# Gridding in Radio Astronomical Image Reconstruction

Urvashi R.V.

December 10, 2002

## Contents

# 1 Introduction

Gridding in Radio Astronomical Data Analyis involves transfering irregularly spaced data onto a regular grid. Data rates and volumes are extremely high, and the operation of gridding is performed frequently.It hence appears as a bottleneck in the entire data processing pipeline. Speeding up the gridding algorithm by parallelizing it, is expected to significantly improve the throughput of the entire data processing pipeline. Straightforward data parallelization, although expected to be effective, will not be optimal. This is because in real time data recording and analysis, the sorting order of the incoming data, is not optimal for the gridding operation, and this leads to concerns about cache locality.
This report discusses the operation of gridding, and emphasizes certain aspects of the procedure, relevant to the design of the gridding algorithm as well as a few directions to take in efficiently parallelizing it. A KeLP implementation of this algorithm with some straightforward data parallelism is also discussed.

# 2 Gridding in Radio Astronomical Data Processing

## 2.1 Imaging - Gridding and Deconvolution

Let $I$ represent a 2 dimensional brightness distribution in the sky.
A Radio Interferometer samples the Spatial Fourier Transform $V = FT(I)$ (2D visibility fn) of $I$, at positions specified by an irregular two dimensional sampling function $S$
The measured spatial frequency components (visibilities)

$$V^M = V \ S \tag{1}$$

Taking 2D inverse FT,

$$I^D = I \star B \tag{2}$$

$I^D$ : `dirty` image - image obtained via direct IFFT of measured $V^M$
$I$ : actual image - brightness distribution in the sky
$B$ : point spread function - transfer function of the telescope

Thus the `dirty` image $I^D$(measured) is a `convolution` of the Actual Image $I$(unknown) and the Point Spread Function of the telescope $B$(known). A `deconvolution` has to be performed to obtain a good estimate of the actual image $I$.

To perform deconvolution, or any FFT, the 2D data has to be regularly spaced, but the raw sampled visibilities are irregularly spaced on the 2D spatial frequency plane.

Gridding involves transfering and accumulating several lists of irregularly spaced data, onto a regular grid.

Deconvolution algorithms are usually iterative, and work with 2D images (function evaluations at each point,2D FFT/IFFT), usually $1K \times 1K$. This can extend to $8K \times 8K$ (Extended VLA, ALMA). These are non-linear deconvolutions, since $B$ is a singular matrix. Some more complex deconvolution algorithms, repeatedly transfer data between the regular and irregular representations,( gridding and regridding ) and this requires that the gridding implementation can be optimally performed.

## 2.2 Sampling Function - UV Tracks

In Radio Interferometry, the irregular sampling function $S$ in the spatial frequency domain, takes the form of a large number of elliptic arcs on the UV plane. The origin of these tracks on the UV plane, suggests a possible data ordering, that may be used in the design of an optimal algorithm.

In one timestamp (time interval over which data is averaged - typically few seconds), each pair of antennas on the ground,measures one spatial frequency component, thus contributing one point, to the sampling function $S$. With $N$ antennas there are $N(N-1)/2$ such pairs, and correspondingly as many points on the UV plane. The position of these sample points, depends on the distance between two physical antennas (for each pair), projected onto a plan perpendicular to the line of sight from a reference antenna to the astronomical radio source being observed. This means that the position of the sample point depends on the antenna spacing, in the direction of the radio source.

At any given timestamp, there are thus $N(N-1)/2$ sample points. As the earth rotates (time changes), the projected spacing between the antennas, constantly changes, with respect to the source (considered to be fixed in the sky). This gives rise to sets of sample points at slightly different positions, on the UV plane. These positions (time varying) are correlated and can be calculated for a given antenna array, direction of the source being observed, and the time of day.

Each pair of antennas, thus forms a track on the UV plane. These tracks are ellipses, whose centres and dimensions, depend on the actual position of the two contributing antennas.

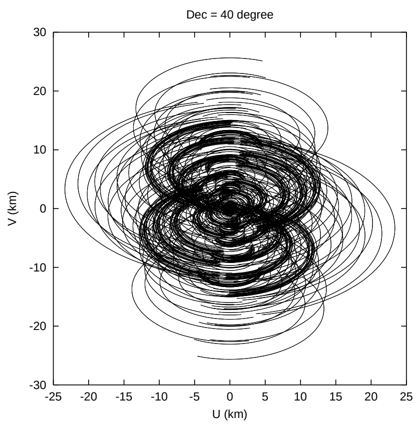


Figure 1: Irregular Sampling Function on the UV plane

The equations of these ellipses are as follows.

$$u^2 + \left( \frac{v - (L_z/\lambda) cos\delta_0}{sin\lambda_0} \right)^2 = \frac{L_x^2 - L_y^2}{\lambda^2} \tag{3}$$

where $\delta_0$ is the hour angle (time) and $\lambda$ is the declination ( from location latitude). This is calculated via the following matrix that transforms antenna pair positions to corresponding UV plane coordinates.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \sin H & \cos H & 0 \\ -\sin\delta\cos H & \sin\delta\sin H & \cos\delta \\ \cos\delta\cos H & -\cos\delta\sin H & \sin\delta \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4}$$

The UV plane (visibility plane) is the Spatial Fourier Transform of the image. The image is a real function, hence the visibility is Hermitian. $V(u, v) = conj(V(-u, -v))$. Hence, additional sample points on the UV plane have to be obtained by reflecting the points about the origin. This is to effectively sample the complex conjugate of the actual incoming data values.

Hence, as time varies, the elliptic tracks created by each pair of antennas, build up to form a partially filled UV-coverage, which is the irregular sampling function $S$. This sampling function has to be transferred to a regular grid, to facilitate the application of FFT algorithms to the data.

## 2.3 Data Rate and Sorting Order

Radio Interferometry works with pairs of antennas (interferometers,visibilities).

$N$ antennas form $N(N-1)/2$ pairs (interferometers) $= N_{ifr}$.

Consider each PAIR of antennas. The time varying signal from the feed of one antenna is multiplied with that of the other (correlation), to produce ONE broadband time varying signal per interferometer. This gives 2D data $N_{ifr} \times N_{time}$.

These broadband signals are sent through sets of filterbanks, to split the bands into $N_{chan}$ narrower bands (channels). This gives 3D data : $N_{ifr} \times N_{time} \times N_{chan}$.

Each antenna has two cross polar feeds and hence two output time varying data streams (R & L). Correlation coefficients between a pair of antennas can be of 4 combinations ( RR,LL,RL,LR ), $N_{polzn} = 4$.
Hence, 4D data : $N_{ifr} \times N_{time} \times N_{chan} \times N_{polzn}$ ( all complex numbers )

Each integration timestamp is typically around thirty seconds, and this decides the final data rate. The following are the sort orders of incoming data vs. that required for gridding.

1. `Incoming data` : $Time \rightarrow Interferometer \rightarrow Polarization \rightarrow Channel$
   Hardware Optimized.

2. `Gridding` : $Polarization \rightarrow Channel \rightarrow Time \rightarrow Interferometer$
   For each time step, different sets of locations on a regular 2D grid have to be accessed. The two dimensions $N_{time} \times N_{ifr}$ after gridding, become 2D spatial frequency co-ordinates $u \times v$.

## 2.4 Gridding Algorithm

Gridding is done on the data, separately for each frequency channel, and polarization. The data set required for one instance of gridding is a 2-dimensional array of interferometer Vs time. Considering row major ordering, this data is usually stored with each timestamp on each row.

For each timestamp, the data entries are UV values corresponding to different pairs of antennas. The mapping between the index of the list of UV values and the antenna pair it corresponds to, is known and fixed.

This 2D array is traversed, and for each UV value, the corresponding antenna pair position co-ordinates are used to calculate the position on the UV plane, where this data point has been sampled.

This position can be analytically calculated from the equation of the ellipse given earlier, or more directly from a the co-ordinate transformation matrix used to derive the equation of the ellipse.

This analytic UV co-ordinate now has to be mapped onto one grid point in the target regular grid. This can be done in several ways. The most straightforward way would be to find the regular grid point nearest to the calculated position, and store it there. If multiple points similarly get mapped to the same grid location, the values are averaged.

The method usually used is a variant of this, which uses a gridding function. A circularly symmetric gaussian-like function (normalized to peak = 1) is 'placed' at each grid point. Any data value whos position is calculated to be within a certain radius of the grid point, is multiplied by the value of this gridding function, at that point. This is done for all such data nearby data points, and these scaled values are then averaged into the grid location. In this manner - all UV values are placed on the regular grid.

In practice, it is necessary to just add up the scaled values as and when a UV value maps onto each location, and the count of the number of values per grid point, is also stored. After all UV values have been mapped onto the grid, the values at each grid point are divided by the corresponding count values, to average them.

The best gridding function ( from an aliasing point of view ) in use, is a prolate spheroidal function. This is expensive to evaluate, and this algorithm requires repeated evaluations at various points. To get around this, a look up table is used, which holds values of this gridding function, upto a certain radius about its centre, sampled at avery high resolution. This way, when each UV value has to be multiplied by the value of this function at various points, the distance of the analytic position , from the actual grid point, is used to index into this look-up table.

## 2.5 Current Implementations and Bottlenecks

Currently most implementations work with a data ordering according to the incoming and stored data. The slowest varying index in this ordering is the Time. This means that for each timestamp, the algorithm sweeps over all antenna pairs, and calculates positions and maps the values onto the grid. Successively accessed grid positions, will usually not be very near each other, which would mean that for each timestamp, data accesses are done almost at random, not taking into account the possibility of locality. This method does NOT take into consideration the fact that each separate antenna pair, forms elliptic tracks in time. Successive positions along each track, are most likely to be adjacent to each other, and if the data accesses can be managed this way - locality can be used to speed things up. For example, if the data is stored, such that the Time is the fastest varying index, this could be done.

An alternate method of using locality is to reorder the input 2D array, or divide it into bins, so that within each bin, all UV locations calculated, are near each other on the final target grid. This would introduce the additional operation of having to transfer the data into these bins, but this has to be analysed and a possible trade off can be arrived at.

# 3 Parallelizing the Gridding Algorithm

Parallelizing this gridding algorithm requires handling data in different optimal sorted order. A few ways of doing this could be as follows.

1. `Data Parallel Gridding - Time` Divide the data in Time. Grid several time chunks separately, and then combine them at the end. This way - while sending the data to separate processors - they can also be re-ordered, to suit the requirements of an optimal gridding routine. This would however still hold memory required for the entire dimensions of the target grid, per processor.

2. `Data Parallel Gridding - other dimensions` The data is divided along some other dimension - say along the interferometer ( antenna pair ). This way too - each processor would hold a copy of the entire grid, but would grid only an irregular subset of the tracks.

3. `Using threads and common memory` Spawn several threads - one for each pair of antennas ( each track ) or a group (bin) of antenna pairs. A single copy of the target grid could suffice here. In this case, concurrent data access conflicts are bound to occur and will have to be handled accordingly.

4. `Interleaved Parallelism` Reorder the data to make use of locality in the target grid, and divide the data between processors, such that each processor has to grid only a part of the entire grid (small regular chunk of the target grid). The advantage of this method is that the size of the region being gridded on each processor is much smaller than the total dimensions of the final target grid. The downside is that certain calculations have to be performed to split the data in this way and again, a trade of will have to be worked out.

5. `Real Time Gridding` For real-time gridding, any of the above variants can be used, but the slowest varying index of the data will unavoidably be the Time. The data can however be accumulated (in one place or separately on each processor), until there is enough, to perform efficient execution ( separate execution on one timestamp at a time will just reduce to the serial algorithm, and accumulating too much will mean large idle time ). This can be pipelined, such that as one chunk is being processed, the next chunk is being accumulated from the real time data acquisition system. The chunk sizes will have to be decided depending on the incoming data rate, processor speedup, and memory. This pipelining can also be done per processor - with separate threads for data I/O and computation.

# 4 KeLP Implementation of Data Parallel Gridding

The first variant of the gridding algorithm, was implemented in KeLP.

## 4.1 Implementation

This application has been implemented in KeLP, and involves some basic parallelization, with optimization based on the column major array ordering in KeLP.

1. The input data is in the form of an XArray of 3D grids. ( Nifr x Ntime x 2 ). Data values are ramdon numbers between 0 and 1.

2. The input XArray is split across processors, along the Ntime axis. This was done so that each processor locally stores data corresponding to several timestamps. The iteration ordering within the node, is then handled optimally, by making the Time axis the fastest varying one. The data was stored into the arrays so that the KeLP indexIterators would traverse the data arrays in this optimal order.

3. The target grids ( Nu x Nv x 3 ) on each node, form an XArray split into chunks as above.

4. On each node, the target positions of each input data value is calculated, and the data value is then copied into this position in the target grid, incrementing a counter for that particular grid position.

5. Locality in the target grid is used by calculating the UV grid positions of each baseline for the whole range of timestamps, as these will go into almost contiguous positions on the target grid. The way in which the initial data is stored in the input XArray is also based on an ordering in which the fastest varying co-ordinate is the time.

6. The target grids on each node - correspond to different chunks in time. Target grids on each node must then be combined to form the final grid. Values at corresponding positions must be added and counters incremented - to be able to calculate a total average value at each grid point. This is done with a call to mpReduceAdd().

## 4.2 Results

1. Speedup - Running times with fixed overall problem size

   The program was run with different fixed workloads, each with 1 to 6 processors, and the running times were noted.

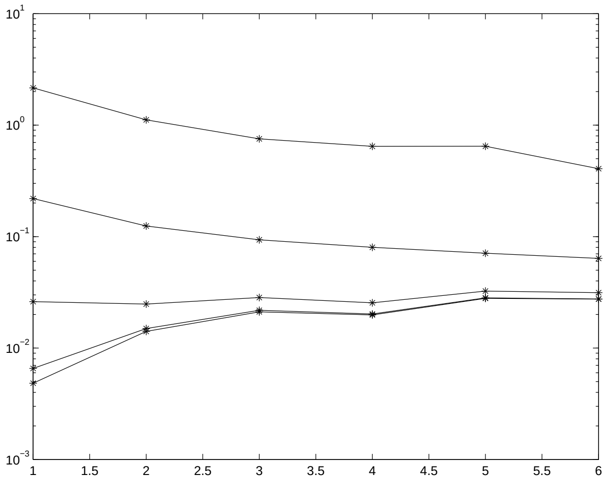


Figure 2: Running times with fixed overall problem size

```
prob size (N) = no of timestamps, divided equally between P procs
-----------------------------------------------------------------------------------
procs :    1             2              3              4             5           6
-----------------------------------------------------------------------------------

10      0.00483799    0.014107       0.021111       0.019812      0.027884    0.02747

100     0.00657105    0.0149751      0.021816       0.020228      0.028216    0.027563

1000    0.026114      0.024808       0.028399       0.0254461     0.032439    0.031415

10000   0.219588      0.124586       0.093522       0.080222      0.070907    0.063704

100000  2.15901       1.11439        0.752981       0.645867      0.647134    0.406188
-----------------------------------------------------------------------------------
```

This data shows that for small problem sizes, the serial program runs faster, and parallelizing it only increases the amount of communication, which increases the total running time.

For a small range of problem sizes, there is no speedup and the serial and parallel programs have approximately the same running times.

For large problem sizes, speedup is clearly observed, as the computation now begins to dominate over the communication required to combine all the grids at the end of the execution.

2. Scaled Speedup - Running times with fixed load per processor

   The program was run using 1 to 6 processors, but the problem was scaled to ensure fixed loads per processor. The following results were obtained.

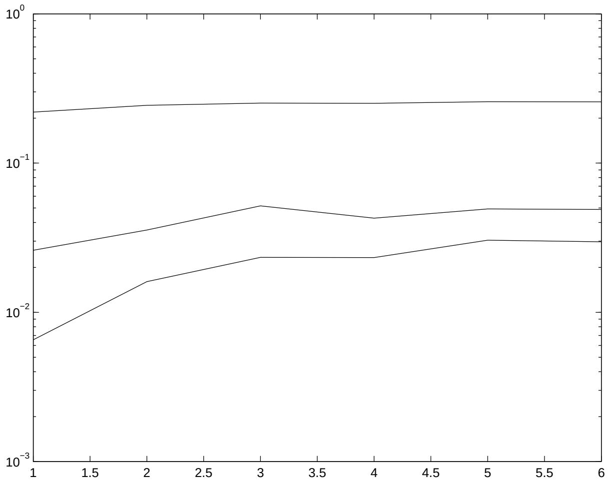


Figure 3: Running times with fixed load per node

Scaled Speedup

| procs : N | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 100 | 0.006549 | 0.0160639 | 0.023356 | 0.023257 | 0.0304279 | 0.029686 |
| 1000 | 0.026036 | 0.035604 | 0.0516421 | 0.042788 | 0.049266 | 0.0488831 |
| 10000 | 0.219566 | 0.24392 | 0.252326 | 0.25138 | 0.257649 | 0.257305 |

From the data, it can be seen that communication plays a significant role, during the final combination of the grids. The load per processor was kept fixed, for each of the curves, thus keeping the computation per node approximately the same, and bringing out the effects of communication. This is consistent with the trends observed with the simple speedup curves.

3. Input and Output Grids

   The following plots show the computed grids on two nodes, and then the combined grid.

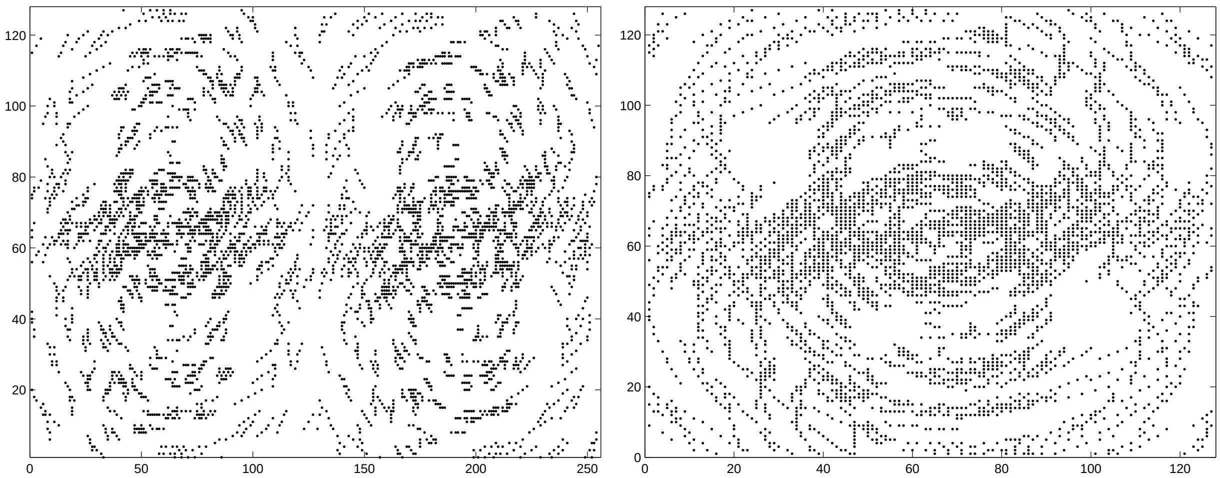


Figure 4: UV Coverage - Separate grids on each node & Combined final grid.

The first plot is in two halves - the first belonging to processor 0 and the second, belonging to processor 1. Each of the tracks, consist of half the total number of timstamps. The first half holds the first half of the timestamps, and the second half holds the second half of the timstamps.

The second plot shows the combined grid - obtained at the end. It can be seen that the tracks are now longer, and are a combination of the two grids from the first plot.

### 4.3 Further Work

1. This current implementation is a basic demonstration of the method. To be effectively tested on real data, it has to be adapted to standard data formats, as well as optimized to handle the other data dimensions ( channel,polarization ), according to the way it's stored.

2. The other variants have to be explored and optimized, and newer ideas have to be thought up.

## 5 Conclusion and References

This work has been a basic study of the gridding algorithm and various possible parallel implementations. The simplest variant has been implemented in KeLP and analysed.

References :


1. KeLP Reference Manual

2. NRAO lectures on Synthesis Imaging