration is limited by synchronization overhead: The serial coordinate descent algorithm uses all available processors to update the gradient map. With our approximation method, the serial algorithm can update an increasingly smaller facet of the gradient map. If we use a PSF window which is $\frac{1}{16}$ of the total 3072^2 size, the serial coordinate descent updates 384^2 elements of the gradient map in each iteration. Utilizing all processors effectively with ever smaller PSFs becomes difficult.

7.4 PSF approximation with parallel coordinate descent

T

We have

Re-introduce parallel

Re-introduce the ESO.

Various tuning parameters.

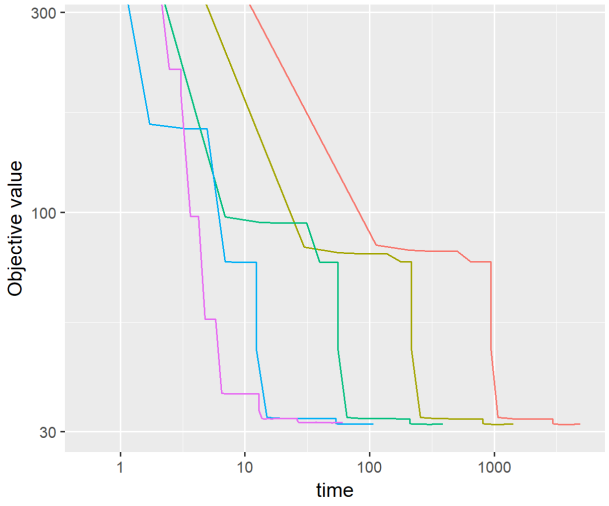
We have shown the parallel coordinate descent algorithm in two variants: With gradient acceleration and without. We developed several heuristics, which includes a set of tuning parameters. In this section, we test our parallel coordinate descent algorithm on the MeerKAT LMC observation. We test the tuning parameters, and show how much our algorithm benefits from the PSF approximation method.

In this test, we measure the wall-clock time of only the deconvolution. The wall-clock time of the Major- and 'Minor' cycle are excluded. Again, we calculate the value of the objective function. We compare different parameters of our parallel coordinate descent algorithm, trying to get the shortest convergence time.

7.4.1 PSF approximation

First, we test the effect of our PSF approximation method. We test different PSF cutouts of our parallel coordinate descent algorithm with gradient acceleration. We use $\tau = 8$ processors with our asynchronous implementation.

Figure 31 shows the result of our parallel coordinate descent algorithm with different PSF fractions, combined with the ESO and the total seconds needed to converge. The y-axis shows the objective value, and the x-axis shows the elapsed seconds. Note that the axis of the figure are logarithmic. The 'kinks' in the lines are due to either the Major or the 'Minor' cycle resetting the residuals (remember: we do not include the wall-clock time of the Major or 'Minor' cycle in this test).



PSF	ESO	Total seconds
1 / 4	2.750	4941
1 / 8	1.437	1433
1 / 16	1.109	390
1 / 32	1.027	108
1 / 64	1.001	61

Figure 31: Convergence times with PSF approximation

Clearly, the parallel coordinate descent algorithm benefits from our PSF approximation method. If we use $\frac{1}{32}$ of the full PSF, the parallel coordinate descent algorithm spends a total of 108 seconds in to deconvolve the image. For comparison, the serial coordinate descent algorithm with PSF approximation takes roughly 1400 seconds, or 23 minutes.

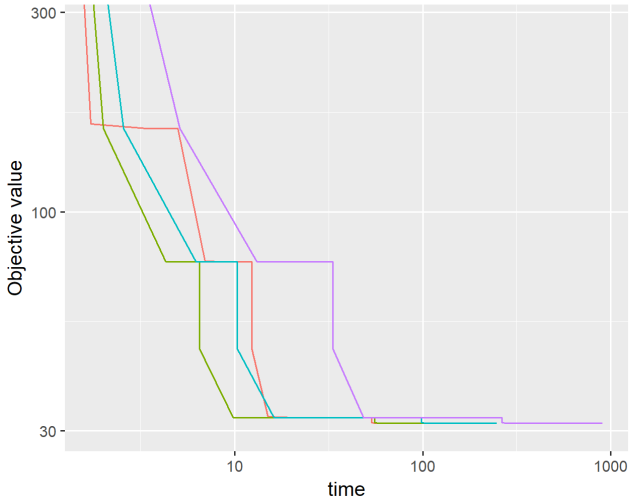
As expected, with increasing PSF approximation, we get an ESO which is ever closer to 1. Remember: An ESO of 1 means that each processor in the parallel coordinate descent algorithm can has the same step size as the serial coordinate descent algorithm. With an approximation of $\frac{1}{32}$ and 8 parallel processors, the ESO is only marginally larger than 1. This suggests that the parallel coordinate descent algorithm may be sped up further with additional processors.

Note that the speedup from fraction $\frac{1}{4}$ to $\frac{1}{8}$ is roughly a factor of 3.5. The same holds true for the speedup of $\frac{1}{8}$ to $\frac{1}{16}$, and $\frac{1}{16}$ to $\frac{1}{32}$. The PSF cutout from $\frac{1}{8}$ to $\frac{1}{16}$ four times fewer pixels. This suggest the speedup may be due to the reduced conflicts in the asynchronous update of the gradient map. With a smaller PSF we reduce the change of several threads updating the same location in the gradient map, and we spend more time in the deconvolution itself.

For the rest of this project, we will use a PSF approximation of $\frac{1}{32}$ for the parallel coordinate descent algorithm. The $\frac{1}{64}$ approximation is faster in this tests, but needs more 'Minor' cycles (Which were excluded from the wall-clock time). Including the time spent in the 'Minor' cycle, the $\frac{1}{32}$ approximation is the fastest overall.

7.4.2 Block size

The parallel coordinate descent algorithm can group several pixels in blocks, and optimize the blocks of pixels in parallel. It is not clear if grouping the pixels in blocks results in shorter convergence times. We test different block sizes in Figure 32, all using the same PSF approximation of $\frac{1}{32}$. We tested out block size of 1^2 (every block contains a single pixel) up to a block size of 8^2:



Block Size	Total seconds
1^2	108
2^2	103
4^2	248
8^2	905

Figure 32: Convergence times with different block sizes

The larger block sizes of 4^2 and 8^2 are significantly slower to converge than a block size of 1^2. Only a block size of 2^2 is slightly faster. Interestingly though, the parallel coordinate descent algorithm is faster to arrive at intermediate results with the block sizes 2^2 and 4^2. This suggests that the parallel coordinate descent algorithm may benefit starting out from larger block sizes, and gradually reducing the block size over several major cycle iterations.

However, two factors lead to the decision to simply use a block size of 1^2 for all major cycle iterations: First, the block size is likely connected to the image resolution. An effective heuristic that starts with larger blocks may become difficult to develop for general observations. Secondly, the parallel coordinate descent implementation becomes more complicated when it has to account for different block sizes. An implementation with a block size of only 1^2 is shorter and simpler.

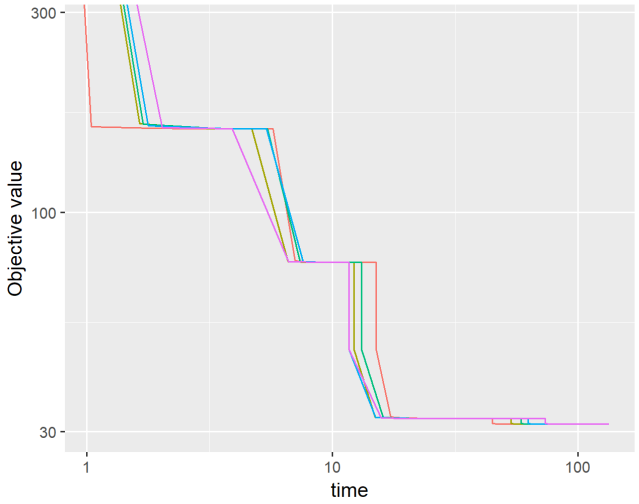
For the rest of this project, we use a block size of 1, meaning every thread is minimizing a single pixel.

7.4.3 Pseudo-random strategy and the Search Fraction

We discussed in Section 6.3, a random block selection strategy seems to perform badly on the deconvolution problem. We created the pseudo-random strategy, which selects a block at random, but searches in it's neighborhood for the optimal block to optimize. It is a mixture between a random and a greedy selection strategy.

But the pseudo-random strategy introduces a new tuning parameter, which we call the 'search factor'. A search factor of 0 says that after a random block has been selected, we search 0% of the neighboring blocks in the active set. It is identical to a pure random selection strategy. A search factor of 1.0 says that after a random block has been selected, we search 100% of its neighbors in the active set. For example, if we have 256 blocks with 8 threads, a search percentage of 1.0 selects a random block and looks at 32 blocks in its neighborhood. A search percentage of 100% is identical to a greedy strategy, where each thread searches its part of the active set.

Our parallel coordinate descent algorithm needs a search percentage larger than 0. We compare different search percentages in Figure 33.



Search Factor	Total seconds
0.01	110
0.1	106
0.2	104
0.4	112
0.8	134

Figure 33: Convergence times with different search percentages.

According to this test, the search factor tuning parameter influences the convergence speed, but tuning it does not lead to a significant speedup. A value between 0.01 and 0.4 leads to a similar wall-clock time. It is only when the parameter is set close to 1 (close to fully greedy) where we see a significant increase in wall-clock time.

Based on this test, we fixed the search percentage tuning parameter to a value of 0.1 for all tests in this project. It was enough to alleviate the problems a pure random strategy introduces in the deconvolution problem (discussed previously in Section 6.3).

However, we do not know whether this parameter generally has a small influence on convergence speed, or if it is merely due to the LMC observation we used for this project. To answer this question, we need further tests on diverse observations.

7.4.4 Acceleration

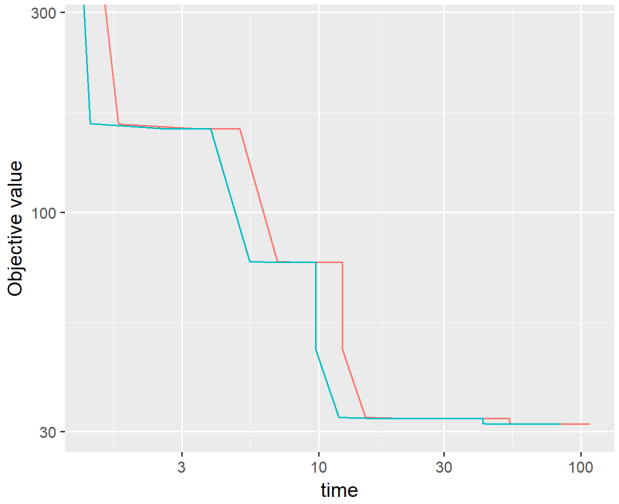
Lastly, we test whether the parallel coordinate descent algorithm is faster with or without gradient acceleration. The accelerated variant needs two versions of the gradient map, which get updated asynchronously with compare-exchange operations. Figure 34 compares the accelerated and non-accelerated parallel coordinate descent algorithm.

The accelerated variant is significantly slower in every part of the algorithm. Our hypothesis is that the two gradient maps necessary for the acceleration also increase the cost of synchronization.

Furthermore the accelerated variant has additional run time costs which were not measured in this test: It has to create a copy of the gradient map and the reconstructed image for the acceleration. Meaning this test shown in Figure 34 is biased in favor of the accelerated variant, yet it is still slower.

7.5 Comparison to the serial coordinate descent algorithm

In the previous section, we tested various tuning parameters of the parallel coordinate descent algorithm. In this section, we compare the serial coordinate descent algorithm with the final, simplified parallel coordinate descent algorithm.



Method	Total seconds
With Acceleration	108
Without Acceleration	84

Figure 34: Convergence time with or without gradient acceleration.

We took the lessons from the tests and implemented a simplified parallel coordinate descent algorithm. It does not use gradient acceleration, and cannot group pixels into blocks. Each thread can only minimize a single pixel in parallel in each iteration. As we will see shortly, the simplified parallel coordinate descent implementation is significantly faster than the previous implementation.

In this test, we compare the wall-clock time spent on the deconvolution algorithm. The time we measure here does not include the Major cycle. For the parallel coordinate descent algorithm, this means we also measure the time spent in the 'Minor' cycle for the parallel coordinate descent algorithm (which was excluded in the previous tests).

We use the same hardware as in the tests of Section 20. The table 2 shows the comparison between the serial and parallel coordinate descent algorithm. The parallel coordinate descent algorithm achieved a speedup factor of roughly 20. While the serial coordinate descent algorithm takes over 20 minutes to deconvolve the image, the parallel coordinate descent algorithm takes less than two minutes in total.

Algorithm	PSF	Major Cycles	Total Seconds in Deconvolution	Speedup factor
Serial CD	$\frac{1}{16}$	4	1486	–
Parallel CD	$\frac{1}{32}$	5	75	≈ 20

Table 2: Speedup comparison of the serial and parallel coordinate descent algorithm. Both algorithms were compared on an Intel Xeon E3-1505M with 8 logical cores.

This parallel coordinate descent implementation is even faster than the implementation tested in the previous Section 7.4. This implementation is simpler because it does not account for different block sizes. Each processor deconvolves a single pixel in parallel. The simplification leads to another decrease in wall-clock time.

The total deconvolution time of the parallel coordinate descent algorithm is in the range of the CLEAN algorithm. An exact comparison was not possible in this project. We did not implement the CLEAN algorithm in our .Net Core pipeline. However, the serial coordinate descent algorithm has an almost identical structure to the standard CLEAN algorithm, and we use this fact to get a rough comparison to CLEAN: The multi-scale CLEAN algorithm required 15'000 iterations to converge on the LMC observation. 15'000 serial coordinate descent iterations on the same hardware take approximately 230 seconds, which is more than the parallel coordinate descent algorithm needed.

The serial coordinate descent algorithm is significantly slower than the parallel algorithm. However, it requires

one major cycle less than the parallel algorithm to arrive at a similar reconstruction. The Figure 35 compares the reconstruction of the two algorithms of the LMC observation.

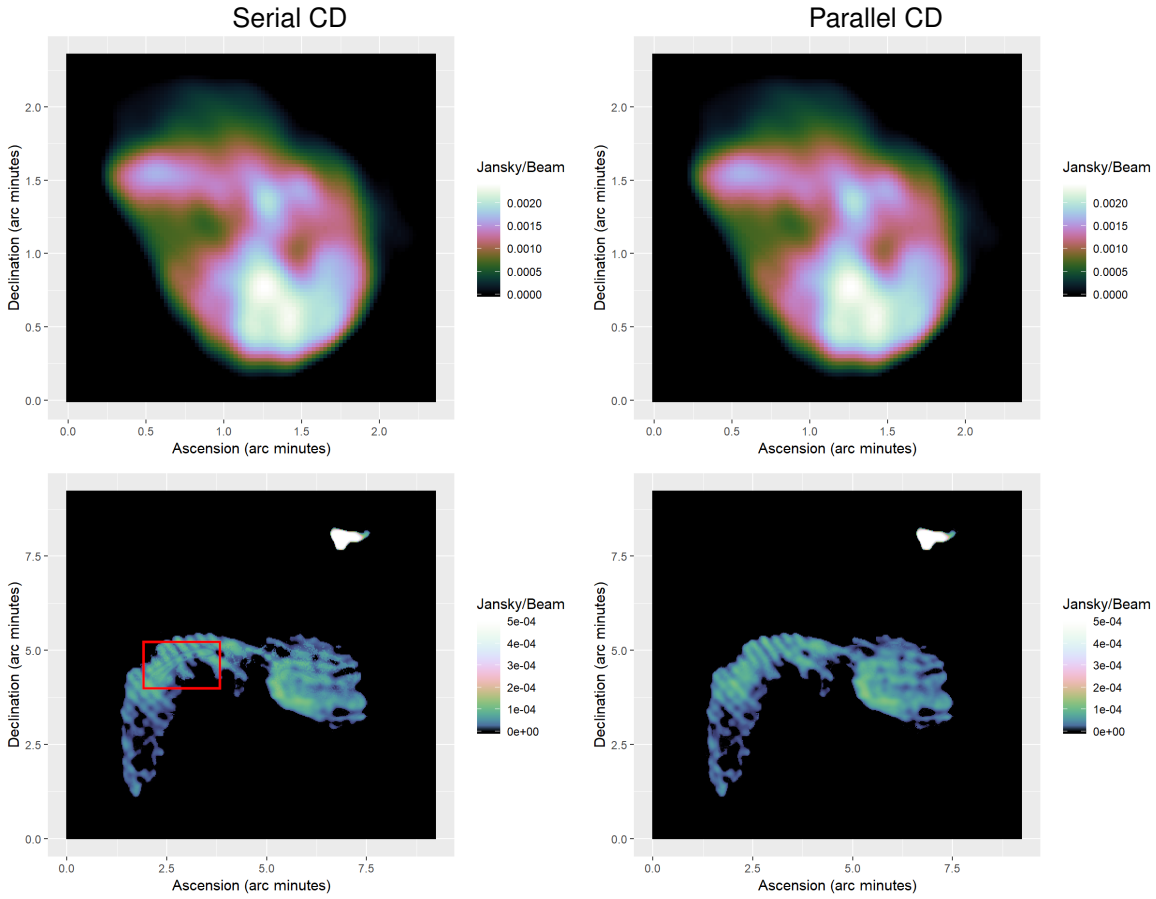


Figure 35: Comparison of the serial and parallel coordinate descent reconstruction of the LMC observation.

Both algorithms arrive at the same reconstruction. Both can super-resolve the N132D supernova-remnant, and have artifacts due to calibration errors. But the serial coordinate descent algorithm has left artifacts from the previous major cycles in the reconstruction: The red rectangle highlights a structure, which was added in a previous major cycle with the implicit path regularization. In a previous major cycle, only the highlighted structure was added. The next major cycle then added all the surrounding structure, including the ”waves” from the calibration error. But the serial coordinate descent algorithm did not have enough iterations to properly integrate the old structure from the previous major cycle, leaving a ”ghost” structure behind.

The serial coordinate descent can correct the artifact, but it either needs more iterations, or even another major cycle. In either case the serial coordinate descent algorithm needs more computing resources.

The parallel coordinate descent algorithm on the other hand does not contain the ”ghost” structure. In a previous major cycle, the parallel algorithm has added a similar structure, but it was able to properly integrate it in the final reconstruction. We suspect this difference is due to the different number of pixels minimized: The reconstruction of the parallel algorithm is the result of roughly half a million single pixel minimizations, while the serial algorithm was had performed roughly 100 thousand single pixel minimizations. The parallel algorithm had more iterations to integrate the structures detected from previous cycles.

Overall the reconstruction of the serial and parallel coordinate descent algorithms are similar. On the LMC dataset, the parallel algorithm was able to reconstruct a slightly superior reconstruction within a fraction of the total run time of the serial algorithm.

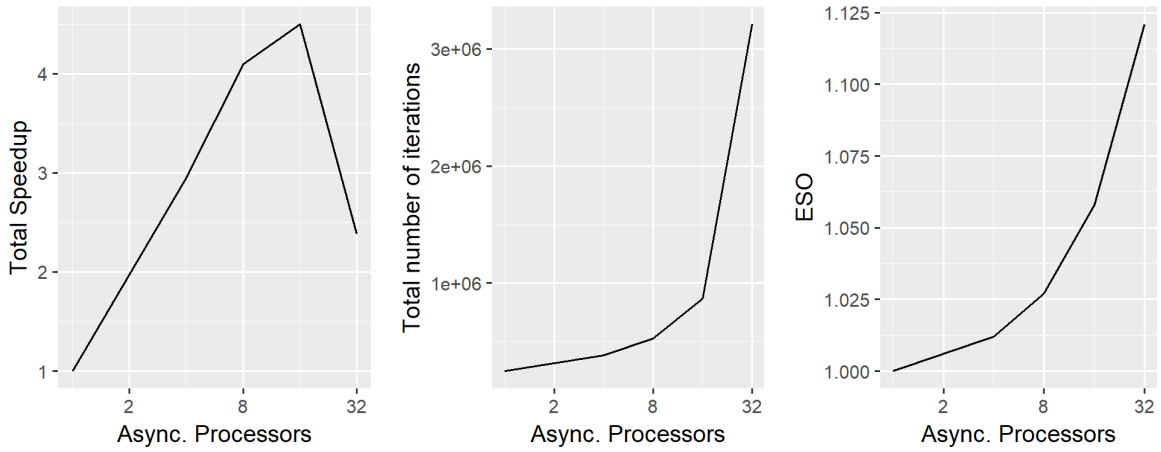


Figure 36: Speedup comparison with increasing number of asynchronous processors.

7.6 Scalability of the parallel coordinate descent algorithm

The parallel coordinate descent algorithm out-performs the serial algorithm significantly on personal computing hardware. Lastly, we want to investigate how the parallel algorithm behaves when we increase the number of asynchronous processors.

Our parallel coordinate descent algorithm is based on PCDM[25]. PCDM has been extended for the distributed setting as Hydra[32], and in its accelerated variant as Hydra^2[11]. Extending our parallel coordinate descent algorithm to the distributed setting was not possible in the time frame of this project. But if the parallel algorithm should be distributed in the future, we can apply the methods developed for Hydra and Hydra^2.

We test the parallel coordinate descent algorithm on a shared memory system (all processors have access to the same main memory), and test how the algorithm behaves for larger number of processors. Figure 36 shows the results. The first graph shows the total speedup, the second plot shows the total number of iterations needed, and the last plot shows the ESO for increasing number of processors.

We see a linear speedup up to 8 processors. But afterwards, the speedup we receive diminishes quickly. After 16 processors, the parallel coordinate descent algorithm even becomes slower when more processors are added. The reason for this behavior lies in the total number of iterations the parallel algorithm needs to reconstruct a solution. As we see in Figure 36 needs over 3 million parallel iterations to converge with 32 asynchronous processors. With 8 processors, it about half a million iterations to converge.

This drastic increase in total iterations quickly diminishes the speedup we receive by adding more asynchronous processors. This is partly due to the increase in the ESO: Adding more asynchronous processors increases the ESO, which in turn reduces the step size by which we can minimize a pixel in each iteration. However, we suspect there are more effects at play than just the ESO. The ESO for the number of processors is also shown in Figure 36 does not increase as drastic as the total number of iterations needed.

Processors	Iterations per second
1	1061.7
4	3849.7
8	9254.9
16	16’721.9
32	32’794.6

Table 3: Throughput of the parallel coordinate descent algorithms with additional processors.

When we look at the throughput of the parallel coordinate descent algorithm in table 3, we see that the number of iterations per second still increases roughly linearly with the number of asynchronous processors. If the parallel algorithm gets slowed down because of communication costs between processors (for example: multiple compare-exchange operations on the same entry in the gradient map), we would see a less-than-linear increase in throughput per added processor. Meaning the slowdown we see in Figure 36 is likely not due to communication costs, but due to the drastic increase in total number of iterations.

Remember that the parallel coordinate descent algorithm uses active set iterations: Each asynchronous processor chooses pixels from the active set for a given number of iterations. When we measure the absolute maximum pixel difference (the maximum step a serial greedy coordinate descent algorithm can take in one iteration) for 8 processors, we observe that it steadily decreases from one active set iteration to the next. At 32 processors, the absolute maximum pixel difference fluctuates from one active set iteration to the next.

This observation leads us to the ESO and our pseudo-random selection strategy: We may see the fluctuation, because we do not choose pixels uniformly at random, and their PSFs overlap more than we estimated with the ESO. In that case, the algorithm is in the danger of diverging, which would explain the fluctuation. The solution in that case would be to increase the ESO.

On the other hand, this fluctuation can also be observed with a proper random selection strategy. The second explanation is that with increasing number of processors, a single processor is simply too close to a random selection strategy. Remember: Our pseudo-random strategy greedily searches a fraction of the active set. With an active set of 1024 entries, 32 processors and a search fraction of 0.1, each processor searches 10% of $\frac{1024}{32}$ entries. By adding more processors, each processor searches through fewer entries, although the total number of entries searched stays the same. The solution in that case would be to increase the Search Fraction.

We tested these two explanations: We ran the parallel coordinate descent algorithm once with an increased ESO (the ESO which arises from 64 processors) and once with an increased Search Fraction. The results are summarized in Table 4.

Test	Processors	ESO	Search Fraction	# Iterations	Total Seconds	Speedup
Standard	16	1.058	0.1	868k	51	4.5
Standard	32	1.121	0.1	3’213k	98	2.4
Larger ESO	32	1.246	0.1	4’220k	130	1.8
Larger Search	32	1.121	0.4	1’171k	47	**5.0**

Table 4: Speedup comparison for the parallel coordinate descent algorithm, once with increased ESO and once with increased Search Fraction.

Increasing the Search Fraction lead to a significant decrease in total iterations, which in turn leads to a significant decrease in the wall-clock time, leading to a speedup factor of 5. This means, with a larger Search Factor, the parallel coordinate descent algorithm is again faster with 32 processors instead of 16, and it would remove the ’dip’ in Figure 36. Noteworthy is that increasing the ESO in our parallel algorithm again lead to an increase in total number of iterations. This result suggests the ESO was not too low, and was not at fault for the slowdown we measured in Figure 36.

Nevertheless, these results are still surprising to us. As we mentioned before, whether we use 1 processor or 16 in our parallel coordinate descent algorithm, we still search the same number of elements in the active set. For a Search Fraction of 0.1, the algorithm always searches 10% for every parallel iteration. Adding more processors simply reduces the number of elements each processor checks. The results from Table 4 suggests that each processor in our parallel algorithm has to search through a minimum number of elements in the active set to run efficiently. But this view does not agree with our results from Figure 33, where we used 8 processors and a Search Fraction of 0.01. Each of the 8 processors searched through fewer elements than

in this test with 32 processors and a Search Fraction of 0.1. But with 8 processors, it did not lead to a large increase in wall-clock time.

It is not clear why exactly our parallel coordinate descent algorithm needs so many more iterations to converge with 32 processors, or why increasing the Search Fraction can remedy the problem. But the total number of iterations needed to converge seem to be the bottleneck for this algorithm. A thorough analysis may lead to a variant of our algorithm which needs fewer iterations to converge and therefore may be even faster. In the time frame of this project, it was not possible to further analyze the behavior.

8 Discussion

In this project, we developed a novel PSF approximation scheme. It leverages the fact that large parts of the PSF from modern interferometers are close to zero, and we can use a sparse approximation of the true PSF. We then developed a parallel coordinate descent deconvolution algorithm that benefits from a sparse PSF. We were able to create a non-blocking parallel implementation, which achieved a speedup factor of 20 compared to the serial coordinate descent deconvolution algorithm, and an estimated speedup of factor 3 to CLEAN.

The parallel coordinate descent algorithm can efficiently use multiple processors to speed up the deconvolution. It is an algorithm where additional processors lead to a small amount of additional overhead. Its tuning parameters behave in an unexpected way for large number of processors (≈ 32). Currently, the parallel algorithm requires an unexpected large number of iterations to converge. By changing its tuning parameters, one can easily reduce the number of iterations needed. The reason why the parallel coordinate descent algorithm behaves in this way is currently unknown. Further research, which aims to understand the behavior, may lead to a parallel algorithm that requires fewer iterations to converge with additional processors. There may still be additional ways to speed up the parallel coordinate descent algorithm.

The degree of parallelism, the number of parallel deconvolutions which can be used efficiently of the parallel algorithm, scales with both the field-of-view of the observation, and the approximate PSF. With larger field of views, and with smaller approximated PSFs, the parallel coordinate descent algorithm can efficiently use a additional processors. This suggests the parallel coordinate descent algorithm can potentially scale with the planned expansion of the MeerKAT interferometer. Under SKA-Mid, the MeerKAT interferometer is planned to receive additional antennas spaced further apart. This increases the resolution of the instrument, together with its data volume. A higher resolution leads to a more concentrated PSF. Our approximation scheme may produce an even smaller approximated PSF to a comparable error, which again speeds up the parallel coordinate descent algorithm.

We created a shared-memory implementation of the parallel coordinate descent algorithm. Under SKA-Mid, the size of the deconvolution problem may potentially exceed the memory capacity of a single machine, requiring a distributed algorithm for deconvolution. Our parallel algorithm is based on the APPROX[26], which was extended to a distributed setting developed in [33]. The same extension can be used to extend our parallel coordinate descent algorithm to a distributed setting. However, it is not clear whether this is necessary.

Modern reconstruction pipelines work in the self-calibration setting: Intermediate solutions of the deconvolution are used to improve the calibration of the visibility measurements. New self-calibration methods account for distortions introduced by the Ionosphere, which change over the the image. This naturally splits the image into smaller facets, where each facet is deconvolved independently[34]. The total number and size of each facet depends on the objects in the observation, the science target and the radio interferometer itself. To our knowledge, it is currently not clear how large one can expect a facet to be.

Our parallel coordinate descent algorithm scales with the amount of sparsity our approximation method can introduce. In our tests, the parallel algorithm converged with an approximate PSF which was $\frac{1}{32}$ of the total image size. If the image is split into smaller facets, then the size difference between the approximate PSF and the facet is significantly smaller. If the approximate PSF is larger than the facet, the parallel algorithm may not be able to use all available processors effectively. On the other hand, if the approximate PSF is smaller than the facet, our parallel algorithm can significantly speed up the deconvolution. The smaller the approximate PSF is compared to the facet, the more processors can be used efficiently, and our parallel coordinate descent algorithm can be faster to converge than multi-scale CLEAN.

The speedup achieved by the parallel coordinate descent method depends on the size of the image to deconvolve, and the size of the approximate PSF. Self-calibration with facets may effectively reduce the image size. But observations which need self-calibration tend to have a large amount of visibility measurements, which

in turn may also lead to a smaller approximate PSF. The PSF approaches a 2d Gaussian with increasing number of visibility measurements. This reduces the error our PSF approximation introduces. With more visibility measurements, it may produce a smaller approximate PSF with a similar error. The parallel coordinate descent algorithm in turn can achieve higher speedups with a smaller approximate PSF. If the parallel coordinate descent algorithm can beat multi-scale CLEAN in a self-calibration task has to be evaluated in the future.

The PSF approximation scheme and the parallel coordinate descent algorithm were tested on a MeerKAT reconstruction. They can also be used on observations created by different interferometers Our approximation scheme exploits that the PSF approximates a 2d Gaussian function with increasing number of measurements. For medium- to high-frequency observations, our approximate PSF can be only a fraction of the full PSF. For low-frequency observations however, the 2d Gaussian function becomes larger. For the same pixel resolution, the approximate PSF also becomes larger. Our approximation scheme and parallel algorithm may not achieve the same speed up on low-frequency observations, depending on the image size and pixel-resolution.

Our serial and parallel coordinate descent algorithms used the elastic net regularization. It is a mixture between the L1- and L2-norms. On our test, the elastic net regularization lead to restored images which were comparable to state-of-the-art deconvolution algorithms like multi-scale CLEAN and MORESANE. The model image resulting from elastic net contained plausible structures, potentially super-resolving the N132D supernova-remnant. However, the model images resulting from elastic net also seemed more susceptible towards calibration errors. This seems to be a general behavior of deconvolution algorithms based on convex optimization[5] [7].

Our results suggests elastic net may be an alternative to multi-scale CLEAN in terms of reconstruction quality, and combined with the parallel coordinate descent algorithm, also in terms of wall-clock time. Our current implementation has one main drawback compared to multi-scale CLEAN algorithm: It leaves a part of the true emission inside the residual image. The pixel magnitudes of the residual image are significantly larger than those of the multi-scale CLEAN residual image. This is at least in part due to the auto-masking strategies used in multi-scale CLEAN: After a certain number of iterations, multi-scale CLEAN is only allowed to change non-zero pixels in the model image. It is forbidden to add new sources to the model. Our serial and parallel coordinate descent algorithm can be extended with a similar auto-masking strategy to CLEAN. This is a necessary extension for any future parallel coordinate descent implementation.

To our knowledge, there is currently little interest in elastic net as a regularization for radio interferometric imaging. Currently, over-complete representations are often used as a regularization. Two frequently used examples are the Starlet [15] and Daubechies wavelet regularization [35]. Both have been shown to produce super-resolved reconstruction in radio astronomy[3, 36]. Compared to the over-complete regularizations. The parallel coordinate descent algorithm is not limited to the elastic net regularization. It can be extended to use Starlets or Daubechies wavelets. However, the parallel algorithm achieves its speedup by exploiting sparsity in the approximate PSF. Elastic net is a pixel-wise regularization, and does not affect the sparsity in the approximate PSF. A regularization like Starlets affects a neighborhood of pixels, which in turn reduces the sparsity in the approximate PSF. The larger the neighborhood, the larger is the combined PSFs of all pixels. Nevertheless, depending on the size of the neighborhood there may still be enough sparsity to exploit with the parallel coordinate descent algorithm. If this leads to a faster algorithm than MORESANE (which also uses the starlet regularization) has to be evaluated in future works.

The implementation of our parallel coordinate descent algorithm does not account for wide-band imaging effects, which is the main limitation of our work. In wide-band imaging, the a deconvolution algorithm has to reconstruct three dimensional image cube, where each image represents a single sub-band. Additionally, a

[5] In this project, we optimized a convex objective function ($\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * PSF\right\|_2^2 + \lambda ElasticNet(x)$). CLEAN optimizes a non-convex objective function.

wide-band deconvolution algorithm needs a regularization across the frequency axis [37]. Wide-band imaging allows for another dimension of parallelization [37]. The parallel coordinate descent algorithm was able to efficiently use multiple processors for deconvolving a single image. In our case, wide-band imaging can potentially lead to a higher degree of parallelism.

The parallel coordinate descent algorithm was tested on a narrow-band imaging problem. It is unknown how it compares to state-of-the-art wide-band imaging algorithms like multi-scale, multi-frequency-synthesis CLEAN (MS-MFS-CLEAN). Interferometers like MeerKAT measure visibilities in 20'000 channels simultaneously. A wide-band deconvolution algorithm is necessary for leveraging the full potential of MeerKAT measurements. The next step for the parallel coordinate descent algorithm is to extend it to the wide-band imaging case, and compare it to state-of-the-art MS-MFS-CLEAN algorithm.

9 Conclusion

In this project, we developed our own image reconstruction pipeline in .Net Core. We implemented the Image Domain Gridder[12] and developed two deconvolution algorithms: A serial and a parallel coordinate descent deconvolution algorithm in the Major/Minor cycle architecture.

For this project, we developed the hypothesis that a deconvolution algorithm in the Major/Minor cycle architecture may be speed up by using an approximate PSF for deconvolution. The deconvolution algorithm already uses an approximation of the true PSF. We showed that the PSF can be further approximated, which simplifies the deconvolution problem for parallel and distributed computing. In this project, we created a novel PSF approximation scheme for the Major/Minor cycle architecture, and implemented a parallel coordinate descent algorithm which can exploit the approximated PSF for a significant speedup. On our test, the parallel coordinate descent algorithm is estimated to outperform standard CLEAN in convergence speed.

The parallel coordinate descent algorithm can use modern non-blocking instructions, running in parallel deconvolutions with little communication overhead. Our implementation is optimized for the CPU on a shared-memory system. The algorithm can easily be extended for the distributed setting, where different machines deconvolve the image in parallel. The parallel coordinate descent algorithm showed competitive convergence speeds in comparison to CLEAN on the CPU. These results may be further improved by using GPU acceleration for the parallel algorithm.

The degree of parallelism scales with the problem size for the parallel coordinate descent algorithm. Large deconvolution problems can be efficiently solved by adding additional processors. This algorithm has the potential to scale with the planned expansion of MeerKAT to SKA-Mid. However, the imaging problem will also change together with MeerKAT's expansion. It remains to be seen if the parallel coordinate descent algorithm is competitive on the SKA-Mid imaging task.

We tested our PSF approximation scheme and parallel coordinate descent algorithm on a real-world observation of MeerKAT. The approximation scheme and parallel algorithm are not inherently limited to a single instrument. However, our PSF approximation scheme tends to be more effective on medium- to high-frequency observations. The PSFs of low-frequency observations cannot be approximated as efficiently. In turn, the parallel coordinate descent algorithm may not achieve the same speedups on low-frequency observations.

We used the elastic net regularization. To our knowledge, it is not widely used in the radio astronomy community. On our test, the elastic net regularization created more plausible reconstructions of radio sources than multi-scale CLEAN. The elastic net was also more sensitive to calibration errors in the image. This is a similar behavior to over-complete regularizations, which are often used for radio interferometric image reconstructions. However, elastic net is a significantly simpler regularization. Leading to a simpler reconstruction algorithm in a parallel or distributed setting.

We demonstrated with the parallel coordinate descent algorithm that an elastic net regularized image can be reconstructed efficiently. These results suggests the elastic net regularization may be a viable alternative for radio interferometric image reconstruction. We tested the elastic net regularization on a single real-world observation. The question, whether elastic net is a viable alternative for image reconstruction, has to explored on multiple observations in the future.

We limited this project to the narrow-band image reconstruction only. This is the main limitation of this work. Wide-band imaging introduces new challenges for an image reconstruction algorithm. However, the parallel coordinate descent algorithm also has the potential to benefit from it. We may increase the degree of parallelism further in the wide-band imaging case. It is unknown how our parallel coordinate descent algorithm compares to multi-scale CLEAN in the wide-band case. The next step is extending the parallel coordinate descent algorithm to the wide-band imaging case.

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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] 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.

[17] 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.

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

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

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

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

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

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

[24] 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.

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

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

[27] Olivier Fercoq and Zheng Qu. Restarting accelerated gradient methods with a rough strong convexity estimate. arXiv preprint arXiv:1609.07358, 2016.

[28] Tobias Glasmachers and Ürün Dogan. Coordinate descent with online adaptation of coordinate frequencies. arXiv preprint arXiv:1401.3737, 2014.

[29] 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.

[30] 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.

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

[32] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. The Journal of Machine Learning Research, 17(1):2657–2681, 2016.

[33] Olivier Fercoq, Zheng Qu, Peter Richtárik, and Martin Taká˘c. Fast distributed coordinate descent (hydra 2).

[34] RJ Van Weeren, WL Williams, MJ Hardcastle, TW Shimwell, DA Rafferty, J Sabater, G Heald, SS Sridhar, TJ Dijkema, G Brunetti, et al. Lofar facet calibration. The Astrophysical Journal Supplement Series, 223(1):2, 2016.

[35] 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.

[36] 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.

[37] 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.

List of Figures

List of Tables

Attachment

10 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, December 30, 2019

Jonas Schwammberger