Evolution of the `imager` performance: Gridding optimization

Sanjay Bhatnagar

September 29, 2003

Performance of the `imager` for full IQUV imaging is poorer than expected. The gridding/de-gridding operations now dominate the run time profile of `imager`. In general, this is a healthy profile since these operations constitute the dominant cost of most image reconstruction algorithms. Further optimization of `imager` therefore must concentrate on the optimization of the gridding/de-gridding processes where I/O costs constitutes the dominant cost.

The issues concerning optimal tuning of the Table system parameters, tuning of the various Tile sizes and shapes (for MS and images), interaction of the Table system buffering with the OS I/O buffering, etc. can be effectively addressed only after this higher level optimization is in place. This report documents the results of the investigations for optimizing the gridding/de-gridding operations and suggestions for changes in the code to improve the performance.

1 Full-IQUV imaging

Full IQUV image reconstruction in AIPS++ is now done independently on each of the individual Stokes planes. 4-plane image cubes are constructed for the images and the PSF. All Stokes planes are gridded simultaneously and then Fourier transformed independently to construct the PSF and the Dirty/Residual Image. For computing the Dirty Image and the PSF, the visibility data is read twice - once to grid the weights and once to grid the visibilities. Data for all Stokes per row in the Measurement-Set (MS) are read simultaneously. All planes of the 4-plane image cube per pixel are therefore required while gridding (or de-gridding) and that in turn dictates the memory requirements for full IQUV gridding. If the total size of the cubes exceed the memory resource value set by the user in `.aipsrc` file, the process of gridding (de-gridding) switches to what is referred to as `tilled-gridding`. In this state, individual tiles of the image cubes are read in and out of the disk depending upon the need. Consequently the process if significantly slower in this state.

In the iterations of the deconvolution process, the dominant cost comes from the residual visibility computation. This requires a Fourier transform and de-gridding (of the model image), residual visibility computation, and the gridding and inverse Fourier transform (of the residual visibilities). This is done once per major cycle of the CS-Clean type of algorithms. The cost of the residual computation is therefore close to the cost of two gridding operations. The cost of each gridding operation itself is proportional to the number of visibilities and the image size dependent cost comes dominantly from the cost of FFTs (which is comparatively much smaller when the number of visibilities is large or comparable to the number of pixels in the image).

The total cost of gridding and de-gridding images and PSF is shown in Fig. 1. As expected, the cost of gridding and de-gridding images is comparable to each other. However there is a constant extra cost of ~ 2 times that of PSF gridding cost, plus possibly an image size dependent cost.

2 Gridding/de-gridding

The break-up of gridding the PSF and gridding/de-gridding the images is shown in Figs. 2 and 3. Gridding in AIPS++ is done as:

```
SkyEquation::initializePut()
  {
    GridFT::initializeToSky();
  }

for all Vis
  SkyEquation::put()
    {
      GridFT::put();
    }
```

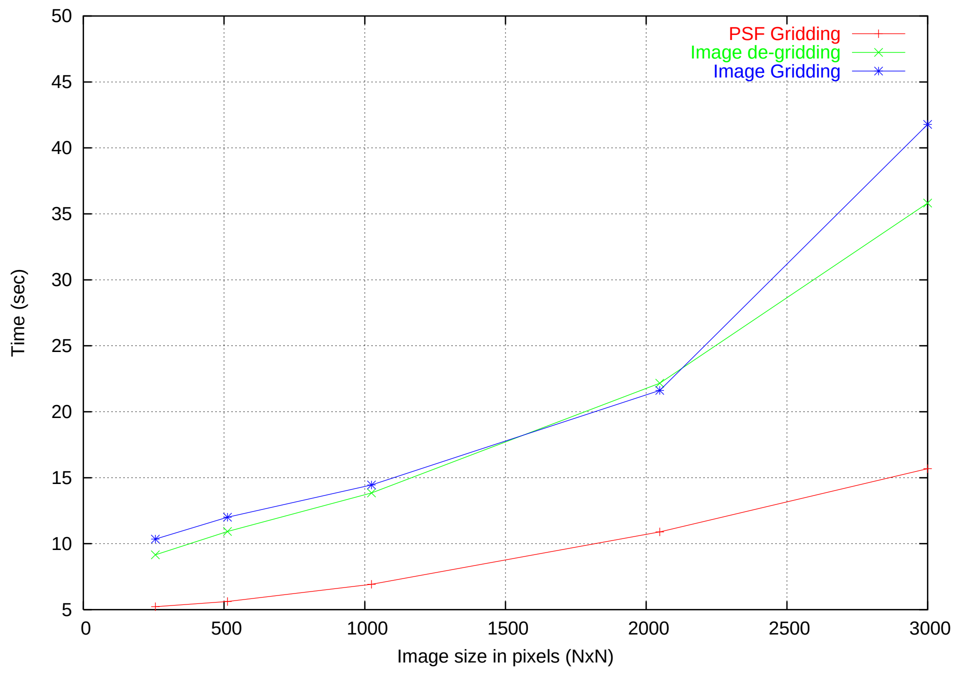

Figure 1: Wall-clock run time for the total cost of gridding/de-gridding operation for the PSF and the images in AIPS++. The break-up of the costs of PSF gridding is shown in Fig. 2 and that for gridding/de-gridding of images is shown in Fig. 3. See Fig. 5 for a test case run with the code modifications discussed in detail in this document.

```
SkyEquation::finalizePut()
  {
    GridFT::finalizeToSky();
    Sum of imaging weights;
    applySkyJonesInv();
    applySkyJonesSquare();
  }
```

The `GridFT::initializeToSky()` method is a light weight inexpensive call (see Fig. 3). The `GridFT::put()` is the actual operation of re-sampling on a regular grid and its cost is shown in Fig. 3 (the green curve). As expected, this is only weakly dependent on the image size. The cost of the `SkyEquation::finalizePut()` call can be further split into the cost of

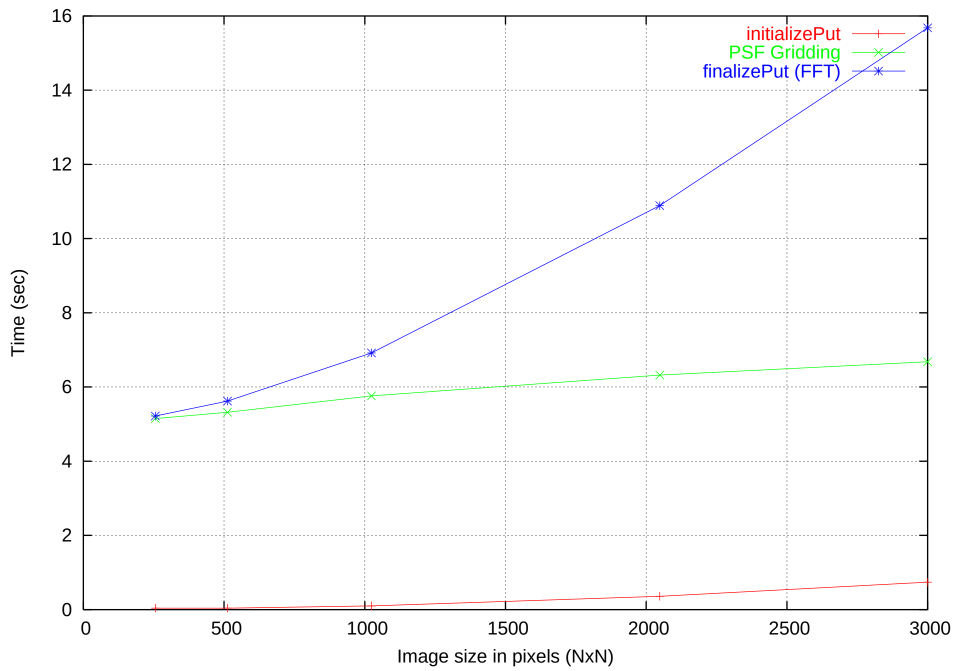

Figure 2: Cumulative wall-clock run time for the gridding of the PSF in AIPS++. The gridding operation is broken up into its major constituents: `initizliePut()`, `Gridding` and `finalizePut()`.

FFT (the `GridFT::finalizeToSky()` call) and application of Jones matrices as shown in the pseudo code above. The cost of the sub-components of `SkyEquation::finalizePut()` are shown in Fig. 4. The Sky Jones are not actually applied for the case of PSF computation, and the extra cost for PSF computation comes from conversion between complex and Stokes representations. Since no Jones matrix application is necessary, this cost can be eliminated. However in the overall runtime for imaging, this will be a minor saving only. The cost of `SkyEquation::finalizePut()` for residual images is significant. Since this is a recurring cost at each major cycle, savings here will have non-negligible impact on the overall run-time and needs to be investigated further. A specialization of these classes should retain this functionality for the application of the image plane corrections.

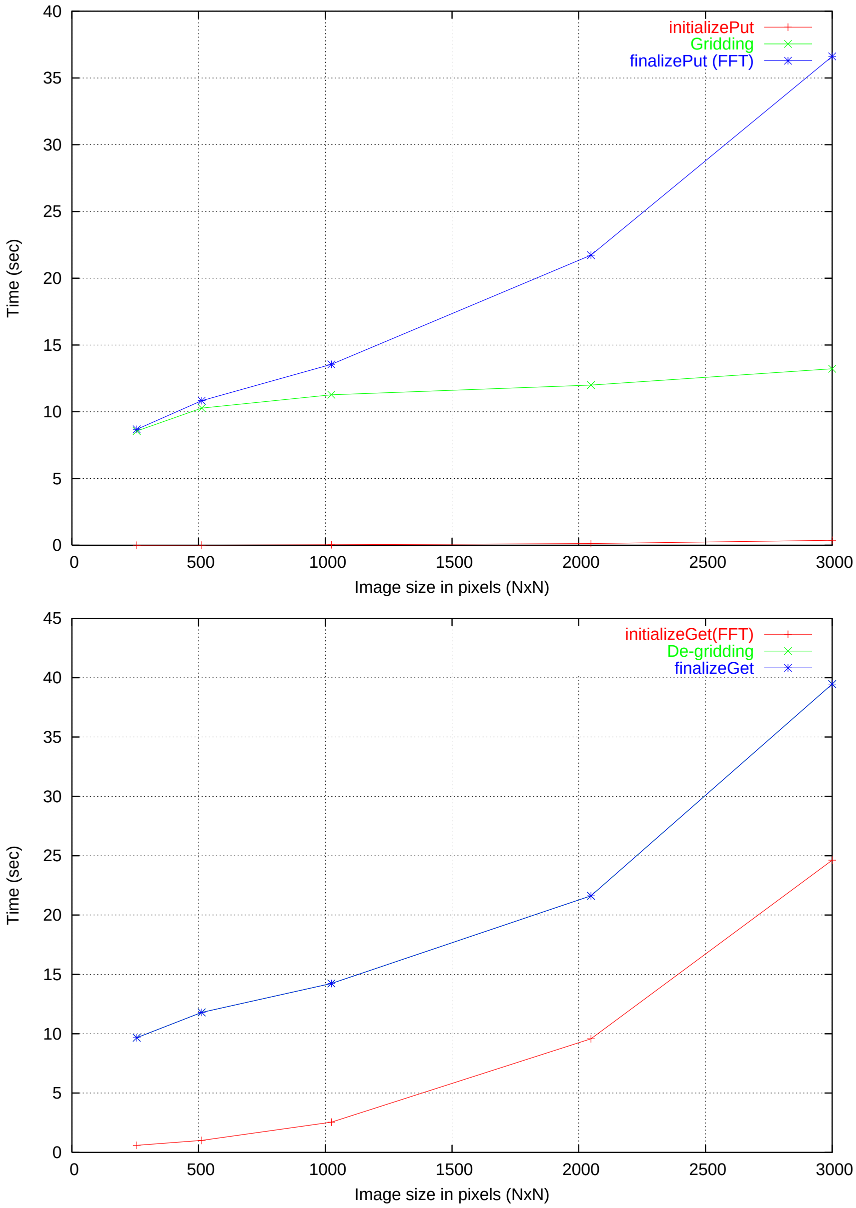

Figure 3: Cumulative wall-clock run time breakup for the gridding/de-gridding of images in AIPS++. The upper panel shows the various component for gridding while the lower panel shows the same for de-gridding operation. The gridding operation is broken up into its major constituents: `initizliePut()`, `Gridding` and `finalizePut()` (FFT cost). The de-gridding operation is broken up into: `initializeGet()` (FFT cost), `De-gridding` and `finalizeGet()`.

2.1 Residual computation

Each major cycle in deconvolution involves computation of the residual image. This is done in `SkyEquation::fullGradientChiSquared()` as follows:

```
SkyEquation::predict()
 {
   FT (Image);
   for all Vis Chunks
    {
      Read UVW and the modelVis column;
      De-grid onto the modelVis;
      Write the modelVis column to the disk;
    }
 }

SkyEquation::fullGradientChisquared()
 {
   predict();
   for all Vis Chunks
    {
      Read the modelVis column;
      modelVis -= Vis;
      Copy the modelVis column to the Vis column;
      Grid the Vis onto a grid;
      Write Vis to the disk;
    }
   finalizePut(), i.e. do the FT of the complex grid;
 }
```

As can be seen, by separating the loop over all visibilities for predicting the model visibilities and gridding them, we incur 3 extra disk I/O operations. Since the total cost of residual computation is dominated by the disk I/O, the extra cost here directly shows up as the degraded performance of `imager`. I believe this extra cost now dominates the difference between the run time for full IQUV imaging in AIPS and AIPS++.

AIPS++ MSs are stored as tiles and some smaller chunk of the entire data is read at a time into the `VisBuffer` object which is used as the data holder in the loop over all visibility chunks. The extra I/O operations

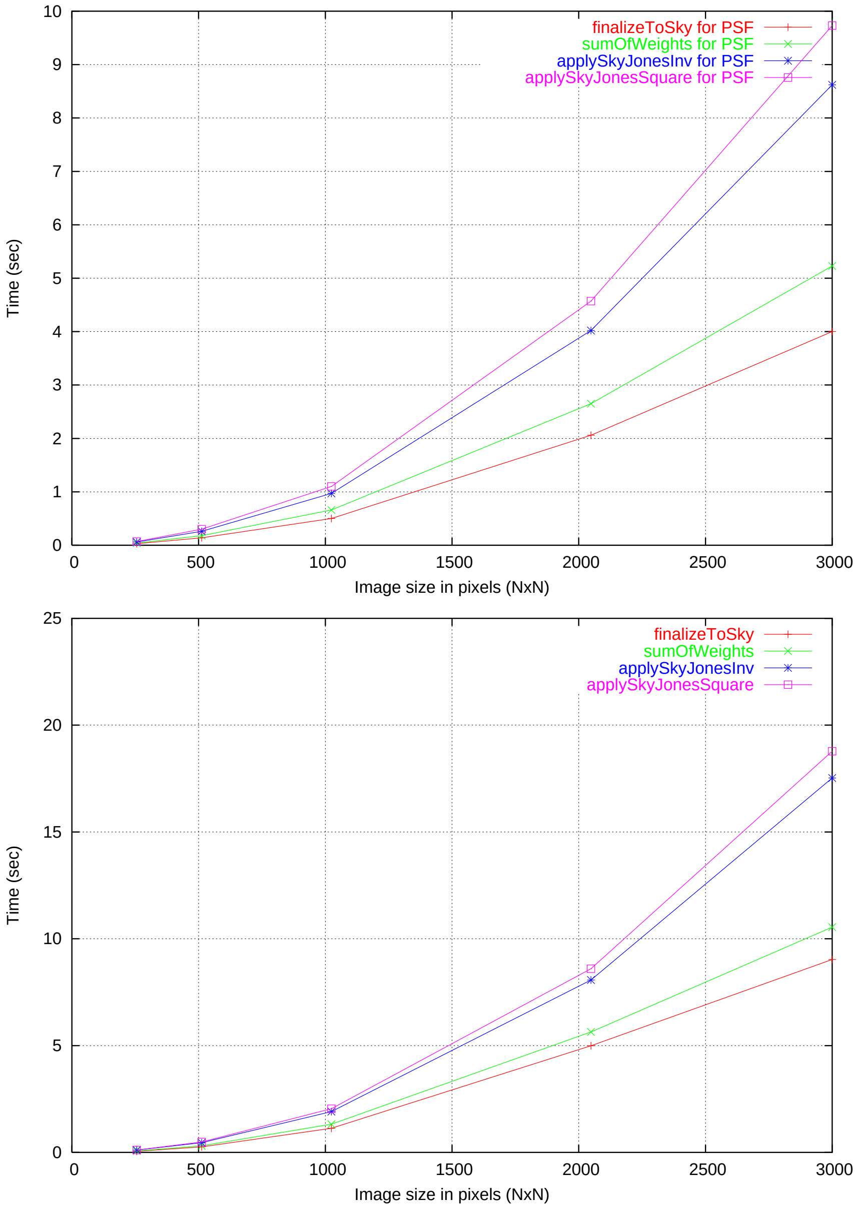

Figure 4: Cumulative wall-clock run time for the components of the `SkyEquation::finalizePut()` operation in AIPS++. The upper panel shows this cost for the PSF computation while the lower panel shows the cost for computing the residual image. The `finalizeToSky()` call performs the actual FFT. `applySkyJonesInv` and `applySkyJonesSquare` calibrates the gridded data. Each of these calls involve converting the data between complex and Stokes representations and the actual application of the metrics. The application of the metrics is not done for the PSF computation.

mentioned above can be eliminated by predicting the model visibilities, computing the residual visibilities and gridding them in a single loop as follows:

```
FT (Image);
for all Vis Chunks
  {
    Read UVW and the modelVis column;
    De-grid onto the modelVis;

    Read the modelVis column;
    modelVis -= Vis;
    Copy the modelVis column to the Vis column;

    Grid the Vis onto a grid;
    Write Vis to the disk;
  }
finalizePut(), i.e. do the FT of the complex grid;
```

The final writing of the residual visibilities to the disk should also be avoided in the major-minor cycle loops.

The downside of this is that in the `GridFT` class, one needs to hold two copies of the complex grid. This in-turn calls for optimization of how the `Lattice` classes handle large images in and out of the disk. For the purpose of gridding, it is probably already doing well by reading the required tile from the disk (*although the tile shape should span across the Stokes planes*). For FFT purposes, the optimum handling would be to read-in a full image plane into the RAM at a time.

A minor memory usage optimization can come by recognizing that all Stokes PSFs are same if the T_{sys} for the two orthogonal polarization signals is same and all polarization at all baselines are measured. However this in general will not be true - though probably true for majority of the cases for a fully debugged interferometer with identical elements. Ignoring minor differences in the T_{sys} for the two polarization or even the absence of full Stokes measurements at all baselines will not affect the quality of reconstructed image.

Preliminary tests to *just* measure the run-time with the above I/O optimization shows the expected improvements in the gridding/de-gridding operations (i.e., they become comparable to the cost of PSF gridding). The measured run-time for image de-gridding compared to PSF gridding is shown

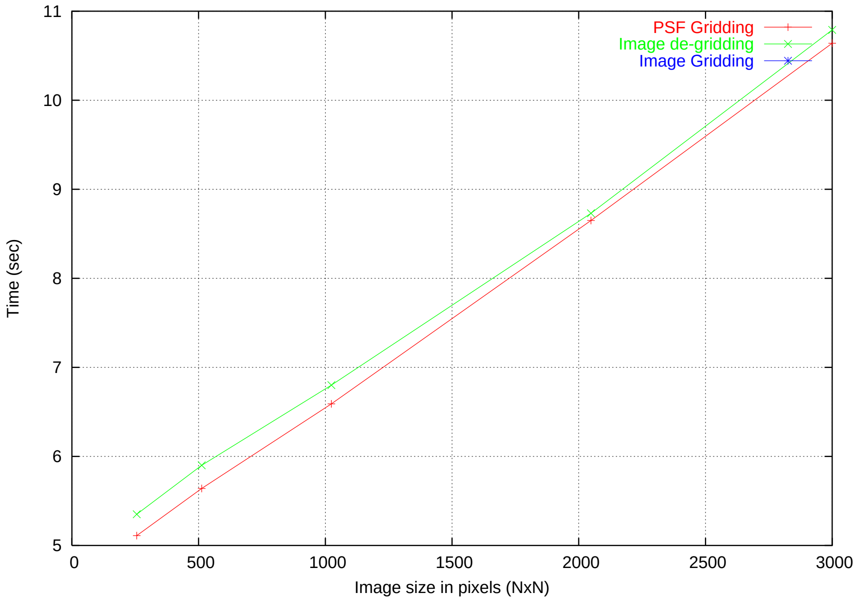

Figure 5: The wall-clock run time for PSF gridding and image de-gridding tests with the modified version of the `GridFT` class. The cost of de-gridding is now comparable to that of PSF gridding and corresponds to a factor of ~ 3.5 improvement at the largest image size. The modifications in the code, as of now, breaks the `imager` when gridding the residual visibilities and needs to be debugged $\smile$.

in Fig. 5. Due to programming errors, the `imager` with the modified `GridFT` class breaks down when gridding the residual visibilities and hence the curve for that could not be measured. I will continue to work on this programming problem (in my "free time"! $\smile$).

3 Memory requirements

The total size of the image cube ($4 \times N \times N$ float values for a single stokes image of size $N \times N$) is used in the `LatticeFFT` class to make decisions about memory model. If this size is larger than the `cache size` (which is really the user defined memory resource in `.aipsrc`), the decomposition of a 2D FFT into 1D FFTs is done in the `LatticeFFT` class. Consequently, the data from each row of the 2D image plane is passed to the `FFTServer`. This in turn requires more memory copies and does not use the optimization in the FFTServer -> LatticeFFT classes. It is therefore more expensive in this state.

Although this optimization is effective only at large image sizes, the memory requirements in LatticeFFT should be relaxed to the size of the single plane of the image cube. Since we never do more than 2D FFTs, and if the Lattice classes are optimized such that if only one plane of an image cube is required, it loads just that plane in the memory, all we need is RAM large enough to hold one plane at a time only (while doing FFTs).

The actual Lattice cursor shape used in the LatticeFFT class is derived from the Lattice<T>::advisedMaxPixels() and Lattice<T>::niceCursorShape() methods. However we compute the memory requirements from the shape of the image cube. Since the above mentioned relaxation of the memory requirements in the LatticeFFT class will crucially depending upon how the Lattice class handles tiled planes of an image cube, this optimization is probably best moved into these methods of the Lattice class such that it is reflected in the niceCursorShape().