P9 Distributed Image Reconstruction for the new Radio Interferometers

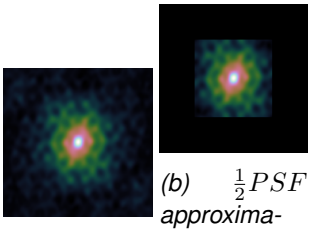
Jonas Schwammberger

January 14, 2020

1 PSF approximation for fast and distributed deconvolution

This section describes our main hypothesis of this work: We can further approximate the PSF in the Major/Minor cycle and exploit it to speed up/distribute coordinate descent based deconvolution algorithms.

A deconvolution algorithm in the Major/Minor cycle works with a PSF, which is constant over the whole image. However, this is not the case for modern radio interferometers. The w-term in the visibility measurements modify the true PSF over the image. This is why we need the Major cycle: After a number of iterations of the deconvolution algorithm, the Major cycle transforms the model image back to visibility measurements, and subtracts the model from the measured visibilities. Then, it transforms the residual visibilities back into image space. The gridder corrects for the w-term. It corrects the error the deconvolution algorithm introduces with a constant PSF. The Major cycle allows us to use an approximation of the true PSF in the deconvolution. We believe this can be exploited to speed up/distribute the deconvolution.



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

Figure 1: PSF approximation typically used in Clark CLEAN

An approximation of the full PSF has already been developed for CLEAN: The Clark CLEAN algorithm [1] uses only a window of the full PSF for the deconvolution. Figure 1 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 1 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.

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 on our serial and parallel coordinate descent algorithm in Section **??**. In this section, we show how we can incorporate an approximate PSF into our coordinate descent algorithms. We discuss only the serial coordinate descent algorithm in this section. The approximation works identical for the parallel coordinate descent algorithm.

1.1 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.

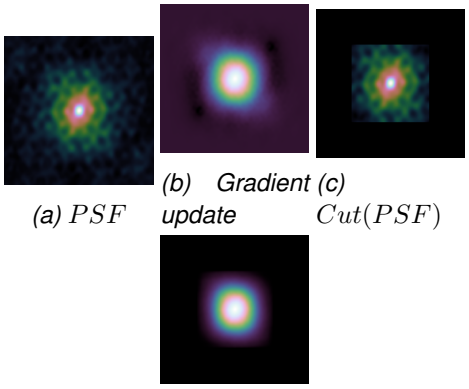
We approximate the PSF by only using a window around the center. Each side is only a fraction of the full PSF size. From now on, we use the method $Cut()$, which cuts out the center window of the PSF. If we use a cut fraction of $\frac{1}{4}$, we cut out a window of the PSF, each side being $\frac{1}{4}$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

Using an approximate PSF influences three parts of the serial coordinate descent 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'.

1.1.1 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 2 shows the effect of the approximation.



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

Figure 2: 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.

1.1.2 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:

$$\underset{x}{minimize} \frac{1}{2} \|I_{dirty} - x * Cut(PSF)\|_2^2 + \lambda ElasticNet(x) \tag{1.1}$$

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 underestimates 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.

1.1.3 Combining the two approximation methods

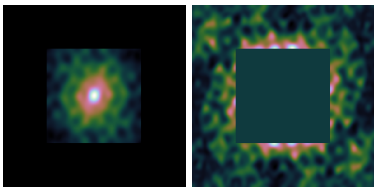
The two approximation methods developed here have opposing downsides: Method 1 is guaranteed to converge to the same result, given enough major cycles, but becomes increasingly inaccurate with more serial coordinate descent iterations. Method 2 does not become increasingly inaccurate, but is not guaranteed 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 or parallel coordinate descent has converged.

1.2 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 3 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 3: 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 3b. 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[2]. 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:

$$\begin{aligned} maxSidelobe &= Max(PSF - Cut(PSF)) \\ gradients &= residuals \star PSF \\ maxLobeGradient &= Max(gradients) * maxSidelobe \\ \lambda_{cycle} &= \frac{maxLobeGradient}{\alpha} \\ \lambda_{cycle} &= Max(\lambda, \lambda_{cycle}) \end{aligned} \tag{1.2}$$

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:

$$\begin{aligned} maxSidelobe &= Max(PSF - cut(PSF)) \\ gradients &= residuals \star PSF \\ \textbf{correction} &= Max\left[1, \frac{Max(gradients)}{Max(residuals) * Lipschitz}\right] \\ maxLobeGradient &= Max(gradients) * maxSidelobe * correction \\ \lambda_{cycle} &= \frac{maxLobeGradient}{\alpha} \\ \lambda_{cycle} &= Max(\lambda, \lambda_{cycle}) \end{aligned} \tag{1.3}$$

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.

2 Parallel coordinate descent methods

In this section, we introduce the parallel coordinate descent algorithm, which benefits significantly from our PSF approximation scheme. The PSF approximation we developed lets the deconvolution algorithm use an approximate PSF. It uses only the center window of the full PSF, where each side of the window is a fraction of the full PSF. Or in other words: The approximation method resulted in a sparse PSF for the deconvolution algorithm. We use the introduced sparsity for parallel deconvolutions, resulting in the parallel coordinate descent algorithm

The serial coordinate descent algorithm minimizes a single pixel in each iteration. Parallel coordinate descent methods can minimize multiple pixels in parallel in each iteration. This section, we introduce the parallel coordinate descent deconvolution algorithm. We created two implementations: One implementation is more general, it uses gradient acceleration to speed up convergence, and can group multiple pixels into blocks. It is a parallel algorithm, which can minimize multiple blocks in parallel in each iteration. But grouping pixels into blocks and gradient acceleration did not lead to an overall faster deconvolution algorithm. This section focuses on our second implementation, which is not accelerated and can only optimize pixels instead. The introduction of the accelerated, parallel, block coordinate descent algorithm and its results can be found in the attachments.

The parallel coordinate descent algorithm can update multiple pixels in parallel. However, if it updates pixels which are next to each other in the image, their PSFs overlap and it over-estimates their pixel values. This can lead the parallel algorithm to diverge. In section 2.1 we introduce the core concept of parallel coordinate descent methods: The Estimated Separability Overapproximation (ESO). We then show how the parallel coordinate descent algorithm can be implemented in an asynchronous manner in Section 2.1. Parallel coordinate descent algorithms have been developed with a random selection strategy in mind. However, a random selection strategy does not perform well on the deconvolution problem. We demonstrate the problem of random selection and introduce our own solution in

Section 2.2.

2.0.1 Estimated Separability Overapproximation (ESO)

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

However, if we update two pixels, 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 pixel, and then the second pixel in a serial algorithm results in significantly lower pixel values for the second pixel. Because their PSFs overlap, both pixels try to explain mostly the same emission. If we update both pixels in parallel, each pixel would try to explain the same emission, and we would overestimate their pixel values.

This over-estimation can lead to a diverging algorithm. To guarantee the convergence of a parallel pixel 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 [3]. The ESO estimates how much the PSFs overlap, if we update τ random pixels in parallel:

$$ESO(\omega, \tau, n) = 1 + \frac{(\omega - 1)(\tau - 1)}{max(1, n - 1)} \qquad (2.1)$$

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 pixels 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 64^2 pixels in size. Plugging the values into the ESO gives us the following result:

$$ESO(\omega, \tau, n) = 1 + \frac{(24 - 1)(4 - 1)}{max(1, 4096 - 1)} \approx 1.017 \qquad (2.2)$$

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 $\tau = 4$ pixels uniformly at random. Indeed, a uniform random selection strategy is a core assumption for the parallel coordinate descent method[3]. Random selection strategies tend to perform badly on the deconvolution problem. Later in Section 2.2, 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.

2.1 Asynchronous parallel coordinate descent algorithm.

So far, we introduced the serial coordinate descent and the ESO. The serial block coordinate descent can update a pixel in a single iteration, and the ESO estimates how much PSFs overlap when we perform parallel update steps. In this section, we put the ESO and the serial algorithm together into a parallel coordinate descent algorithm based on APPROX[4]. The implementation is asynchronous, meaning each processor chooses its pixel to minimize, and updates the gradient map independently of the other processors.

In this project, we use a τ-nice uniform sampling, which can be easily implemented with asynchronous processors: Each of the τ processors

chooses its pixel to minimize uniformly at random. The asynchronous processors are only forbidden to select the same pixel. To ensure this in the parallel algorithm, we introduce a new array: The 'pixelLocks'. Each asynchronous processor writes its processor id in the pixelLocks map for the pixel it is currently updating. If there is already a processor id written at the specific location, the processor simply selects another random pixel.

The write to the pixelLocks array can be easily implemented with a compare-exchange operation. Compare-exchange is an atomic instruction on modern computing hardware. It checks for a value at a specific memory location. If the check returns true, it exchanges the value at the memory location. The compare-exchange instruction is atomic, meaning there cannot be another process which has modified the memory location in the meantime. This leads us to the following algorithm:

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

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

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

eso = ESO(CountNonZero(PSF), t, x.Length)

concurrentIterations = 1000 / t
pixelLocks = new Array[,]

do
  //asynchronous iterations
  parallelDiffs = new Array[]
  parallel for each processorId in (0:t)+1
    for concurrentIter in 0:concurrentIterations
      pixelLocation = AtomicLockRandomPixel(pixelLocks, processorId)

      //Step 2: update pixel asynchronously
      lipschitz = LipschitzMap[pixelLocation]
      lipschitz = lipschitz * eso

      oldValue = x[pixelLocation]
      tmp = gradientsMap[pixelLocation] + oldValue * lipschitz
      optimalValue = Max(tmp - lambda*alpha) / (lipschitz + (1 - alpha)*lambda)
      diff = optimalValue - oldValue

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

      //Step 3: Update gradients asynchronously
      shiftedUpdate = Shift(gradientUpdate, pixelLocation)
      AtomicSum(gradientsMap, -shiftedUpdate * diff)

      //unlock pixel
      pixelLocks[pixelLocation] = 0
  maxParallelDiff = Max(parallelDiffs)
while maxParallelDiff < epsilon
```

Each of the τ processors performs a number of iterations independently of the other processors. This is what we call an asynchronous 'batch' of iterations. In each asynchronous iteration, a single processor chooses a pixel to minimize uniformly at random, which is not locked by a different processor. We multiply the Lipschitz constant with the ESO, which ensures convergence for parallel updates. The gradient map gets updated asynchronously. Each processor writes its changes atomically to memory. We also implemented the atomic sum with a compare-exchange operation on the CPU.

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.

2.2 The problem with random selection for deconvolution

Our parallel coordinate descent algorithm developed in this section does not perform well on the deconvolution problem. The reason lies in the random selection strategy: In the first few iterations, the deconvolution algorithm selects pixels 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 pixels. Before the parallel algorithm can make a meaningful update to a neighboring pixel, it first needs to select the same pixel again. In short, the first iterations always over-estimate the pixel values, which leads to slow convergence rates.

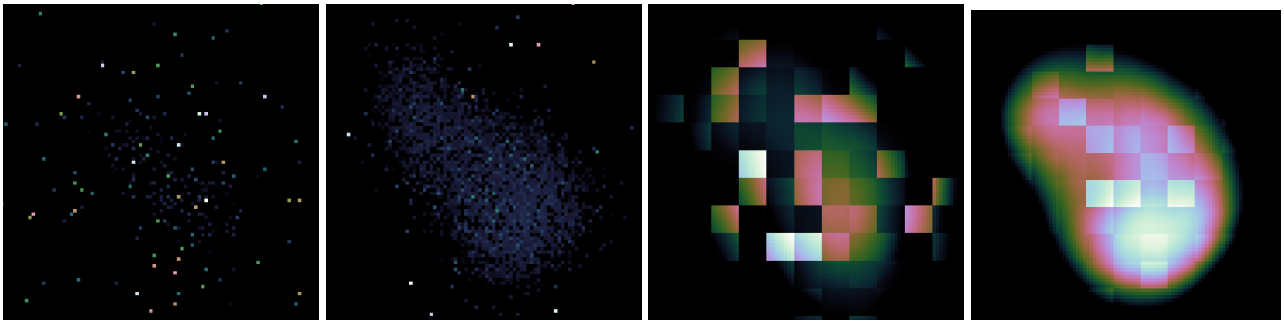
The Figure 4 shows the behavior on the LMC observation. The reconstructions receive obvious artifacts from the random selection strategy. The pixels, which get selected in the first few iterations, keep their over-estimated values. The parallel algorithm needs to select them several times to reduce their value. That is why even after 80k iterations, the N132D supernova remnant gets only hinted at in Figure 4b. Until the over-estimated pixels 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 4d, 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 pixel at random, but greedily search in the neighborhood for the optimal pixel to optimize. The size of the neighborhood can be defined by the user as the 'Search Factor' parameter. It is essentially a mix between a greedy and a random selection strategy. If we choose a Search Factor of 1.0, the neighborhood is the whole image, and we arrive at a greedy strategy. If we choose a Search Factor of 0.0, then the neighborhood is one pixel, and 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 ESO. The optimal value for the Search Fraction is explored later in Section 3.2.1.

We have developed three related extensions, which speed up the parallel coordinate descent algorithm in practice. We use an active set, a restarting strategy, and a 'Minor' cycle. The active set only allows the parallel algorithm to choose from a subset of pixels, which are likely to be non-zero in the final image. The restarting strategy resets the active set when it may not contain relevant pixels. Restarting is an important strategy for gradient accelerated algorithms, which have not been discussed in this section. Lastly, we re-introduce a 'Minor' cycle. The parallel coordinate descent algorithm can exploit our PSF approximation scheme. With increasingly small PSF windows, the parallel algorithm achieves ever faster convergence times. But the downside is it also requires more Major cycles to converge. To combat this problem, we periodically reset the residual image: We convolve the intermediate solution with the full PSF, and subtract it from the residuals. This is what we call a 'Minor' cycle. How these three extensions work in detail can be found in the attachments.



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

Figure 4: Random parallel deconvolutions on the LMC N132D supernova remnant.

3 Tests on MeerKAT LMC observation

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 5 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 5 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 $\approx$77 arc minutes). 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.

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

In Section **??** 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.

3.1 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 [5] implementation of multi-scale CLEAN and MORESANE (IUWT)[1], 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 func-

[1]The WSCLEAN package has a re-implementation of the original MORESANE algorithm[6], which is called IUWT.

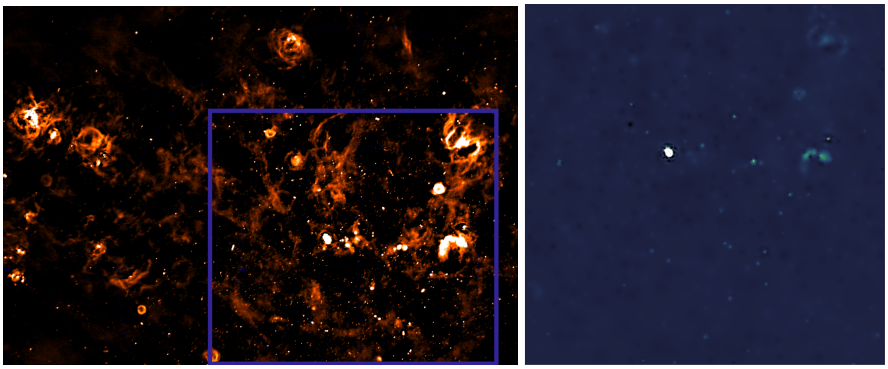


Figure 5: Narrow band image section used.

tion representing the instrument's accuracy[2]. 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 5 in the residuals. The Figure 6 shows the model- and residual image of multi-scale CLEAN, serial coordinate descent and MORESANE.

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.

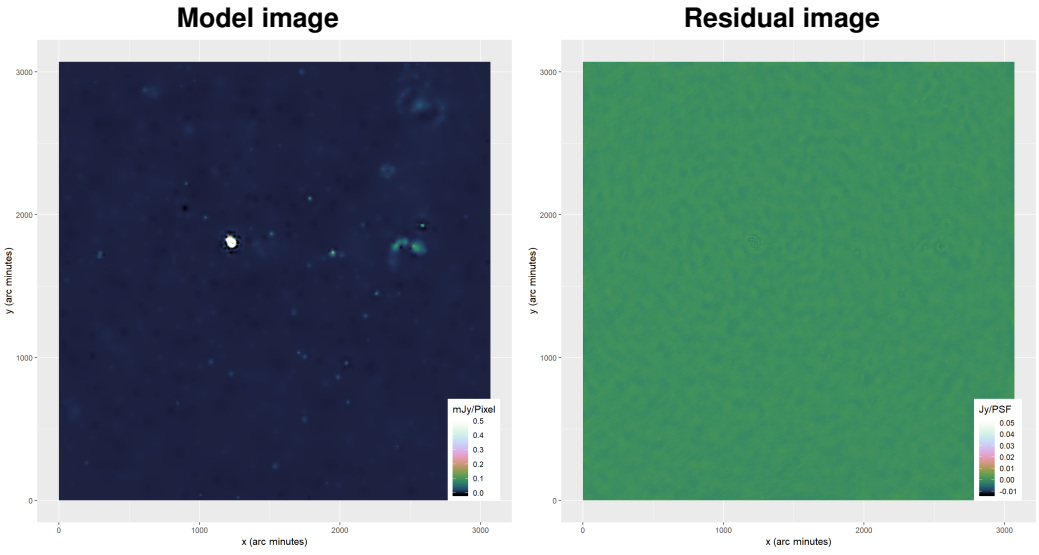
Even with an auto-masking strategy, the MORESANE algorithm left notable emissions in the residual image. The serial coordinate descent algorithm with elastic net regularization does not use an auto-masking strategy. Nevertheless, its residual image has lower magnitudes than MORESANE. These results may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

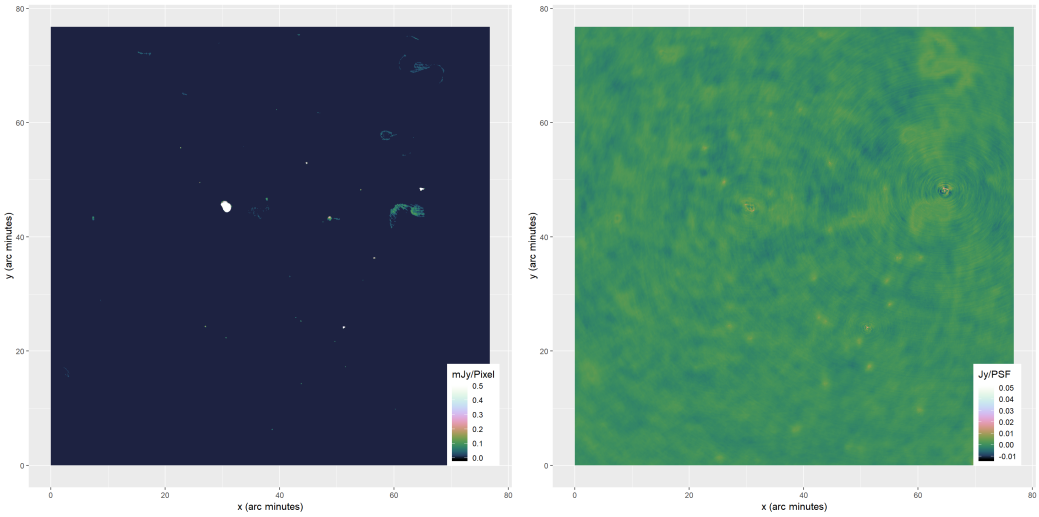
3.1.1 Super-resolution of the N132D supernova-remnant

In Figure 7. 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

[2] Usually, the residuals are added to the restored image. We compare the restored images without the added residuals.

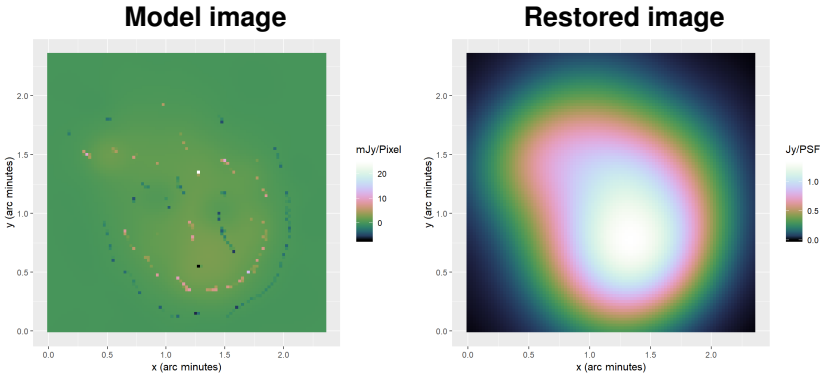


(a) Multi-scale CLEAN.

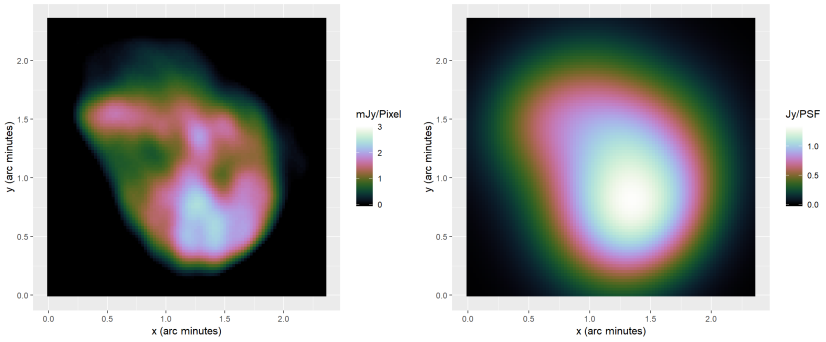


(b) Coordinate descent with elastic net.

Figure 6: Comparison of the whole image



(a) Multi-scale CLEAN.



(b) Coordinate descent with elastic net.

Figure 7: Comparison on the N132D supernova-remnant.

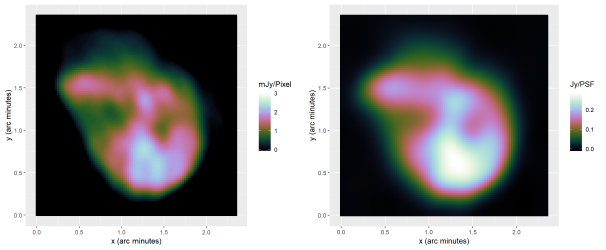
multi-scale CLEAN has removed almost all the relevant emission. This leads to a difference in the magnitudes of the restored image.

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.

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[7]. 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 8. 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) Briggs MS-CLEAN*

Figure 8: 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.

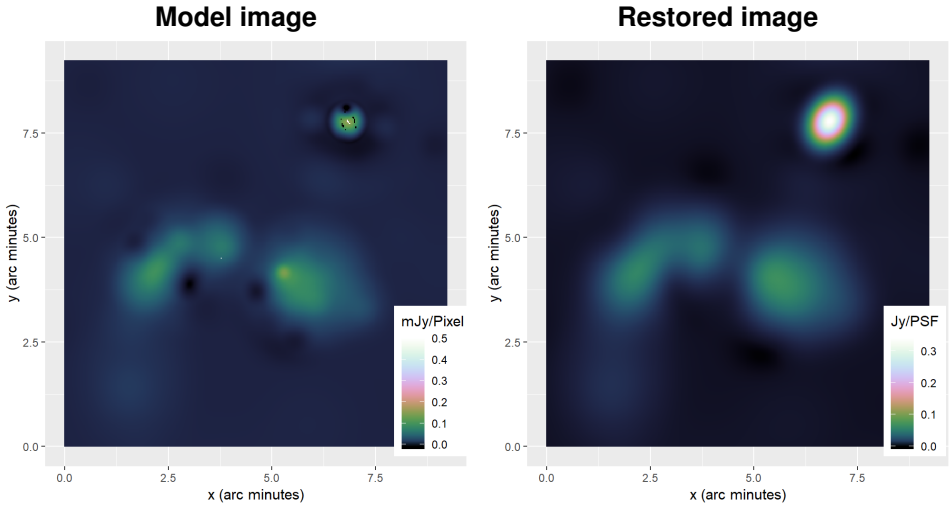
3.1.2 Influence of calibration errors

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 10 compares the model and restored images of the three algorithms.

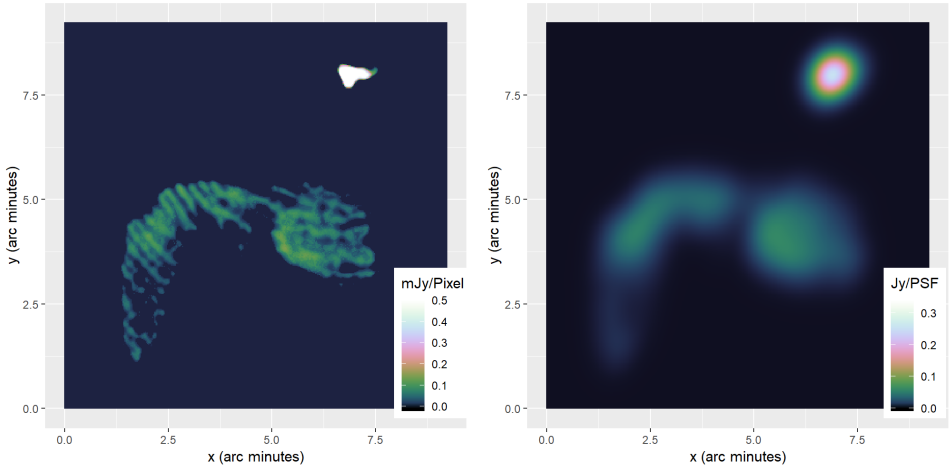
The multi-scale CLEAN and MORESANE model images contain significant negative pixel values in this section of the reconstruction. For a reasonable comparison, we had to clip the negative values, which are now shown as black pixels in the model images. Due to the bright point source in the image, we also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

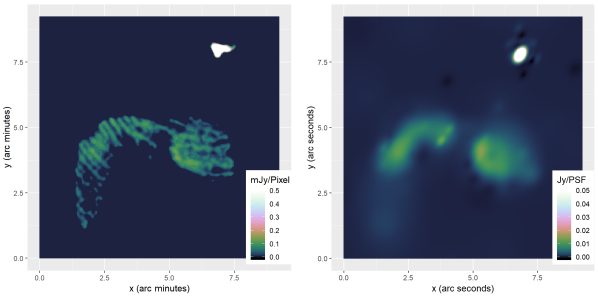


(a) Multi-scale CLEAN



(b) Coordinate Descent with elastic net.

Figure 9: Influence of calibration errors



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

Figure 10: Comparison to briggs-weighted multi-scale CLEAN as Ground Truth.

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, 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 de-

scent. 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 [6, 8]. 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.

3.2 PSF approximation with parallel coordinate descent

First, we test the effect of our PSF approximation method on the parallel coordinate descent algorithm. We test the deconvolution with increasingly smaller windows around the center of the full PSF. We measure the wall-clock time of the asynchronous iterations with $\tau = 8$ processors. We measure the value of the objective function after a batch $8 * 1000$ asynchronous iterations.

Note that the tests in this section are performed using the parallel block coordinate descent algorithm. It is able to group pixels into blocks, and minimize several blocks in parallel in each iteration. But grouping pixels into blocks slowed down the performance of the parallel block coordinate algorithm. The results of these tests can be found in the attachments.

Figure 11 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 excluded the time spent in the gridder or FFT, and only measured the time a batch of $8 * 1000$ asynchronous iterations).

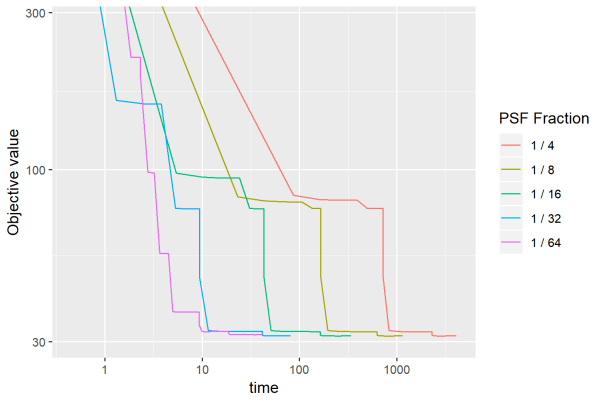


Figure 11: Convergence times with PSF approximation

The Figure 11 shows one line representing the value of the objective function after each batch of asynchronous iterations. Each line contains discontinuities. They either represent a Major cycle, or a

PSF	ESO	Total seconds
1 / 4	2.750	4094
1 / 8	1.437	1148
1 / 16	1.109	339
1 / 32	1.027	81
1 / 64	1.001	42

'Minor' cycle which was performed in-between the batch iterations. The wall-clock time of both the Major and 'Minor' cycle are excluded in this test. 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 81 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.

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 spends more time outside the asynchronous batch iterations(we excluded everything but the time spent in the asynchronous batch iterations) than with $\frac{1}{32}$.

3.2.1 Pseudo-random strategy and the Search Factor

We discussed in Section 2.2, a random pixel selection strategy seems to perform badly on the deconvolution problem. We created the pseudo-random strategy, which selects a pixel at random, but searches in it's neighborhood for the optimal pixel 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 pixel has been selected, we search 0% of the neighboring pixels in the active set. It is identical to a pure random selection strategy. A Search Factor of 1.0 says that after a random pixel has been selected, we search 100% of its neighbors in the image. For example, if we have 256 pixel with 8 threads, a search percentage of 1.0 selects a random pixel and looks at 32 pixel in its neighborhood. A search percentage of 100% is identical to a greedy strategy, where each thread searches its partition of the entire image.

Our parallel coordinate descent algorithm needs a Search Factor larger than 0. We compare different Search Factors in Figure 12.

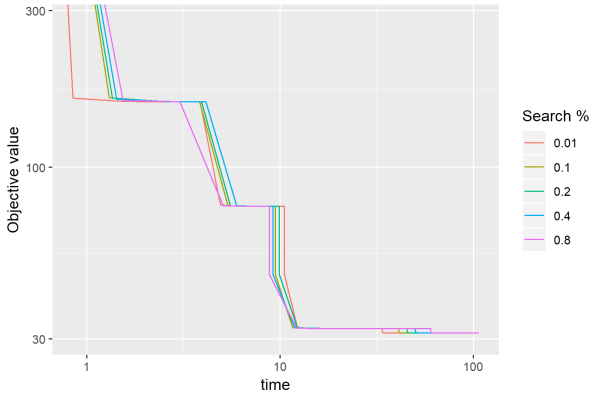


Figure 12: Convergence times with different search percentages.

Search Factor	Total seconds
0.01	83
0.1	81
0.2	83
0.4	89
0.8	106

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.

The parallel coordinate descent algorithm is not deterministic. From each run to another, the algorithm selects different pixels at different points in time. This leads to an additional source of variance when measuring the total seconds to converge. In this project, we have not systematically tested the convergence time variance. The results presented in Figure 12 are from a single experiment run, where a Search Factor of 0.1 lead to a slightly faster convergence time. This can change from experiment run to experiment run. At different runs had either the Search Factor 0.01, 0.1 or 0.2 as the fastest.

We fixed the search percentage tuning parameter to a value of 0.1 for all tests in this project. Based on this test, it does not have a significant influence on the convergence time, if the Search Factor is chosen small enough. It was enough to alleviate the problems a pure random strategy introduces in the deconvolution problem (discussed previously in Section 2.2). 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 different observations.

3.3 Comparison to the serial coordinate descent algorithm

In the previous section, we tested the convergence speeds of the parallel (block) coordinate descent algorithm with different tuning parameters. We showed how it achieves significant speedup with our PSF approximation scheme, and showed the influence of the Search Factor tuning parameter. We excluded the negative tests: Grouping pixels into blocks did not lead to a faster convergence time. Also, using gradient acceleration [4] did not result in faster convergence times. These tests can be found in the attachments.

In this section, we compare the serial coordinate descent algorithm with the final, simplified parallel coordinate descent algorithm. We took the lessons from the previous 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 faster than the previous implementation.

In this test, we compare the total wall-clock time spent on the deconvolution algorithm. Previously, we only measured the time spend in a batch of asynchronous iterations. This test also includes the time spent on auxiliary tasks in the deconvolution algorithm, and only exclude the time spent in the Major cycle.

We use the same hardware as in the tests of Section **??**. The table 1 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.

Alg.	PSF	Major	Seconds	Speed
Serial	$\frac{1}{16}$	4	1486	–
Parallel	$\frac{1}{32}$	5	75	≈ 20

Table 1: 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 3.2. 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 about three times more than the par-

allel coordinate descent algorithm .

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 13 compares the reconstruction of the two algorithms 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.

3.4 Scalability of the parallel coordinate descent algorithm

The parallel coordinate descent algorithm outperforms 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[3]. PCDM has been extended for the distributed setting as Hydra[9], and in its accelerated variant as Hydra^2[10]. 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 14 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 14 needs over 3 million parallel iterations to converge with 32 asynchronous processors. With 8 processors, it needs 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 14 does not increase as drastic as the total number of iterations needed.

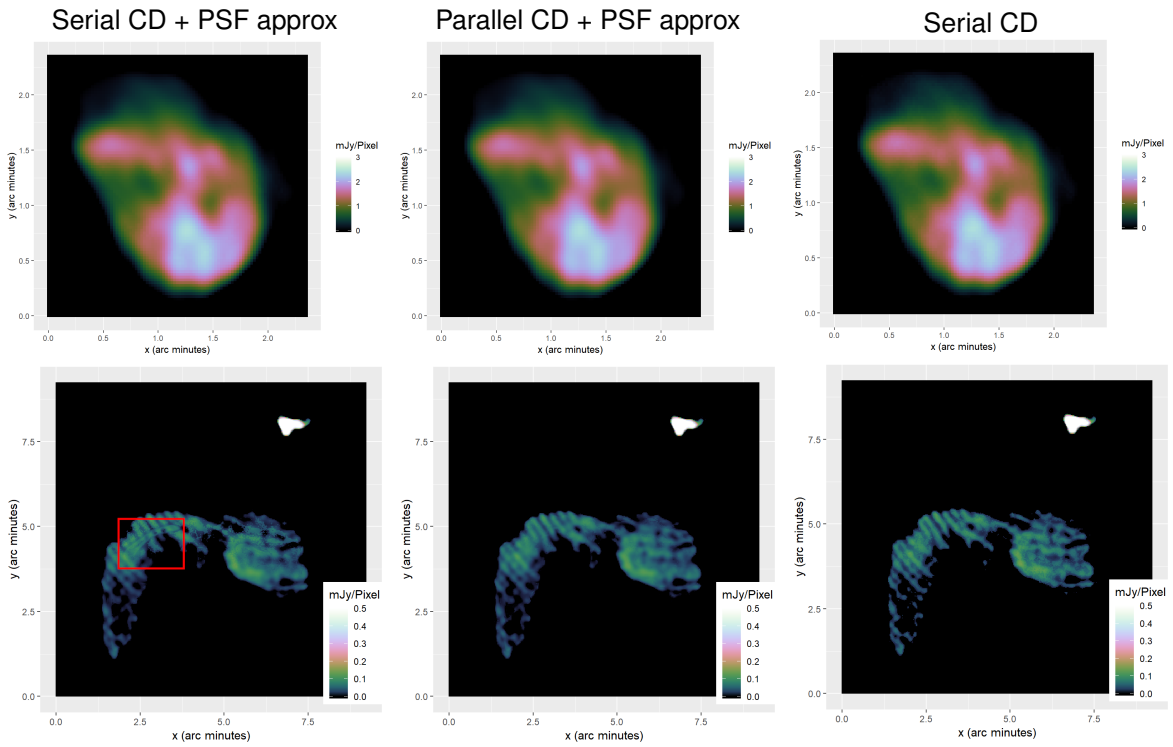


Figure 13: Comparison of the serial and parallel coordinate descent reconstruction of the LMC observation.

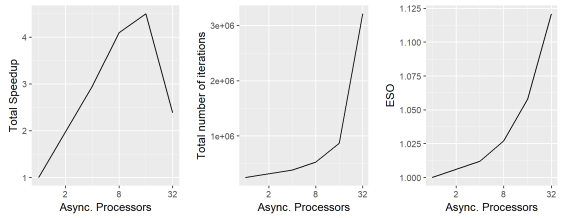


Figure 14: Speedup comparison with increasing number of asynchronous processors.

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

Table 2: Throughput of the parallel coordinate descent algorithms with additional processors.

When we look at the throughput of the parallel coordinate descent algorithm in table 2, 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 14 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 runs a batch of asynchronous deconvolutions: Each asynchronous processor chooses pixels from the image for a given number of iterations, independently of the other processors. 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 batch of asynchronous iterations 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 image. With an image of 1024 entries, 32 processors and a Search Factor of 0.1, each processor searches 10% of $\frac{1024}{32}$ of the image. By adding more processors, each processor searches through fewer entries, although the total number of entries searched stays the same. By adding addtional processors, each single processor acts more according to a random selection strategy. The solution in that case would be to increase the Search Factor tuning parameter.

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 Factor. The results are summarized in Table 3.

Increasing the Search Factor 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. With a larger Search Factor, the parallel coordinate descent algorithm is faster with 32 processors than with 16, and we removed the 'dip' in Figure 14. 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 14.

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 pixels in the image. For a Search Factor of 0.1, the algorithm always searches 10% for every parallel iteration. Adding more processors simply reduces the number of pixels each processor checks. The results from Table 3 suggests that each processor in our parallel algorithm has to search through a minimum number of pixels in the image to run efficiently. But this view does not agree with our results from Figure 12, where we used 8 processors and a Search Factor of 0.01. Each of the 8 processors searched through fewer pixels than in this test with 32 processors and a Search Factor 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 Factor 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.

4 Discussion and Conclusion

Test	Processors	ESO	Search Factor	# 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 3: Speedup comparison for the parallel coordinate descent algorithm, once with increased ESO and once with increased Search Factor.

References

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

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

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

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

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